ECE 312 (SP26):

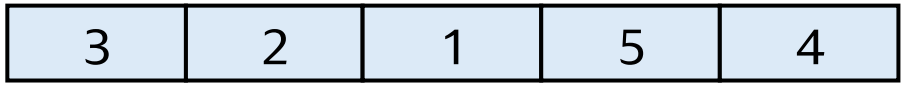
Sorting Algorithms

Neil Zhao

neil.zhao@utexas.edu

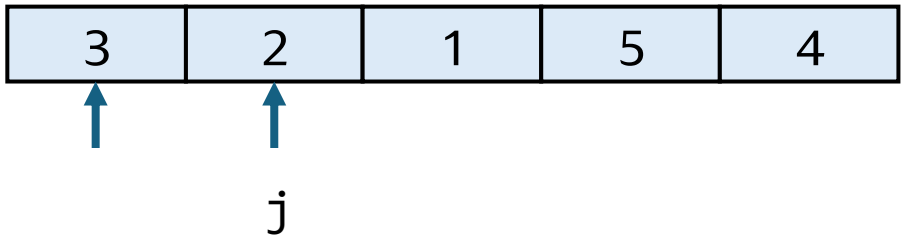
Bubble Sort Naïve

```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



Bubble Sort Naïve

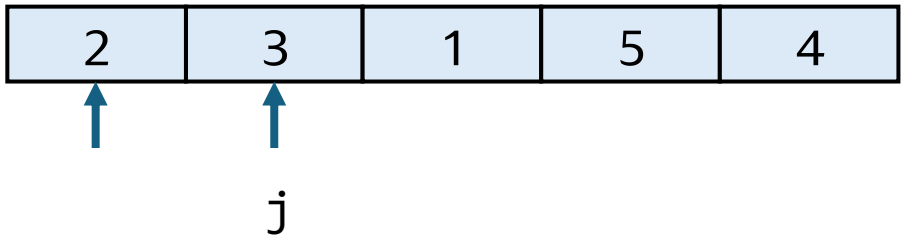
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



i = 0; j = 1

Bubble Sort Naïve

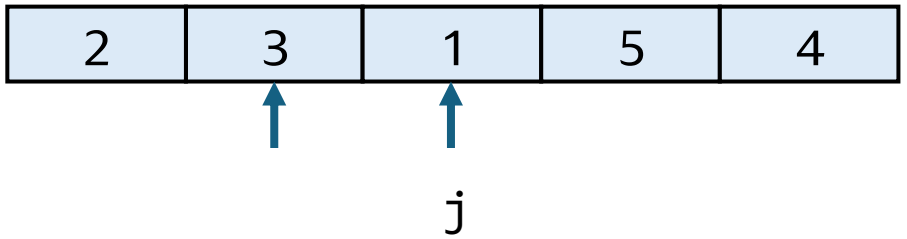
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



i = 0; j = 1

Bubble Sort Naïve

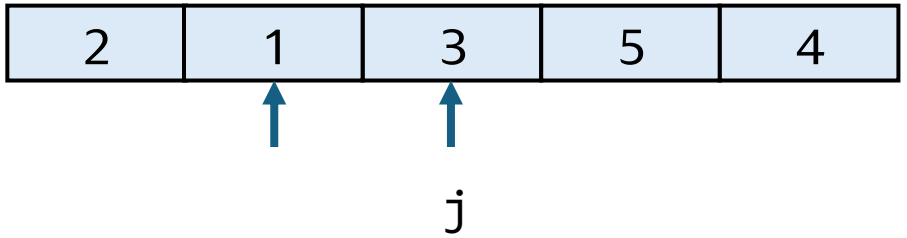
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



i = 0; j = 2

Bubble Sort Naïve

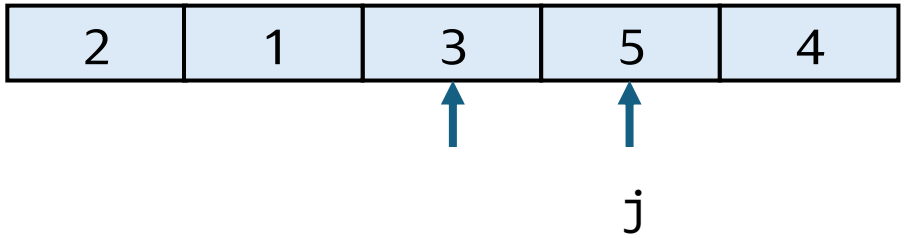
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```




```
i = 0; j = 2
```

Bubble Sort Naïve

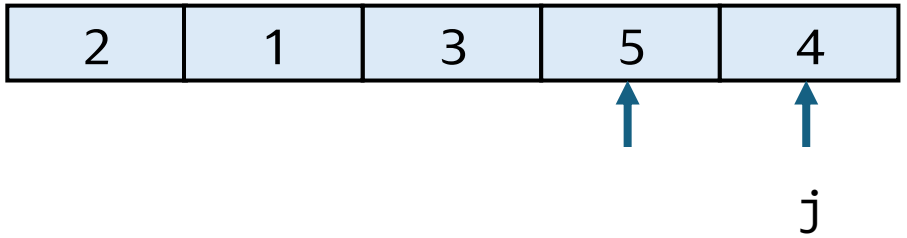
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```




```
i = 0; j = 3
```

Bubble Sort Naïve

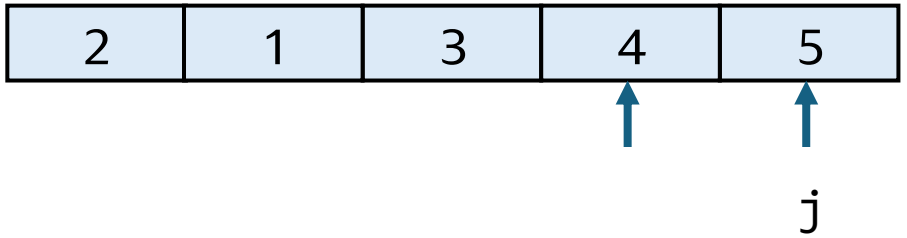
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



`i = 0; j = 4`

Bubble Sort Naïve

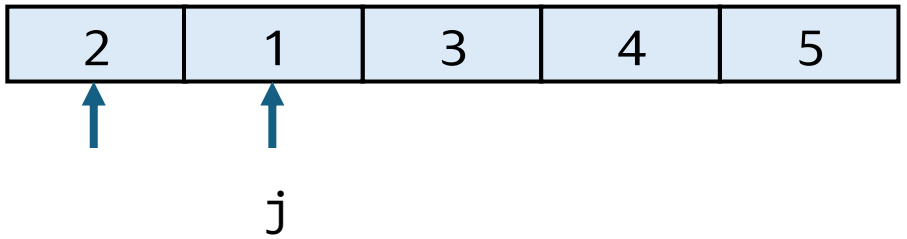
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



i = 0; j = 4

Bubble Sort Naïve

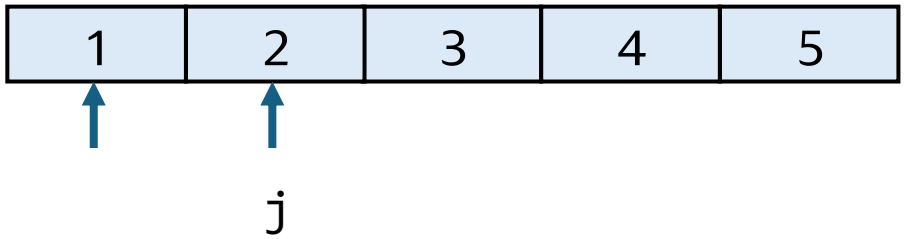
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



i = 1; j = 1

Bubble Sort Naïve

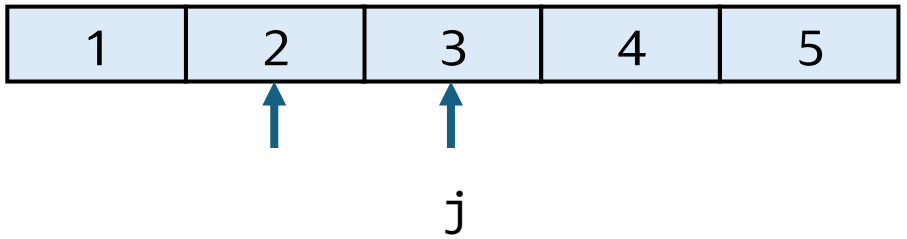
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



i = 1; j = 1

Bubble Sort Naïve

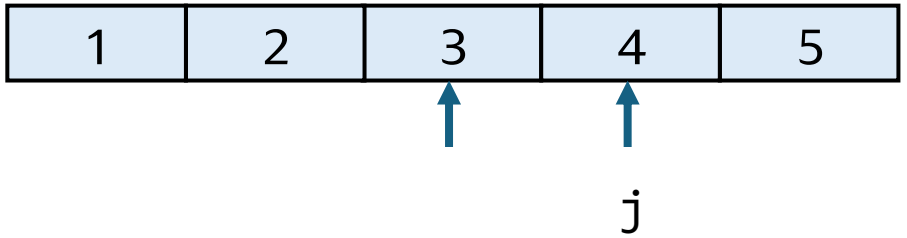
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



`i = 1; j = 2`

Bubble Sort Naïve

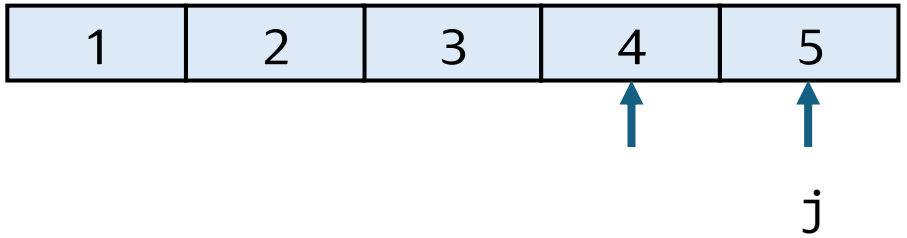
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```




```
i = 1; j = 3
```

Bubble Sort Naïve

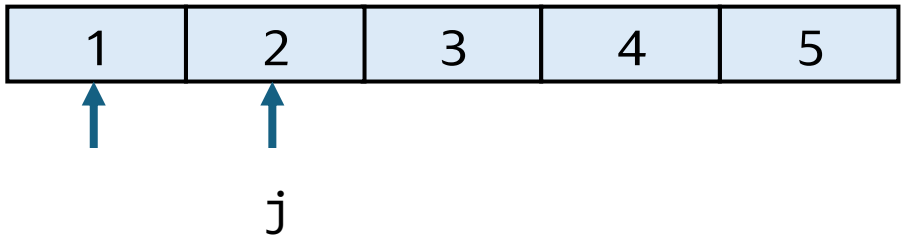
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



i = 1; j = 4

Bubble Sort Naïve

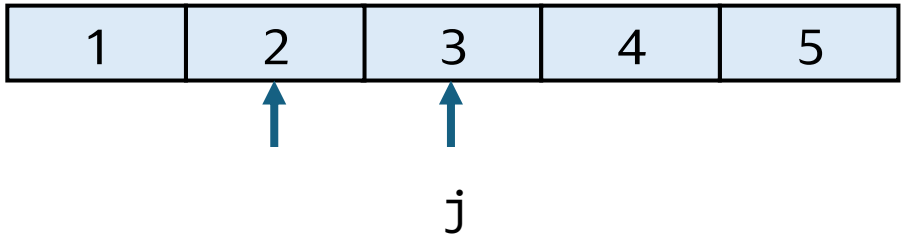
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



`i = 2; j = 1`

Bubble Sort Naïve

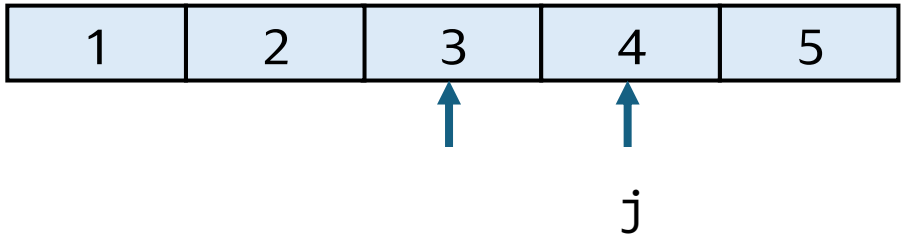
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



i = 2; j = 2

Bubble Sort Naïve

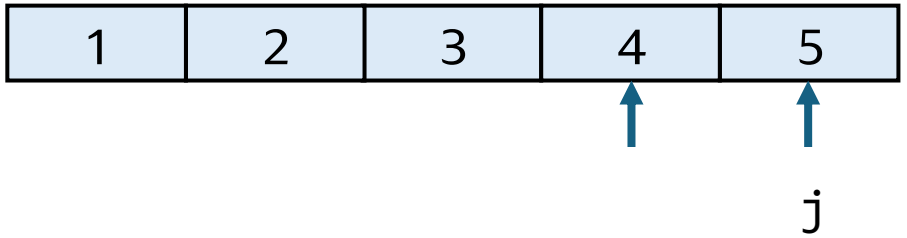
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



`i = 2; j = 3`

Bubble Sort Naïve

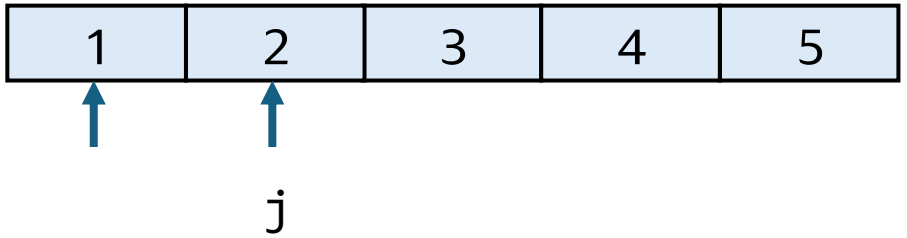
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



i = 2; j = 4

Bubble Sort Naïve

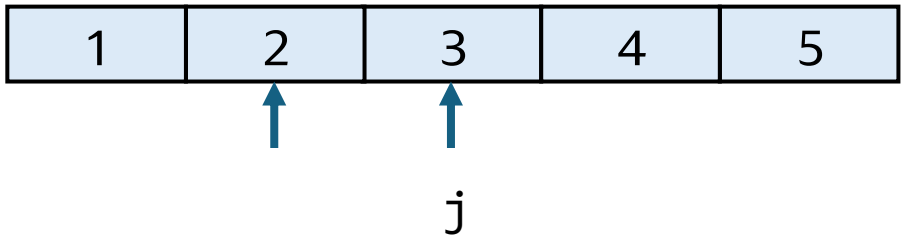
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



i = 3; j = 1

Bubble Sort Naïve

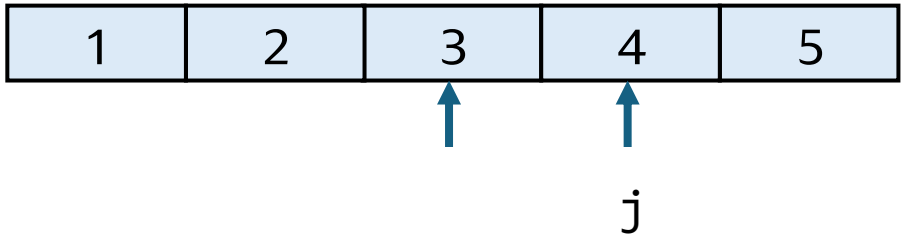
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



i = 3; j = 2

Bubble Sort Naïve

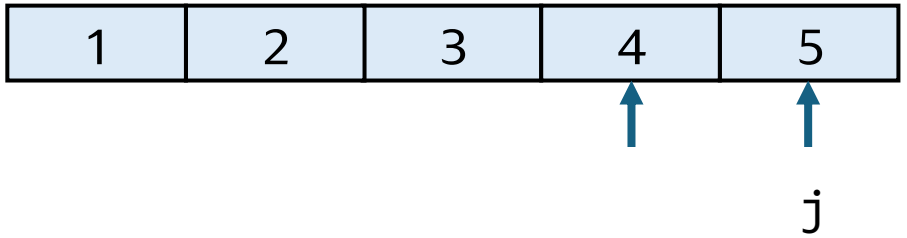
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```




```
i = 3; j = 3
```

Bubble Sort Naïve

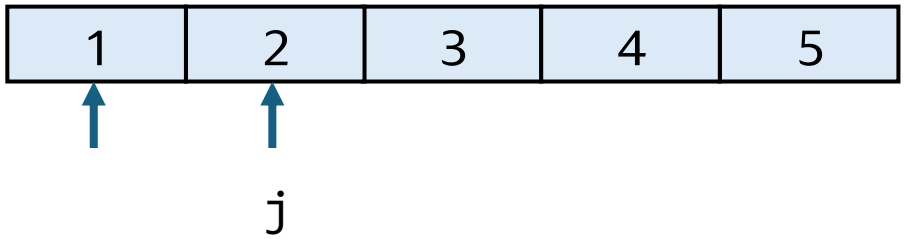
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



i = 3; j = 4

Bubble Sort Naïve

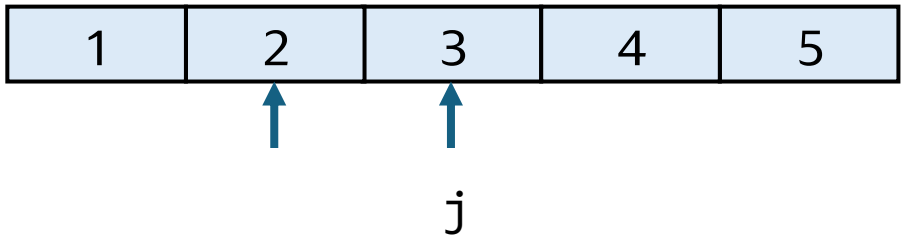
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



`i = 4; j = 1`

Bubble Sort Naïve

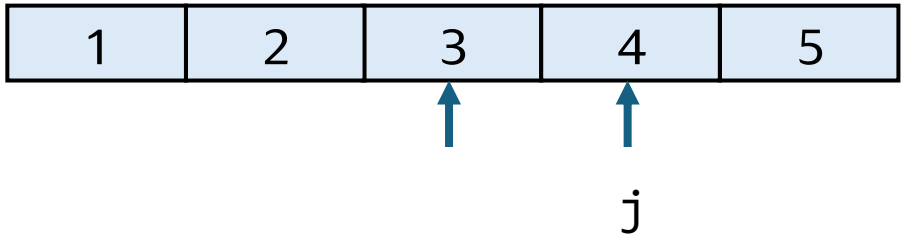
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



i = 4; j = 2

Bubble Sort Naïve

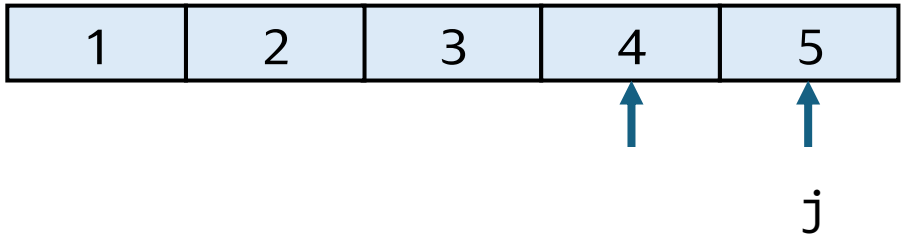
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



i = 4; j = 3

Bubble Sort Naïve

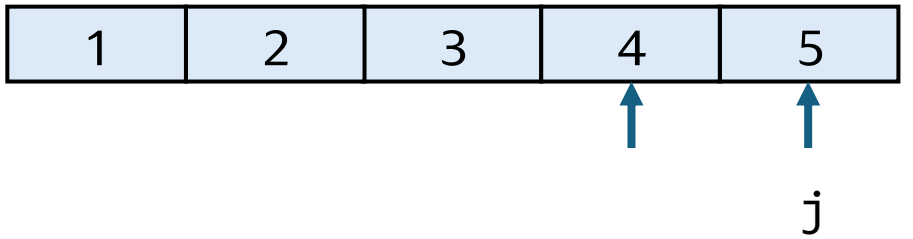
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



i = 4; j = 4

Bubble Sort Naïve

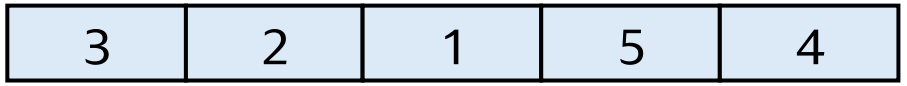
```
void bubble_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
      }
    }
  }
}
```



Worst-case: $(n-1)^2$ comparisons + $n(n-1)/2$ swaps
Best-case: $(n-1)^2$ comparisons

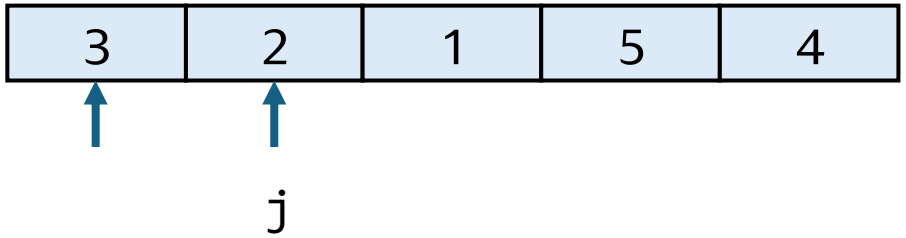
Bubble Sort with Early Termination

```
void bubble_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



Bubble Sort with Early Termination

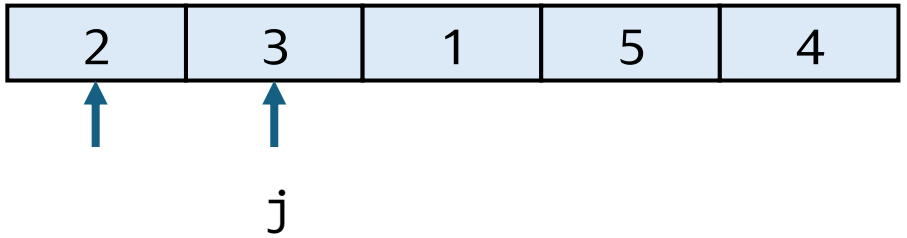
```
void bubble_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 0; j = 1; swapped = false

Bubble Sort with Early Termination

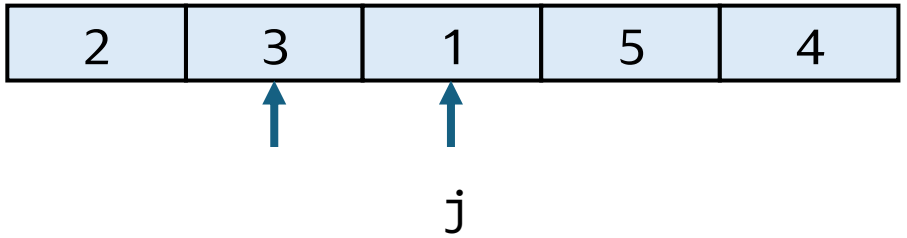
```
void bubble_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 0; j = 1; swapped = true

Bubble Sort with Early Termination

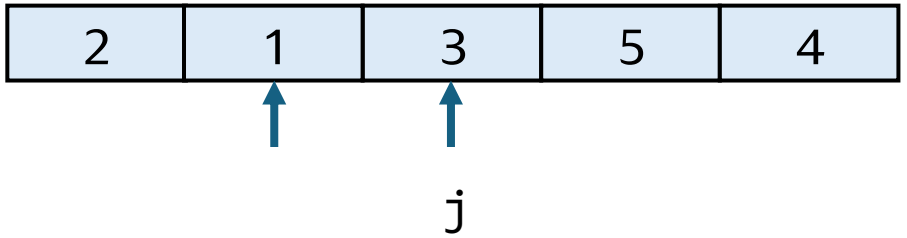
```
void bubble_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 0; j = 2; swapped = true

Bubble Sort with Early Termination

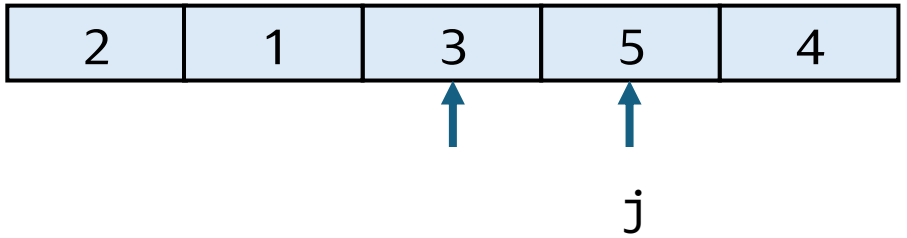
```
void bubble_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 0; j = 2; swapped = true

Bubble Sort with Early Termination

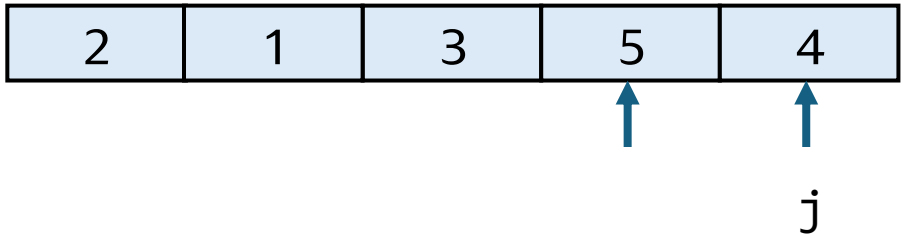
```
void bubble_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 0; j = 3; swapped = true

Bubble Sort with Early Termination

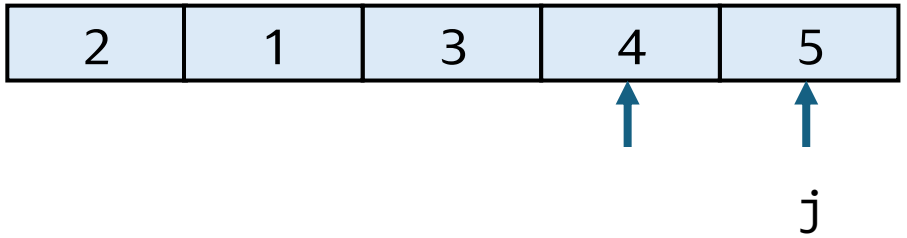
```
void bubble_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 0; j = 4; swapped = true

Bubble Sort with Early Termination

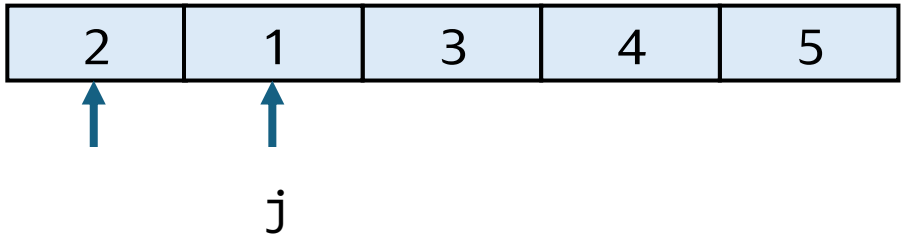
```
void bubble_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 0; j = 4; swapped = true

Bubble Sort with Early Termination

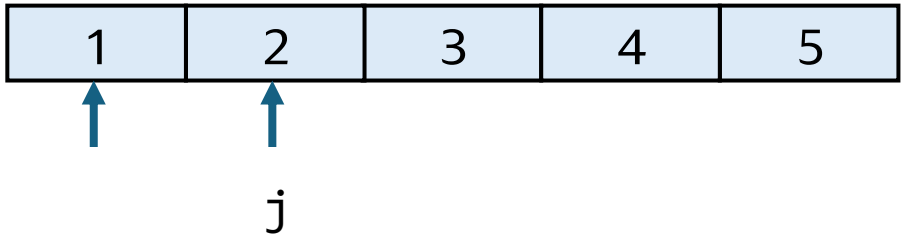
```
void bubble_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 1; j = 1; swapped = false

Bubble Sort with Early Termination

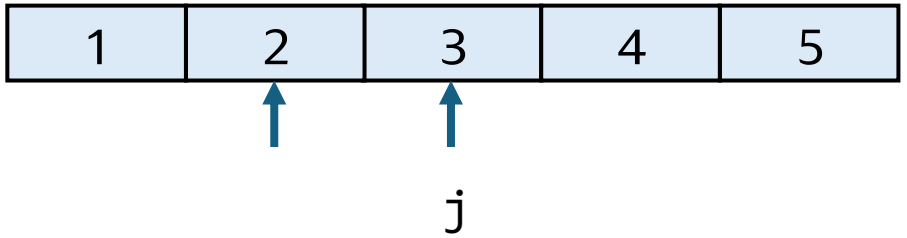
```
void bubble_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 1; j = 1; swapped = true

Bubble Sort with Early Termination

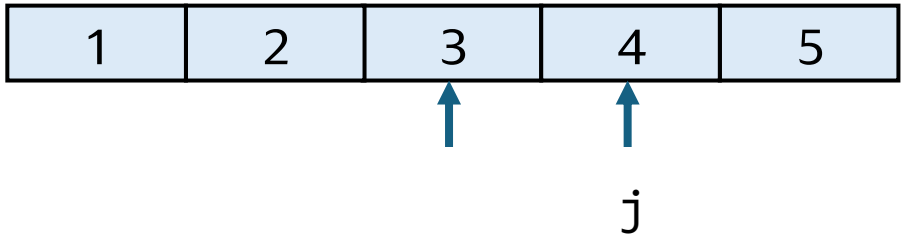
```
void bubble_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 1; j = 2; swapped = true

Bubble Sort with Early Termination

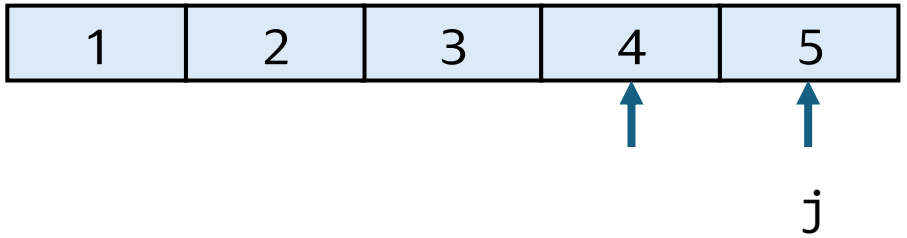
```
void bubble_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 1; j = 3; swapped = true

Bubble Sort with Early Termination

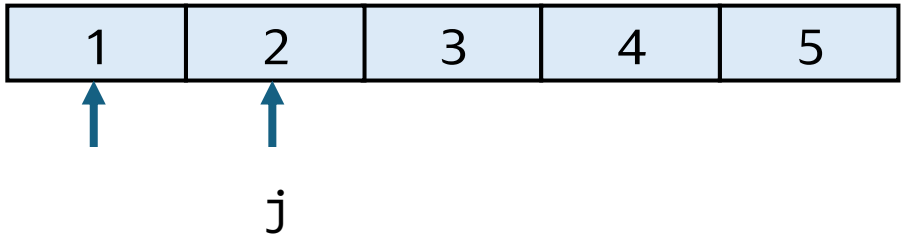
```
void bubble_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 1; j = 4; swapped = true

Bubble Sort with Early Termination

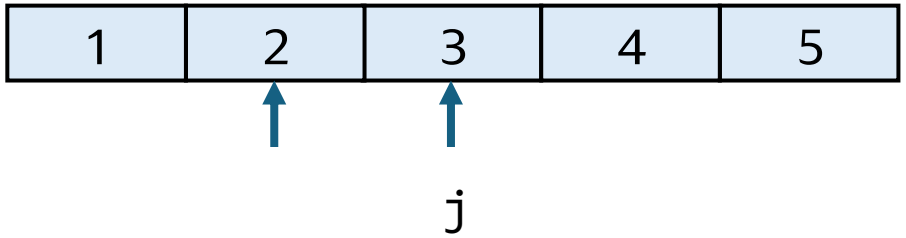
```
void bubble_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 2; j = 1; swapped = false

Bubble Sort with Early Termination

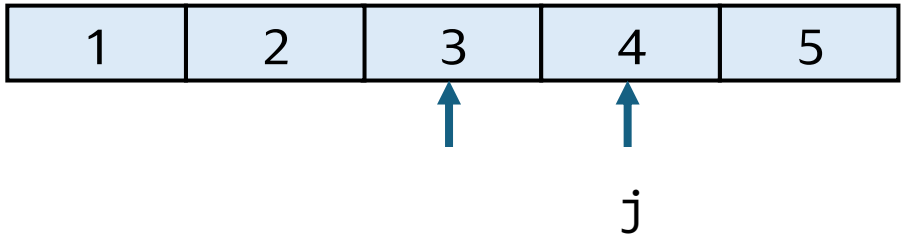
```
void bubble_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 2; j = 2; swapped = false

Bubble Sort with Early Termination

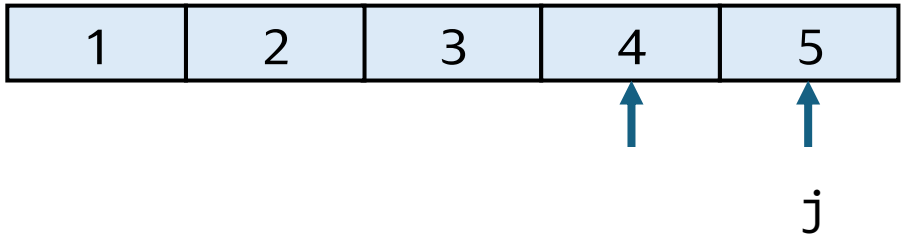
```
void bubble_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 2; j = 3; swapped = false

Bubble Sort with Early Termination

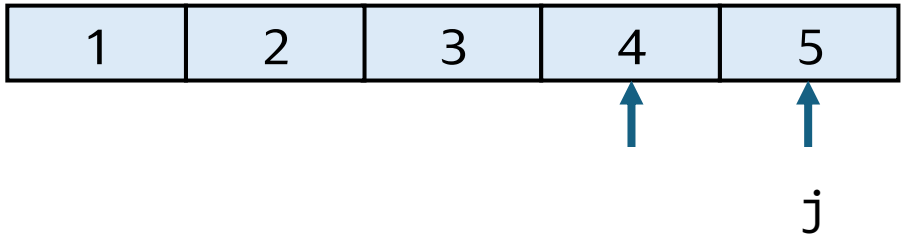
```
void bubble_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 2; j = 4; swapped = false

Bubble Sort with Early Termination

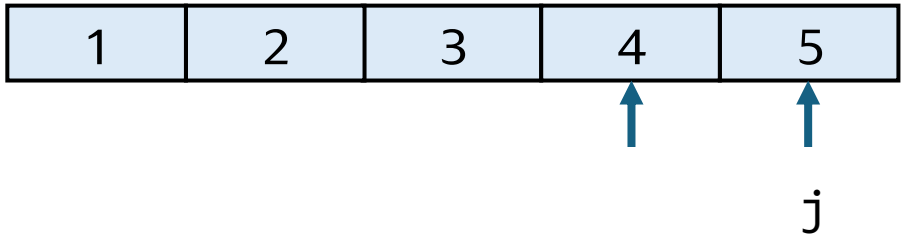
```
void bubble_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 2; j = 4; swapped = false

Bubble Sort with Early Termination

```
void bubble_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```

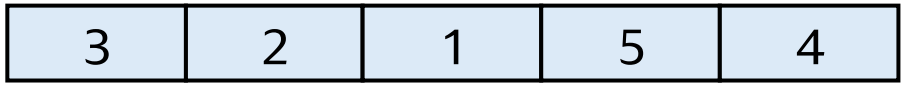


Worst-case: $(n-1)^2$ comparisons + $n(n-1)/2$ swaps

Best-case: $(n-1)$ comparisons

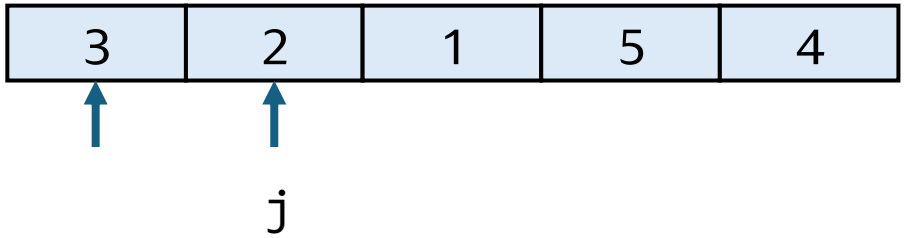
Bubble Sort with Early Termination + Sorted Region

```
void bubble_sort_v2(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz - i; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



Bubble Sort with Early Termination + Sorted Region

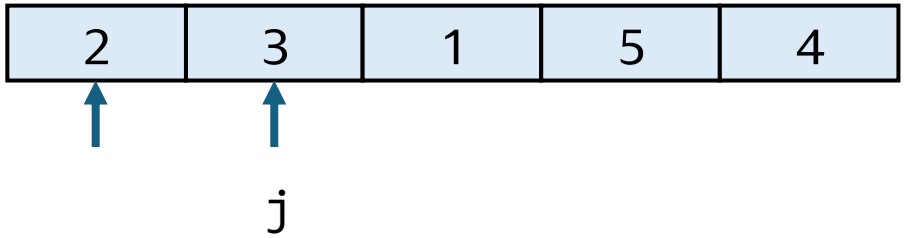
```
void bubble_sort_v2(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz - i; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 0; j = 1; swapped = false

Bubble Sort with Early Termination + Sorted Region

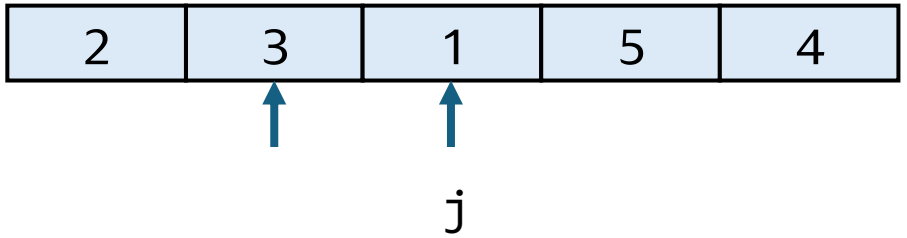
```
void bubble_sort_v2(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz - i; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 0; j = 1; swapped = true

Bubble Sort with Early Termination + Sorted Region

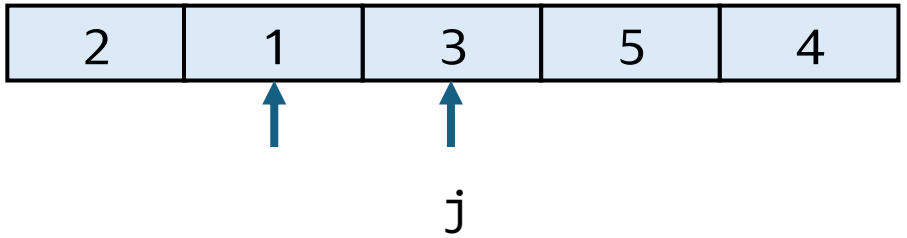
```
void bubble_sort_v2(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz - i; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 0; j = 2; swapped = true

Bubble Sort with Early Termination + Sorted Region

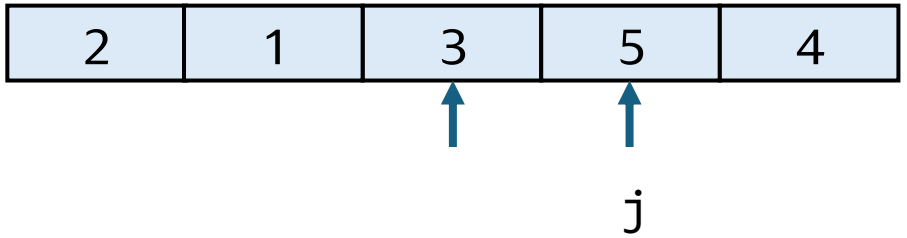
```
void bubble_sort_v2(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz - i; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 0; j = 2; swapped = true

Bubble Sort with Early Termination + Sorted Region

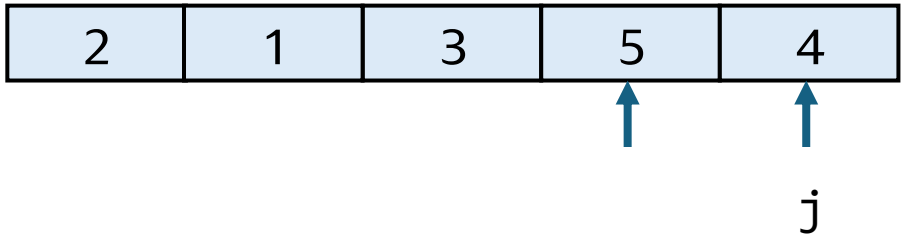
```
void bubble_sort_v2(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz - i; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 0; j = 3; swapped = true

Bubble Sort with Early Termination + Sorted Region

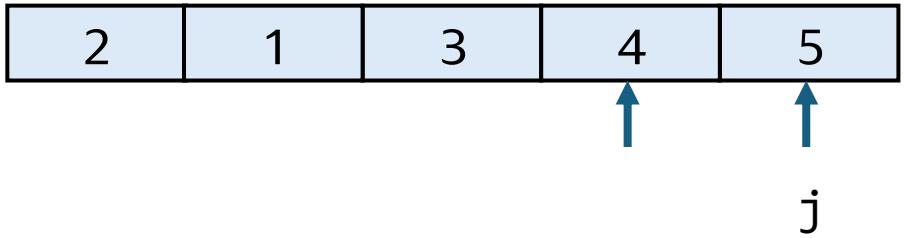
```
void bubble_sort_v2(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz - i; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 0; j = 4; swapped = true

Bubble Sort with Early Termination + Sorted Region

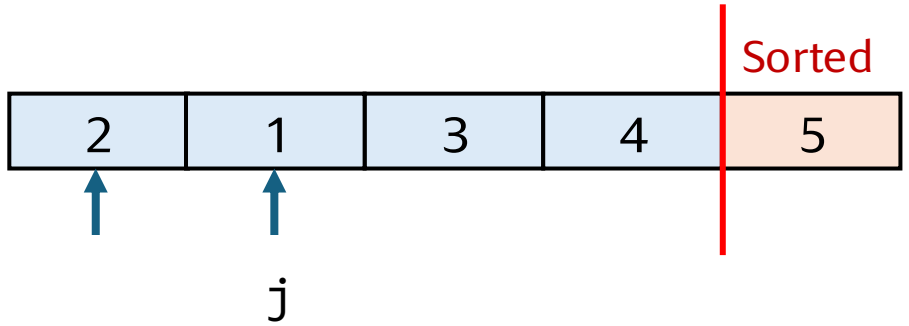
```
void bubble_sort_v2(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz - i; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 0; j = 4; swapped = true

Bubble Sort with Early Termination + Sorted Region

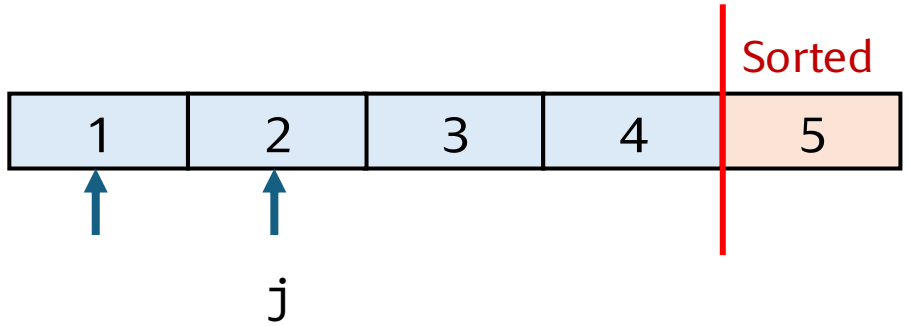
```
void bubble_sort_v2(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz - i; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 1; j = 1; swapped = false

Bubble Sort with Early Termination + Sorted Region

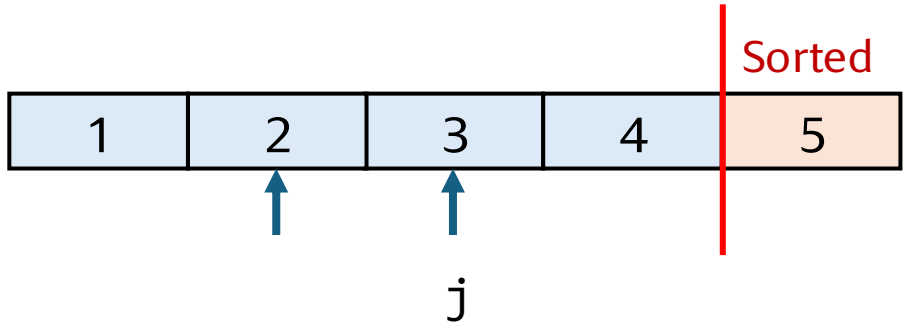
```
void bubble_sort_v2(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz - i; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 1; j = 1; swapped = true

Bubble Sort with Early Termination + Sorted Region

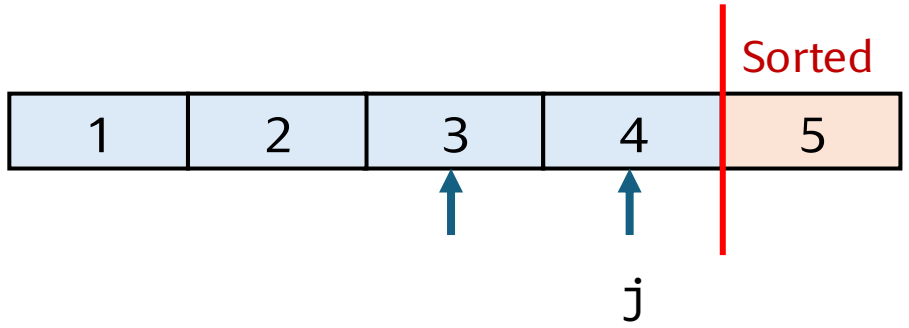
```
void bubble_sort_v2(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz - i; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 1; j = 2; swapped = true

Bubble Sort with Early Termination + Sorted Region

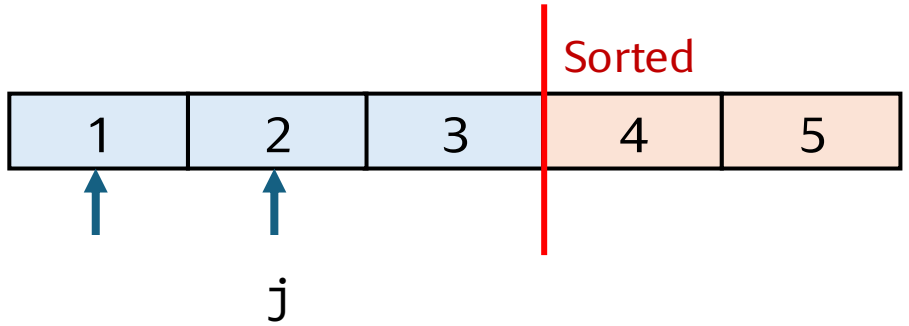
```
void bubble_sort_v2(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz - i; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 1; j = 3; swapped = true

Bubble Sort with Early Termination + Sorted Region

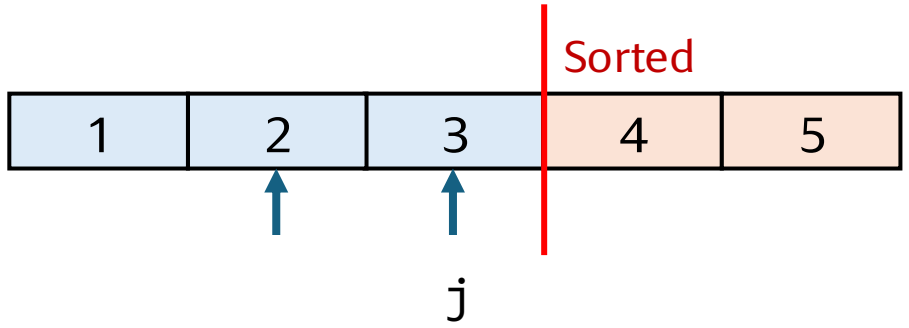
```
void bubble_sort_v2(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz - i; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```



i = 2; j = 1; swapped = false

Bubble Sort with Early Termination + Sorted Region

```
void bubble_sort_v2(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz - i; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```

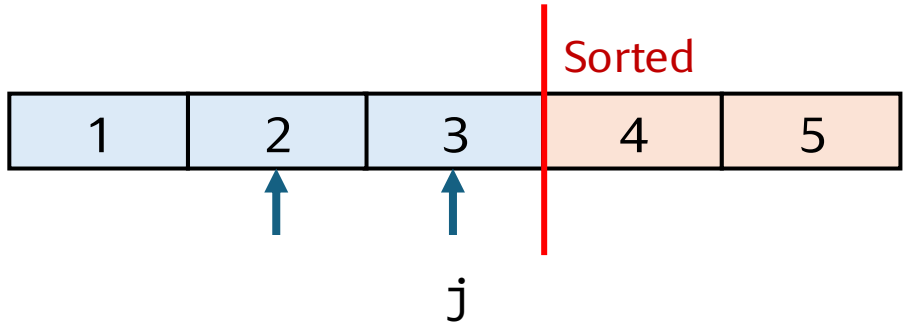


i = 2; j = 2; swapped = false

Bubble Sort with Early Termination + Sorted Region

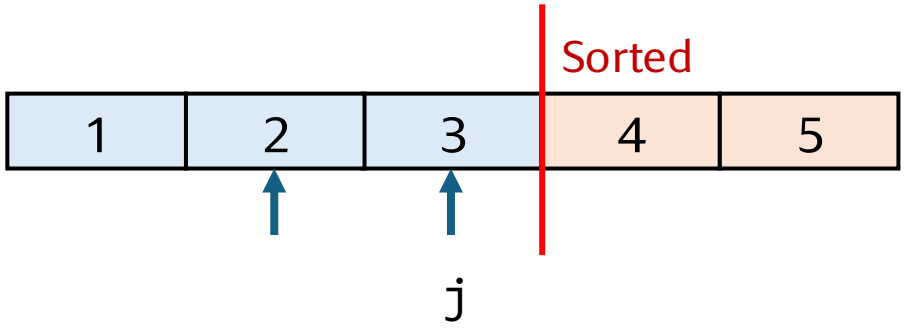
```
void bubble_sort_v2(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz - i; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```

i = 2; j = 2; swapped = false



Bubble Sort with Early Termination + Sorted Region

```
void bubble_sort_v2(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t i = 0; i < sz - 1; i++) {
    bool swapped = false;
    for (size_t j = 1; j < sz - i; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        swapped = true;
      }
    }

    if (!swapped) break;
  }
}
```

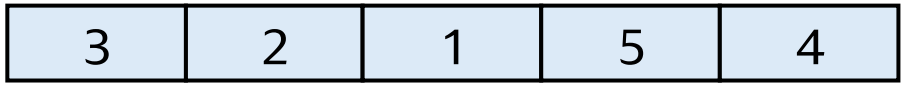


Worst-case: $n(n-1)/2$ (comparisons + swaps)

Best-case: $(n-1)$ comparisons

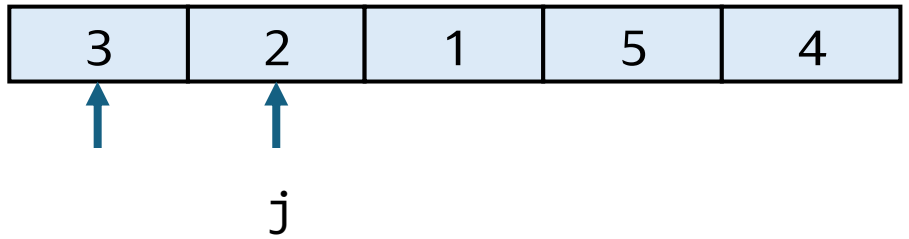
Bubble Sort with Early Termination + Better Sorted Region

```
void bubble_sort_v3(int *arr, size_t sz) {
  if (sz <= 1) return;

  size_t end = sz;
  for (size_t i = 0; i < sz - 1; i++) {
    size_t last_swapped = 0;
    for (size_t j = 1; j < end; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        last_swapped = j;
      }
    }
    if (!last_swapped) break;
    end = last_swapped;
  }
}
```



Bubble Sort with Early Termination + Better Sorted Region

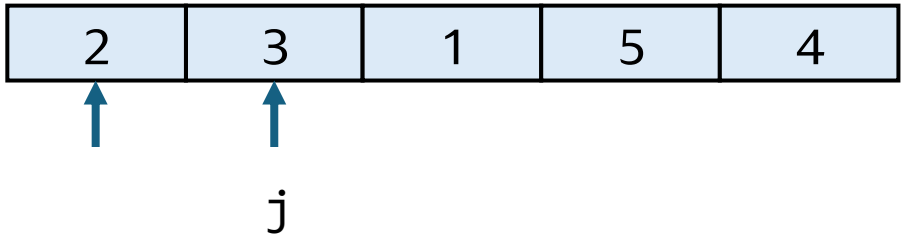
```
void bubble_sort_v3(int *arr, size_t sz) {
  if (sz <= 1) return;

  size_t end = sz;
  for (size_t i = 0; i < sz - 1; i++) {
    size_t last_swapped = 0;
    for (size_t j = 1; j < end; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        last_swapped = j;
      }
    }
    if (!last_swapped) break;
    end = last_swapped;
  }
}
```



`end = 5; i = 0; j = 1; last_swapped = 0`

Bubble Sort with Early Termination + Better Sorted Region

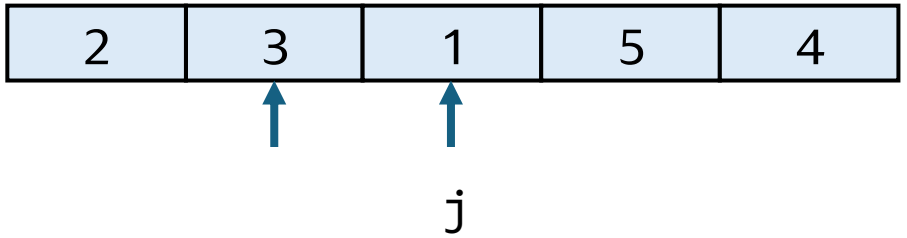
```
void bubble_sort_v3(int *arr, size_t sz) {
  if (sz <= 1) return;

  size_t end = sz;
  for (size_t i = 0; i < sz - 1; i++) {
    size_t last_swapped = 0;
    for (size_t j = 1; j < end; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        last_swapped = j;
      }
    }
    if (!last_swapped) break;
    end = last_swapped;
  }
}
```



end = 5; i = 0; j = 1; last_swapped = 1

Bubble Sort with Early Termination + Better Sorted Region

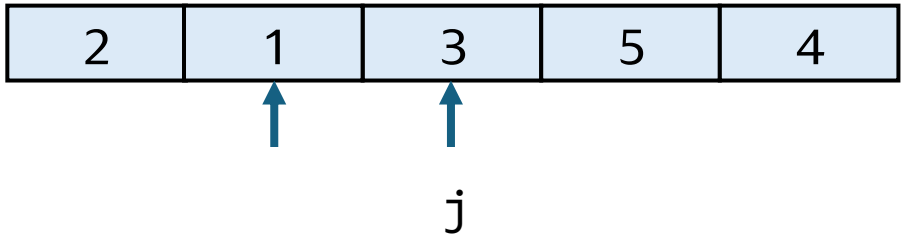
```
void bubble_sort_v3(int *arr, size_t sz) {
  if (sz <= 1) return;

  size_t end = sz;
  for (size_t i = 0; i < sz - 1; i++) {
    size_t last_swapped = 0;
    for (size_t j = 1; j < end; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        last_swapped = j;
      }
    }
    if (!last_swapped) break;
    end = last_swapped;
  }
}
```



end = 5; i = 0; j = 2; last_swapped = 1

Bubble Sort with Early Termination + Better Sorted Region

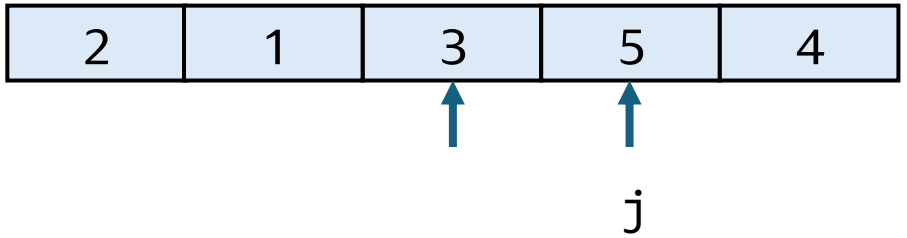
```
void bubble_sort_v3(int *arr, size_t sz) {
  if (sz <= 1) return;

  size_t end = sz;
  for (size_t i = 0; i < sz - 1; i++) {
    size_t last_swapped = 0;
    for (size_t j = 1; j < end; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        last_swapped = j;
      }
    }
    if (!last_swapped) break;
    end = last_swapped;
  }
}
```



`end = 5; i = 0; j = 2; last_swapped = 2`

Bubble Sort with Early Termination + Better Sorted Region

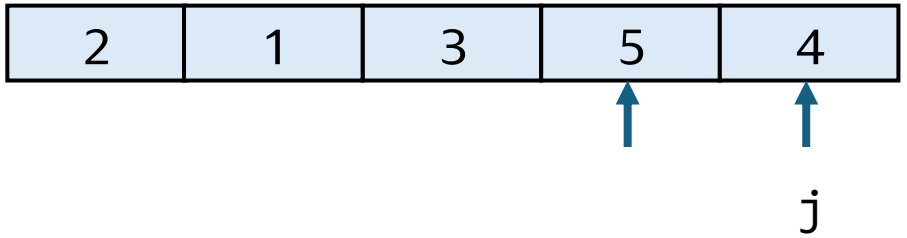
```
void bubble_sort_v3(int *arr, size_t sz) {
  if (sz <= 1) return;

  size_t end = sz;
  for (size_t i = 0; i < sz - 1; i++) {
    size_t last_swapped = 0;
    for (size_t j = 1; j < end; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        last_swapped = j;
      }
    }
    if (!last_swapped) break;
    end = last_swapped;
  }
}
```



`end = 5; i = 0; j = 3; last_swapped = 2`

Bubble Sort with Early Termination + Better Sorted Region

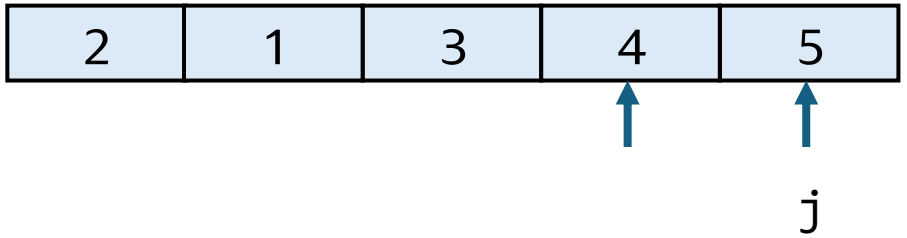
```
void bubble_sort_v3(int *arr, size_t sz) {
  if (sz <= 1) return;

  size_t end = sz;
  for (size_t i = 0; i < sz - 1; i++) {
    size_t last_swapped = 0;
    for (size_t j = 1; j < end; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        last_swapped = j;
      }
    }
    if (!last_swapped) break;
    end = last_swapped;
  }
}
```



end = 5; i = 0; j = 4; last_swapped = 2

Bubble Sort with Early Termination + Better Sorted Region

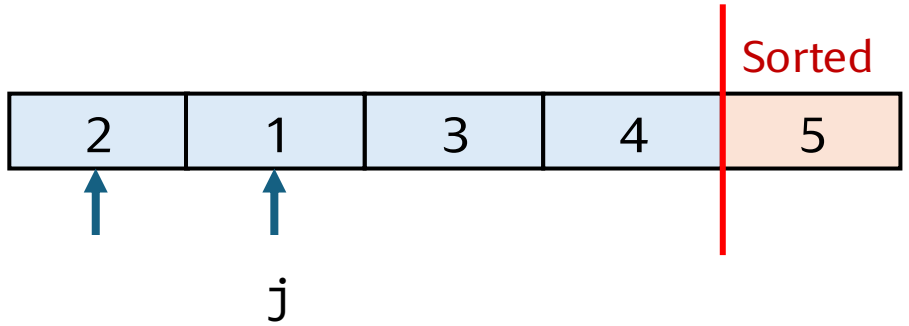
```
void bubble_sort_v3(int *arr, size_t sz) {
  if (sz <= 1) return;

  size_t end = sz;
  for (size_t i = 0; i < sz - 1; i++) {
    size_t last_swapped = 0;
    for (size_t j = 1; j < end; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        last_swapped = j;
      }
    }
    if (!last_swapped) break;
    end = last_swapped;
  }
}
```



end = 5; i = 0; j = 4; last_swapped = 4

Bubble Sort with Early Termination + Better Sorted Region

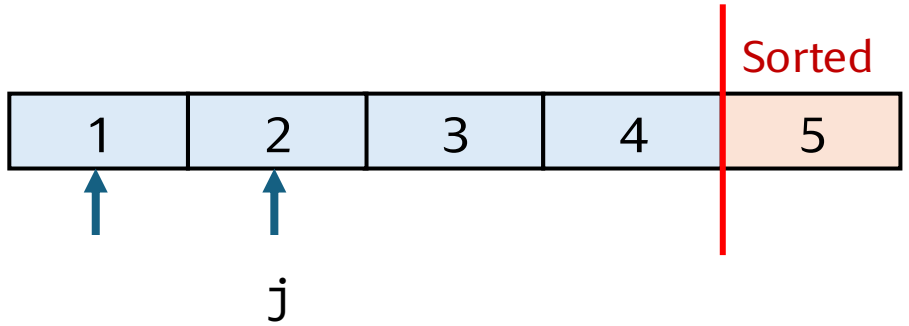
```
void bubble_sort_v3(int *arr, size_t sz) {
  if (sz <= 1) return;

  size_t end = sz;
  for (size_t i = 0; i < sz - 1; i++) {
    size_t last_swapped = 0;
    for (size_t j = 1; j < end; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        last_swapped = j;
      }
    }
    if (!last_swapped) break;
    end = last_swapped;
  }
}
```



`end = 4; i = 0; j = 1; last_swapped = 0`

Bubble Sort with Early Termination + Better Sorted Region

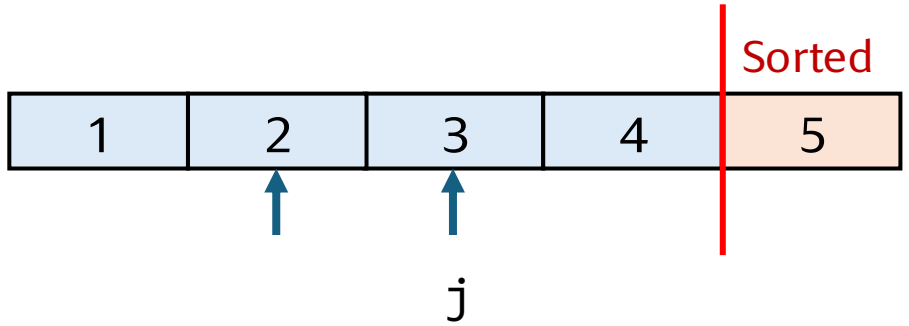
```
void bubble_sort_v3(int *arr, size_t sz) {
  if (sz <= 1) return;

  size_t end = sz;
  for (size_t i = 0; i < sz - 1; i++) {
    size_t last_swapped = 0;
    for (size_t j = 1; j < end; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        last_swapped = j;
      }
    }
    if (!last_swapped) break;
    end = last_swapped;
  }
}
```



end = 4; i = 1; j = 1; last_swapped = 1

Bubble Sort with Early Termination + Better Sorted Region

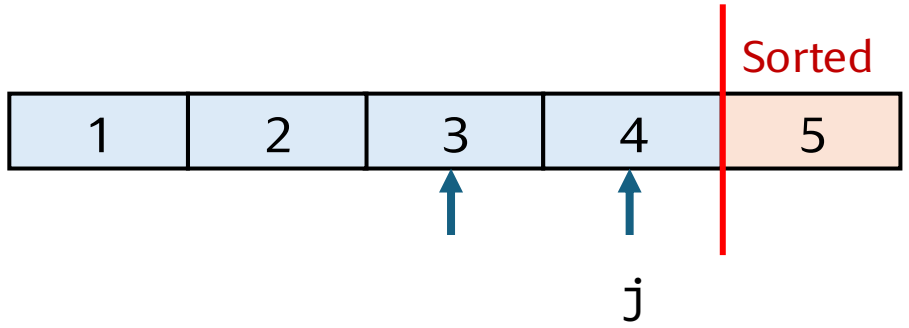
```
void bubble_sort_v3(int *arr, size_t sz) {
  if (sz <= 1) return;

  size_t end = sz;
  for (size_t i = 0; i < sz - 1; i++) {
    size_t last_swapped = 0;
    for (size_t j = 1; j < end; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        last_swapped = j;
      }
    }
    if (!last_swapped) break;
    end = last_swapped;
  }
}
```



end = 4; i = 1; j = 2; last_swapped = 1

Bubble Sort with Early Termination + Better Sorted Region

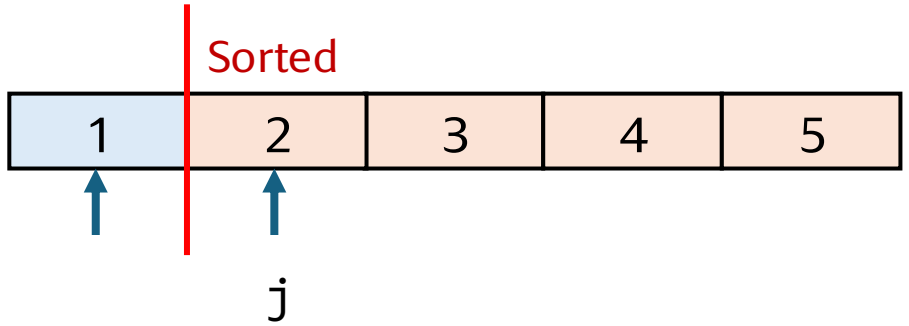
```
void bubble_sort_v3(int *arr, size_t sz) {
  if (sz <= 1) return;

  size_t end = sz;
  for (size_t i = 0; i < sz - 1; i++) {
    size_t last_swapped = 0;
    for (size_t j = 1; j < end; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        last_swapped = j;
      }
    }
    if (!last_swapped) break;
    end = last_swapped;
  }
}
```



end = 4; i = 1; j = 3; last_swapped = 1

Bubble Sort with Early Termination + Better Sorted Region

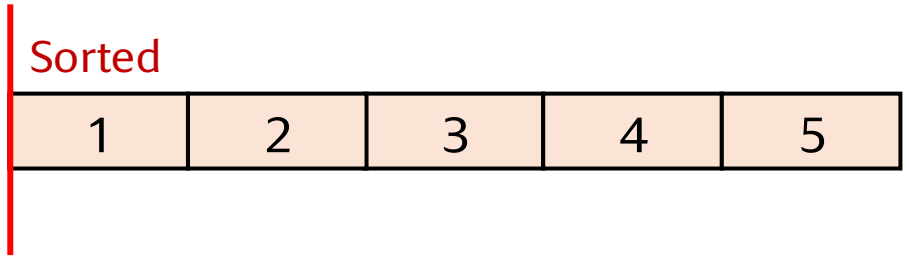
```
void bubble_sort_v3(int *arr, size_t sz) {
  if (sz <= 1) return;

  size_t end = sz;
  for (size_t i = 0; i < sz - 1; i++) {
    size_t last_swapped = 0;
    for (size_t j = 1; j < end; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        last_swapped = j;
      }
    }
    if (!last_swapped) break;
    end = last_swapped;
  }
}
```



`end = 1; i = 2; j = 1; last_swapped = 0`

Bubble Sort with Early Termination + Better Sorted Region

```
void bubble_sort_v3(int *arr, size_t sz) {
  if (sz <= 1) return;

  size_t end = sz;
  for (size_t i = 0; i < sz - 1; i++) {
    size_t last_swapped = 0;
    for (size_t j = 1; j < end; j++) {
      if (arr[j] < arr[j - 1]) {
        swap_int(&arr[j - 1], &arr[j]);
        last_swapped = j;
      }
    }
    if (!last_swapped) break;
    end = last_swapped;
  }
}
```

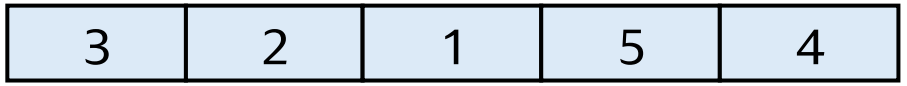


Worst-case: $n(n-1)/2$ (comparisons + swaps)

Best-case: $(n-1)$ comparisons

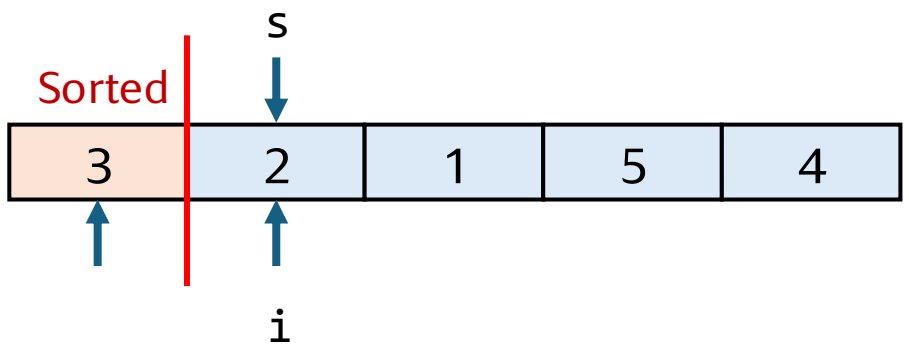
Naïve Insertion Sort

```
void insertion_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    for (size_t i = s; i > 0; i--) {
      if (arr[i] < arr[i - 1])
        swap_int(&arr[i], &arr[i - 1]);
      else
        break;
    }
  }
}
```



Naïve Insertion Sort

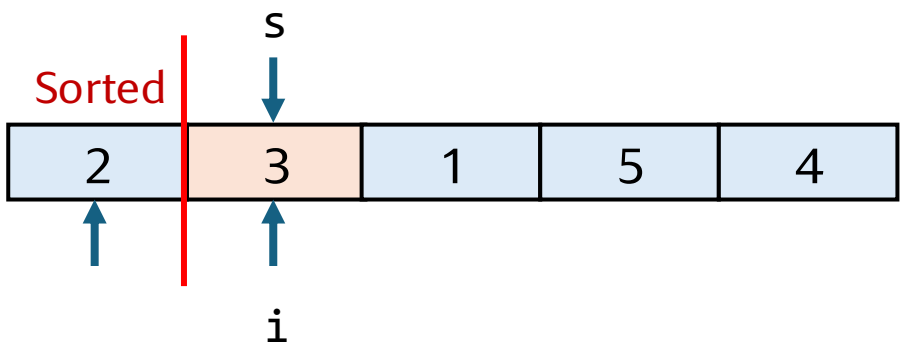
```
void insertion_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    for (size_t i = s; i > 0; i--) {
      if (arr[i] < arr[i - 1])
        swap_int(&arr[i], &arr[i - 1]);
      else
        break;
    }
  }
}
```



s = 1; i = 1

Naïve Insertion Sort

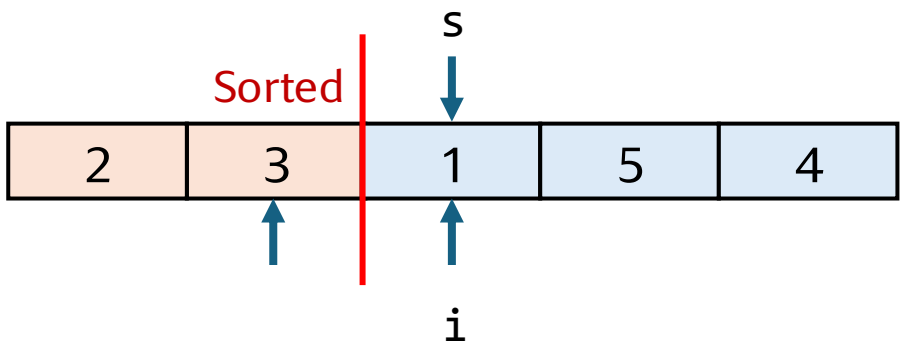
```
void insertion_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    for (size_t i = s; i > 0; i--) {
      if (arr[i] < arr[i - 1])
        swap_int(&arr[i], &arr[i - 1]);
      else
        break;
    }
  }
}
```



`s = 1; i = 1`

Naïve Insertion Sort

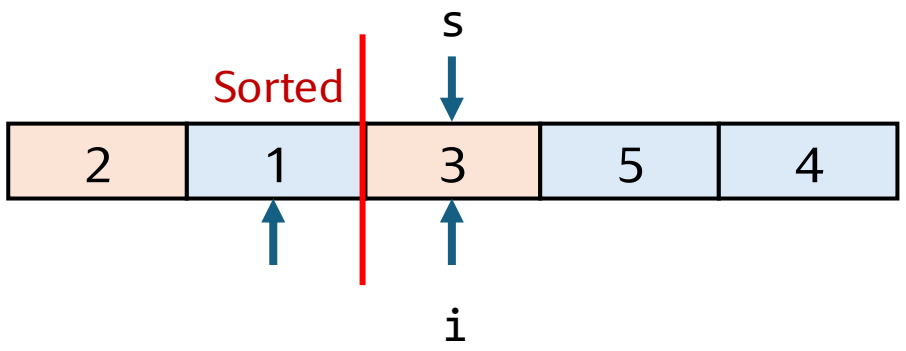
```
void insertion_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    for (size_t i = s; i > 0; i--) {
      if (arr[i] < arr[i - 1])
        swap_int(&arr[i], &arr[i - 1]);
      else
        break;
    }
  }
}
```



s = 2; i = 2

Naïve Insertion Sort

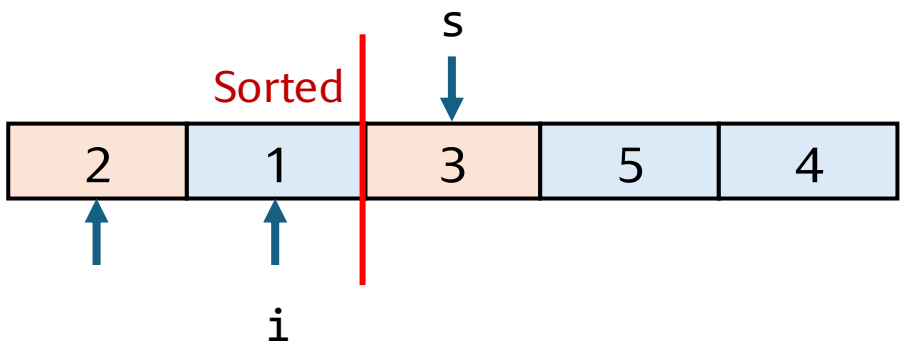
```
void insertion_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    for (size_t i = s; i > 0; i--) {
      if (arr[i] < arr[i - 1])
        swap_int(&arr[i], &arr[i - 1]);
      else
        break;
    }
  }
}
```



s = 2; i = 2

Naïve Insertion Sort

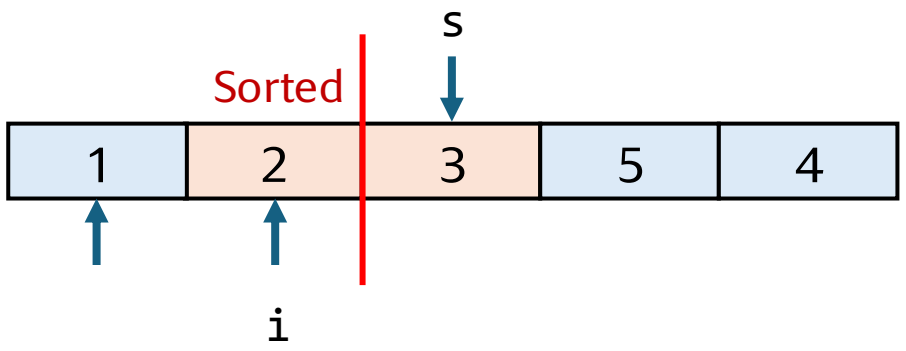
```
void insertion_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    for (size_t i = s; i > 0; i--) {
      if (arr[i] < arr[i - 1])
        swap_int(&arr[i], &arr[i - 1]);
      else
        break;
    }
  }
}
```



s = 2; i = 1

Naïve Insertion Sort

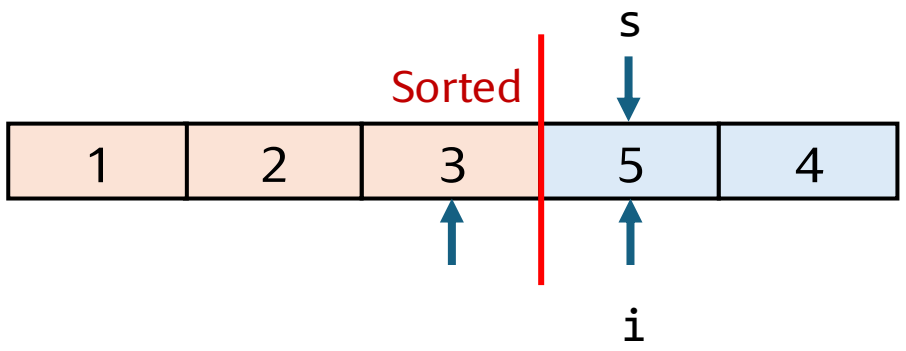
```
void insertion_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    for (size_t i = s; i > 0; i--) {
      if (arr[i] < arr[i - 1])
        swap_int(&arr[i], &arr[i - 1]);
      else
        break;
    }
  }
}
```



s = 2; i = 1

Naïve Insertion Sort

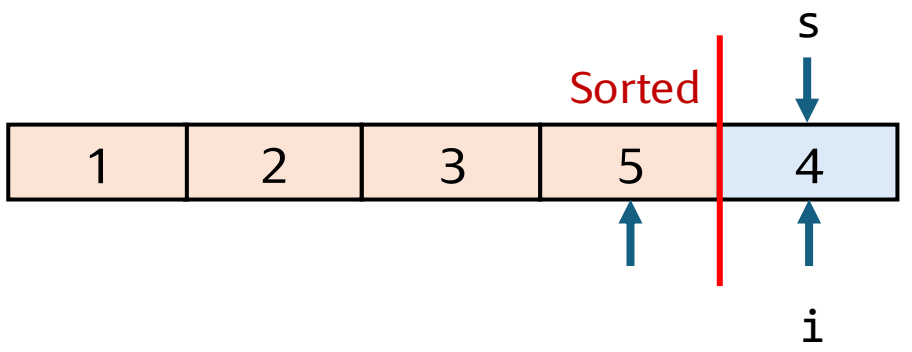
```
void insertion_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    for (size_t i = s; i > 0; i--) {
      if (arr[i] < arr[i - 1])
        swap_int(&arr[i], &arr[i - 1]);
      else
        break;
    }
  }
}
```



s = 3; i = 3

Naïve Insertion Sort

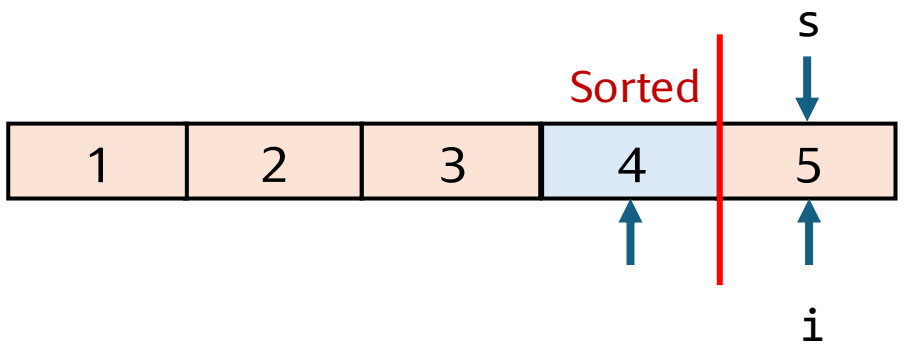
```
void insertion_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    for (size_t i = s; i > 0; i--) {
      if (arr[i] < arr[i - 1])
        swap_int(&arr[i], &arr[i - 1]);
      else
        break;
    }
  }
}
```



s = 4; i = 4

Naïve Insertion Sort

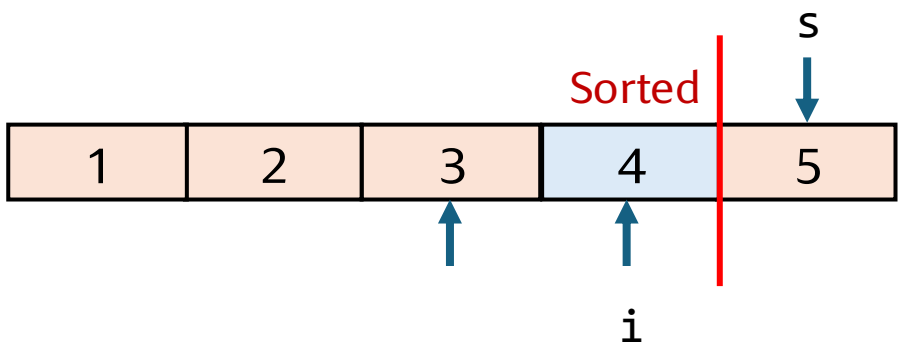
```
void insertion_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    for (size_t i = s; i > 0; i--) {
      if (arr[i] < arr[i - 1])
        swap_int(&arr[i], &arr[i - 1]);
      else
        break;
    }
  }
}
```



s = 4; i = 4

Naïve Insertion Sort

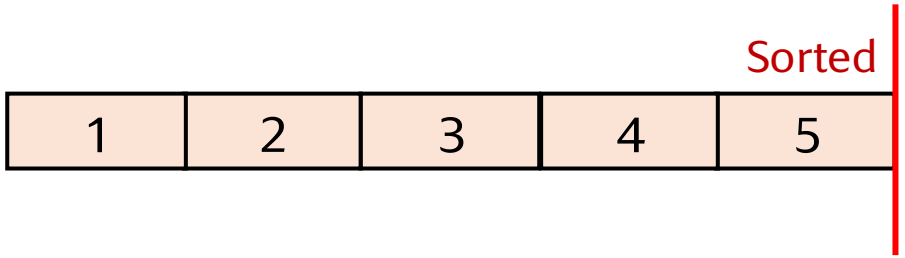
```
void insertion_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    for (size_t i = s; i > 0; i--) {
      if (arr[i] < arr[i - 1])
        swap_int(&arr[i], &arr[i - 1]);
      else
        break;
    }
  }
}
```



s = 4; i = 3

Naïve Insertion Sort

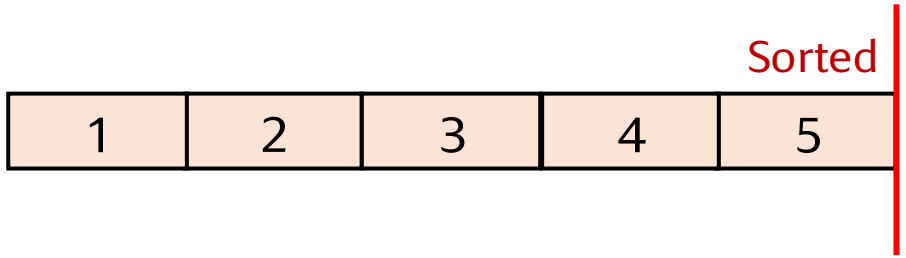
```
void insertion_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    for (size_t i = s; i > 0; i--) {
      if (arr[i] < arr[i - 1])
        swap_int(&arr[i], &arr[i - 1]);
      else
        break;
    }
  }
}
```



s = 4; i = 3

Naïve Insertion Sort

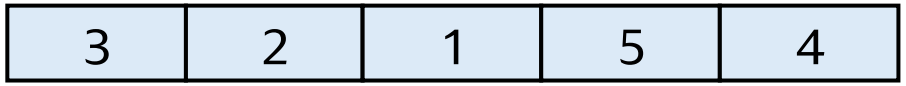
```
void insertion_sort_v0(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    for (size_t i = s; i > 0; i--) {
      if (arr[i] < arr[i - 1])
        swap_int(&arr[i], &arr[i - 1]);
      else
        break;
    }
  }
}
```



Worst-case: $n(n-1)/2$ (comparisons + swaps)
Best-case: $(n-1)$ comparisons

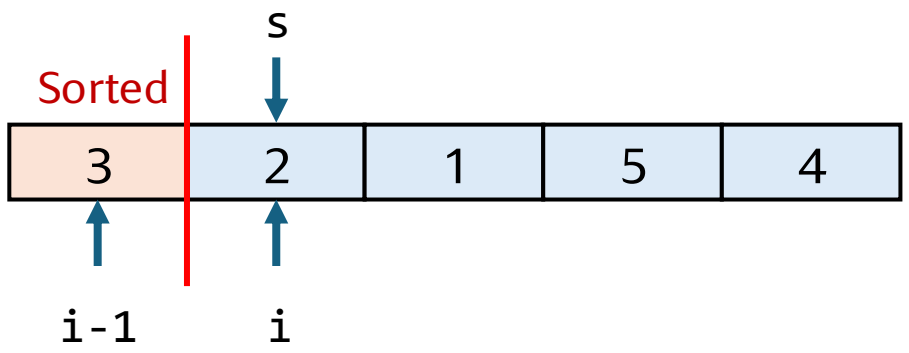
Insertion Sort with Moving

```
void insertion_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    size_t i = s;
    int elem = arr[s];
    while (i > 0 && elem < arr[i - 1]) {
      arr[i] = arr[i - 1];
      i -= 1;
    }
    arr[i] = elem;
  }
}
```



Insertion Sort with Moving

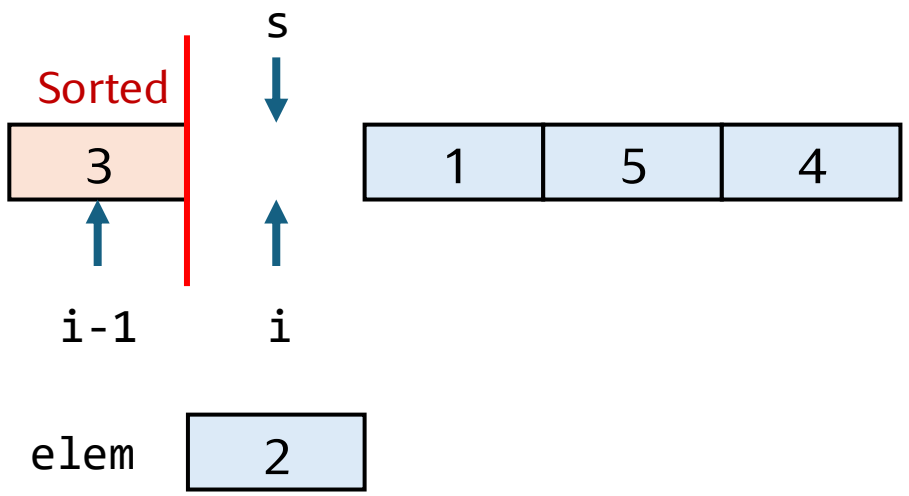
```
void insertion_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    size_t i = s;
    int elem = arr[s];
    while (i > 0 && elem < arr[i - 1]) {
      arr[i] = arr[i - 1];
      i -= 1;
    }
    arr[i] = elem;
  }
}
```



s = 1; i = 1

Insertion Sort with Moving

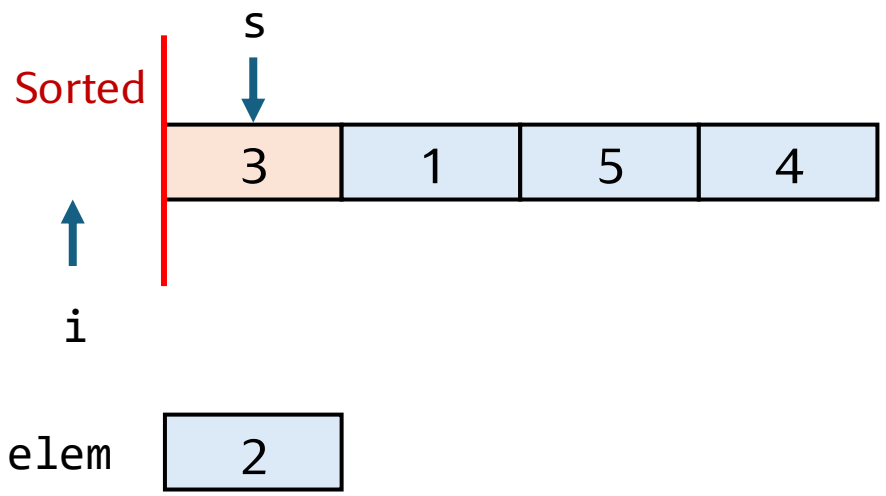
```
void insertion_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    size_t i = s;
    int elem = arr[s];
    while (i > 0 && elem < arr[i - 1]) {
      arr[i] = arr[i - 1];
      i -= 1;
    }
    arr[i] = elem;
  }
}
```



s = 1; i = 1; elem = 2

Insertion Sort with Moving

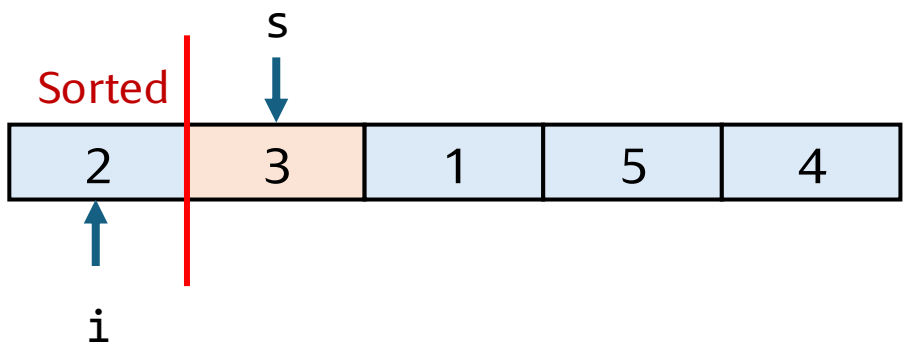
```
void insertion_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    size_t i = s;
    int elem = arr[s];
    while (i > 0 && elem < arr[i - 1]) {
      arr[i] = arr[i - 1];
      i -= 1;
    }
    arr[i] = elem;
  }
}
```



`s = 1; i = 0; elem = 2`

Insertion Sort with Moving

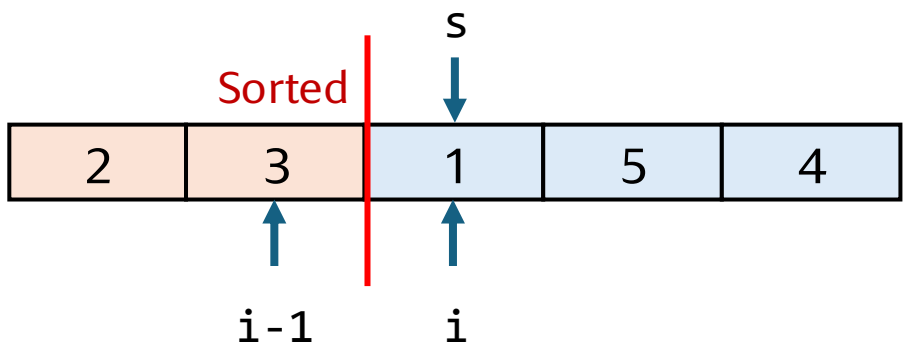
```
void insertion_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    size_t i = s;
    int elem = arr[s];
    while (i > 0 && elem < arr[i - 1]) {
      arr[i] = arr[i - 1];
      i -= 1;
    }
    arr[i] = elem;
  }
}
```



`s = 1; i = 0; elem = 2`

Insertion Sort with Moving

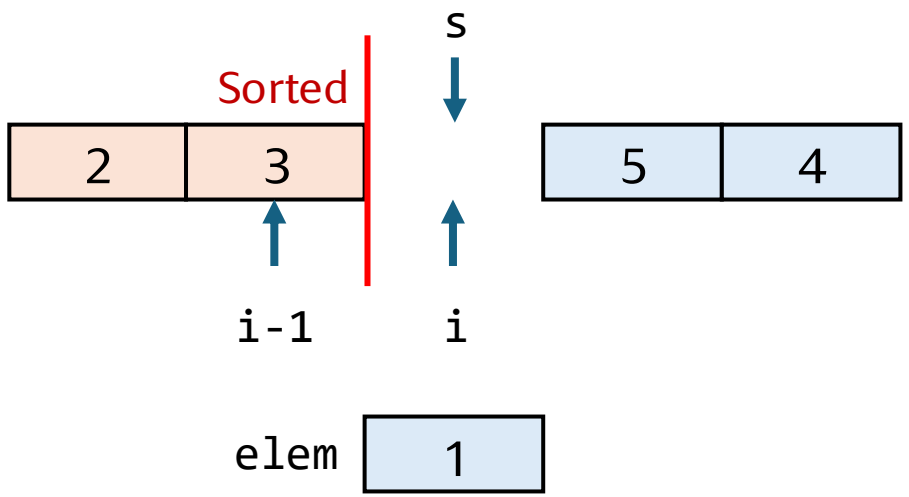
```
void insertion_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    size_t i = s;
    int elem = arr[s];
    while (i > 0 && elem < arr[i - 1]) {
      arr[i] = arr[i - 1];
      i -= 1;
    }
    arr[i] = elem;
  }
}
```



s = 2; i = 2

Insertion Sort with Moving

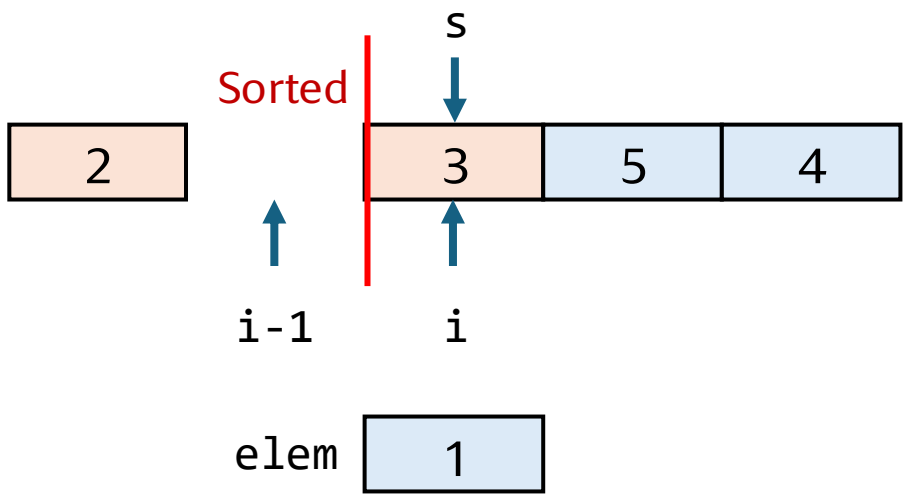
```
void insertion_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    size_t i = s;
    int elem = arr[s];
    while (i > 0 && elem < arr[i - 1]) {
      arr[i] = arr[i - 1];
      i -= 1;
    }
    arr[i] = elem;
  }
}
```



`s = 2; i = 2; elem = 1`

Insertion Sort with Moving

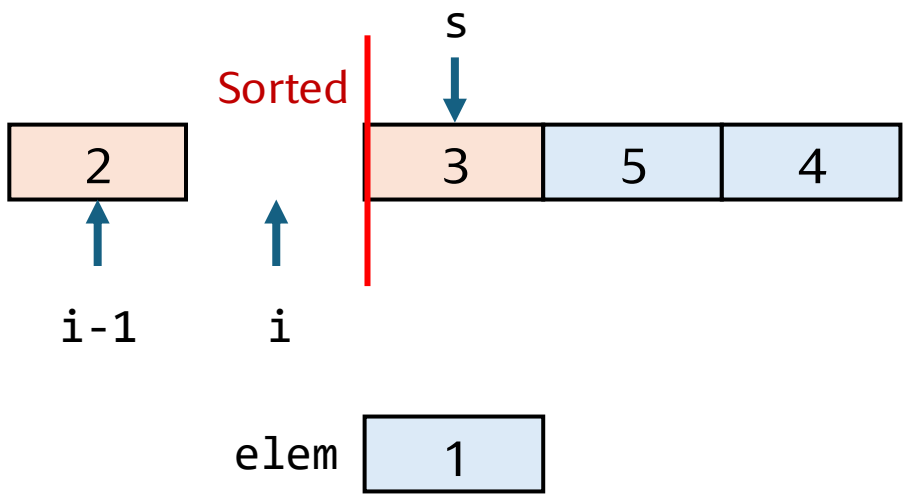
```
void insertion_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    size_t i = s;
    int elem = arr[s];
    while (i > 0 && elem < arr[i - 1]) {
      arr[i] = arr[i - 1];
      i -= 1;
    }
    arr[i] = elem;
  }
}
```



s = 2; i = 2; elem = 1

Insertion Sort with Moving

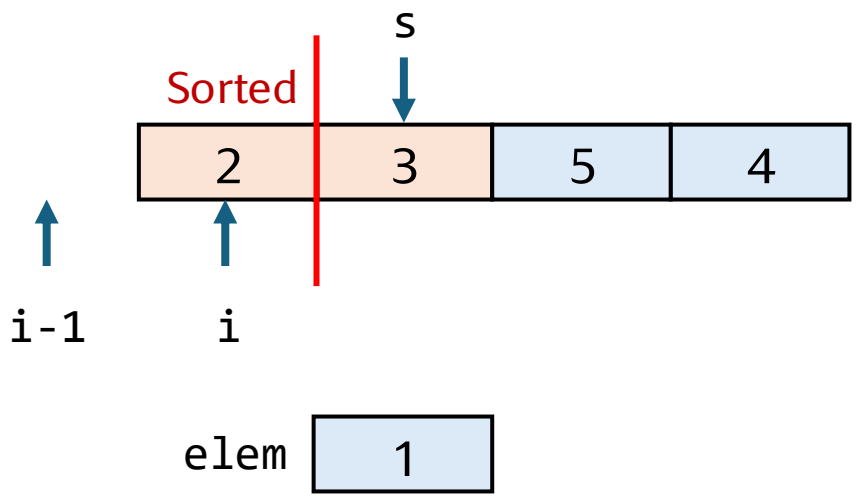
```
void insertion_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    size_t i = s;
    int elem = arr[s];
    while (i > 0 && elem < arr[i - 1]) {
      arr[i] = arr[i - 1];
      i -= 1;
    }
    arr[i] = elem;
  }
}
```



s = 2; i = 1; elem = 1

Insertion Sort with Moving

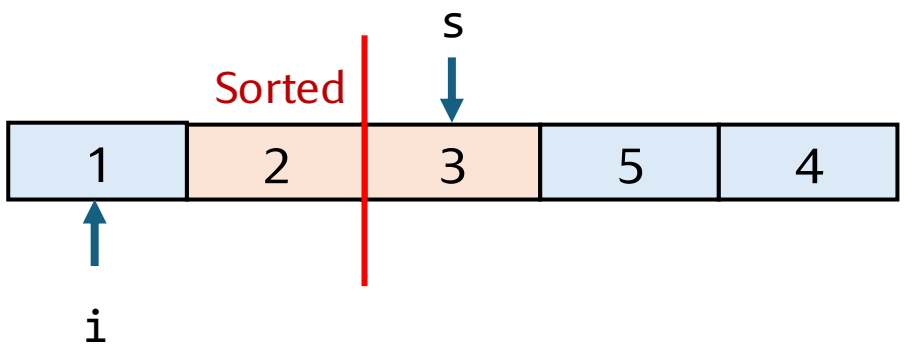
```
void insertion_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    size_t i = s;
    int elem = arr[s];
    while (i > 0 && elem < arr[i - 1]) {
      arr[i] = arr[i - 1];
      i -= 1;
    }
    arr[i] = elem;
  }
}
```



s = 2; i = 1; elem = 1

Insertion Sort with Moving

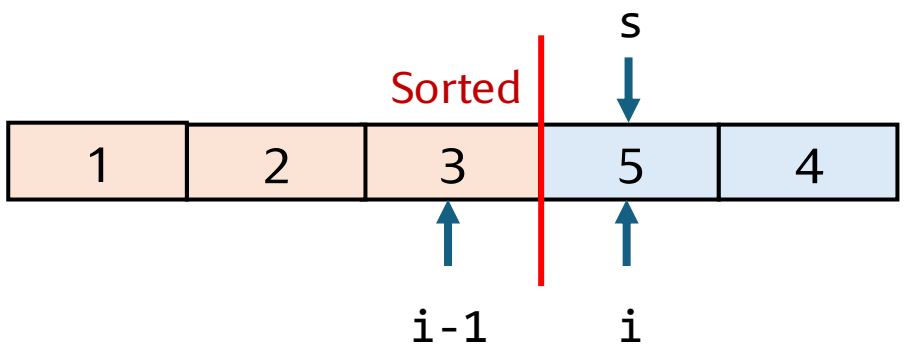
```
void insertion_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    size_t i = s;
    int elem = arr[s];
    while (i > 0 && elem < arr[i - 1]) {
      arr[i] = arr[i - 1];
      i -= 1;
    }
    arr[i] = elem;
  }
}
```



`s = 2; i = 0; elem = 1`

Insertion Sort with Moving

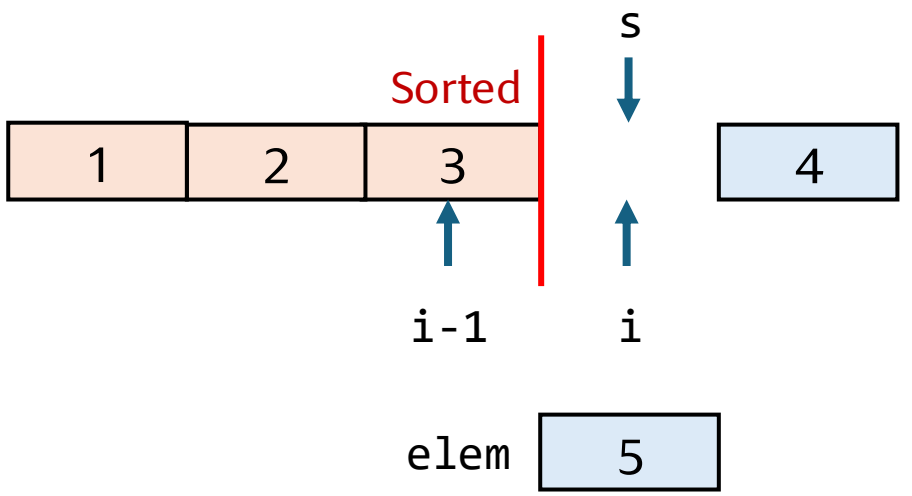
```
void insertion_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    size_t i = s;
    int elem = arr[s];
    while (i > 0 && elem < arr[i - 1]) {
      arr[i] = arr[i - 1];
      i -= 1;
    }
    arr[i] = elem;
  }
}
```



s = 3; i = 3

Insertion Sort with Moving

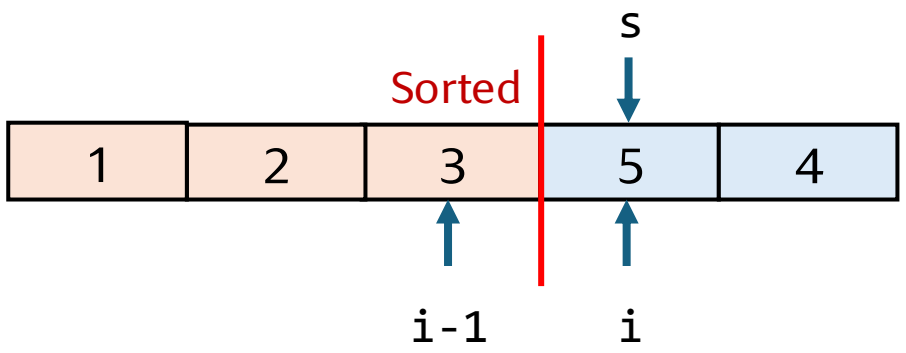
```
void insertion_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    size_t i = s;
    int elem = arr[s];
    while (i > 0 && elem < arr[i - 1]) {
      arr[i] = arr[i - 1];
      i -= 1;
    }
    arr[i] = elem;
  }
}
```



s = 3; i = 3; elem = 5

Insertion Sort with Moving

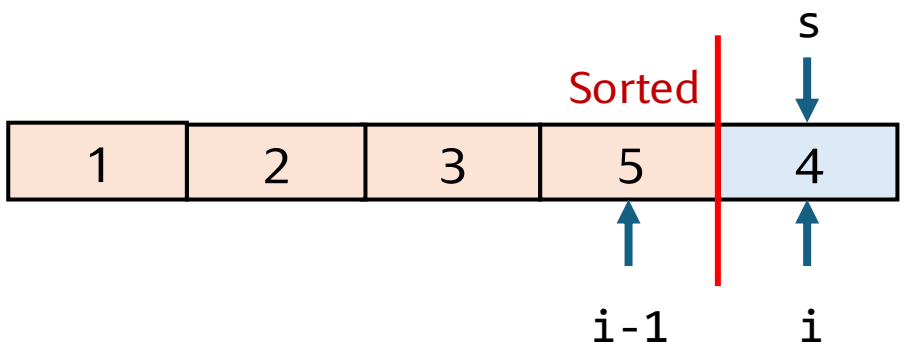
```
void insertion_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    size_t i = s;
    int elem = arr[s];
    while (i > 0 && elem < arr[i - 1]) {
      arr[i] = arr[i - 1];
      i -= 1;
    }
    arr[i] = elem;
  }
}
```



s = 3; i = 3; elem = 5

Insertion Sort with Moving

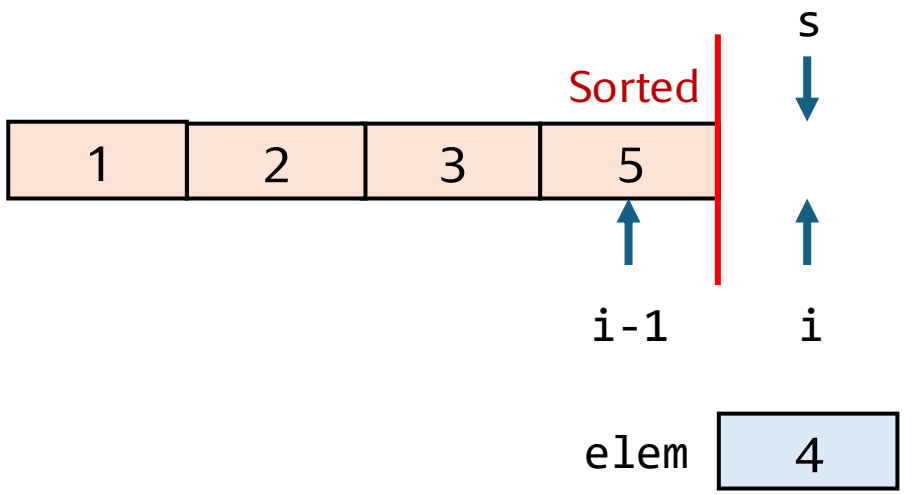
```
void insertion_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    size_t i = s;
    int elem = arr[s];
    while (i > 0 && elem < arr[i - 1]) {
      arr[i] = arr[i - 1];
      i -= 1;
    }
    arr[i] = elem;
  }
}
```



s = 4; i = 4

Insertion Sort with Moving

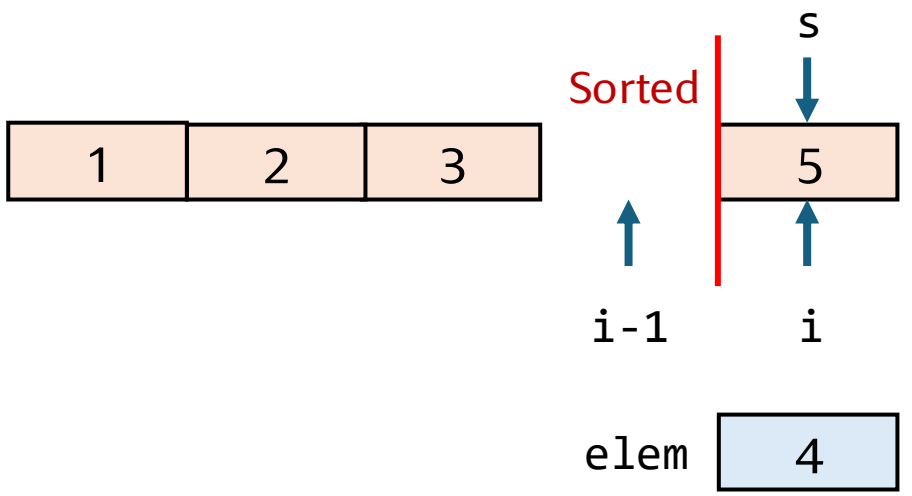
```
void insertion_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    size_t i = s;
    int elem = arr[s];
    while (i > 0 && elem < arr[i - 1]) {
      arr[i] = arr[i - 1];
      i -= 1;
    }
    arr[i] = elem;
  }
}
```



`s = 4; i = 4; elem = 4`

Insertion Sort with Moving

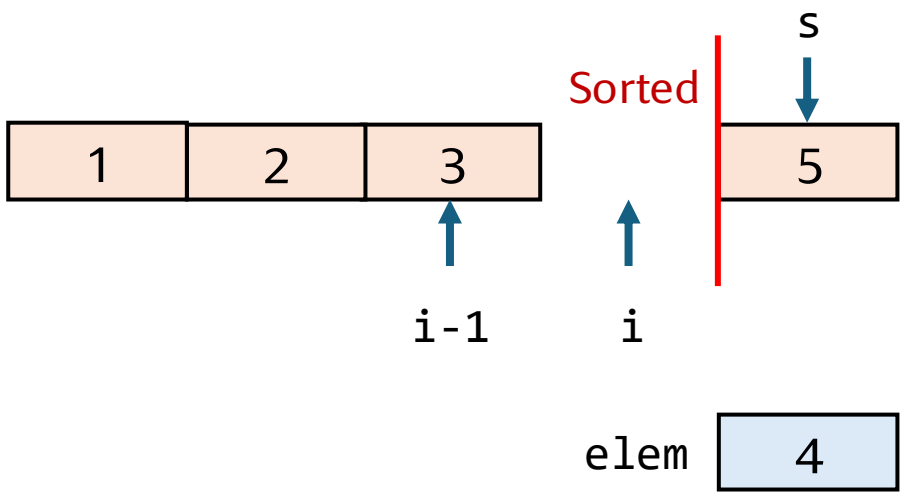
```
void insertion_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    size_t i = s;
    int elem = arr[s];
    while (i > 0 && elem < arr[i - 1]) {
      arr[i] = arr[i - 1];
      i -= 1;
    }
    arr[i] = elem;
  }
}
```



s = 4; i = 4; elem = 4

Insertion Sort with Moving

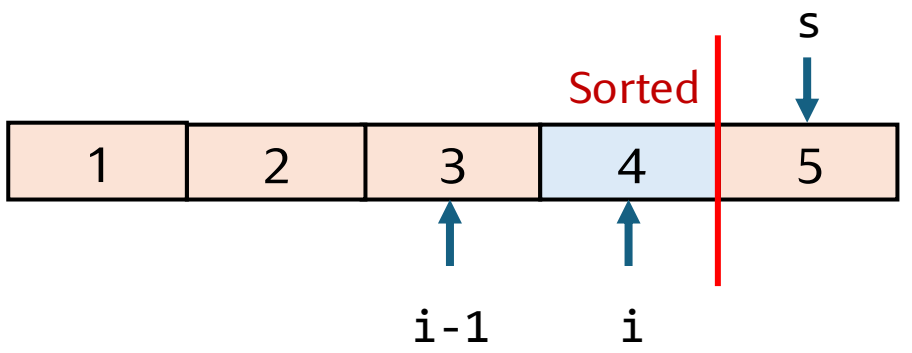
```
void insertion_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    size_t i = s;
    int elem = arr[s];
    while (i > 0 && elem < arr[i - 1]) {
      arr[i] = arr[i - 1];
      i -= 1;
    }
    arr[i] = elem;
  }
}
```



s = 4; i = 3; elem = 4

Insertion Sort with Moving

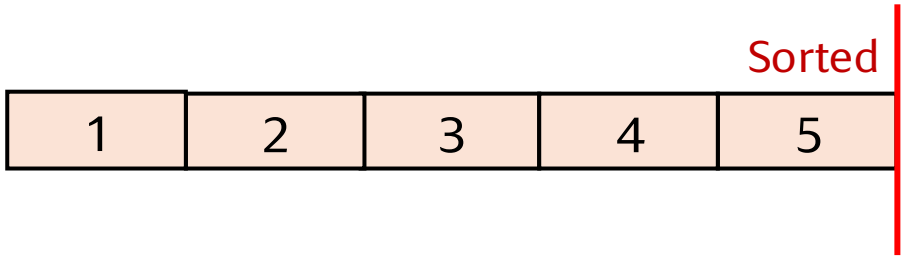
```
void insertion_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    size_t i = s;
    int elem = arr[s];
    while (i > 0 && elem < arr[i - 1]) {
      arr[i] = arr[i - 1];
      i -= 1;
    }
    arr[i] = elem;
  }
}
```



s = 4; i = 3; elem = 4

Insertion Sort with Moving

```
void insertion_sort_v1(int *arr, size_t sz) {
  if (sz <= 1) return;

  for (size_t s = 1; s < sz; s++) {
    size_t i = s;
    int elem = arr[s];
    while (i > 0 && elem < arr[i - 1]) {
      arr[i] = arr[i - 1];
      i -= 1;
    }
    arr[i] = elem;
  }
}
```



Worst-case: $n(n-1)/2$ (comparisons + **moves**)

Best-case: $(n-1)$ comparisons

Swapping vs. Moving

Swapping

```
int tmp = arr[i];     // READ #1
arr[i] = arr[i - 1]; // READ #2, WRITE #1
arr[i-1] = tmp;       // WRITE #2
```

$n(n-1)/2$ (comparisons + **swaps**)

2 reads 2 reads +2 writes

$2n(n-1)$ reads + $n(n-1)$ writes
= (2 reads + write) n^2-(2 reads + write) n

Moving

```
arr[i] = arr[i - 1]; // READ #1, WRITE #1
```

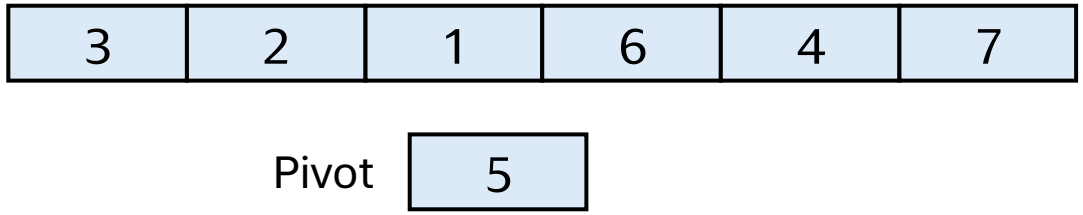
$n(n-1)/2$ (comparisons + **moves**)

2 reads 1 read + 1 write

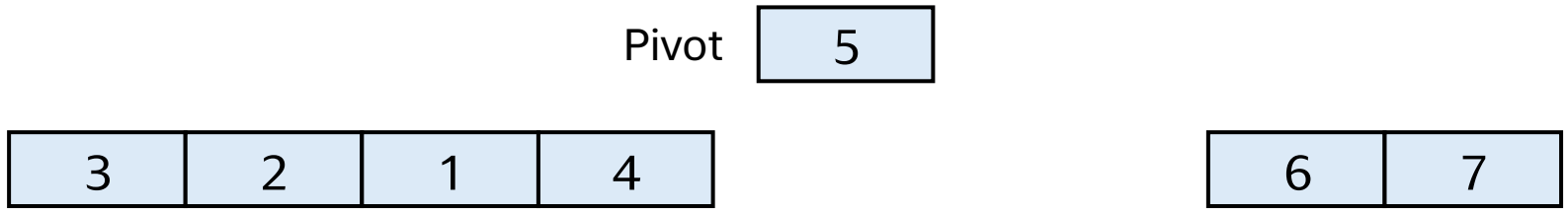
$1.5n(n-1)$ reads + $0.5n(n-1)$ writes
= (1.5 reads + 0.5 writes) n^2 – (1.5 reads + 0.5 writes) n

Quick Sort

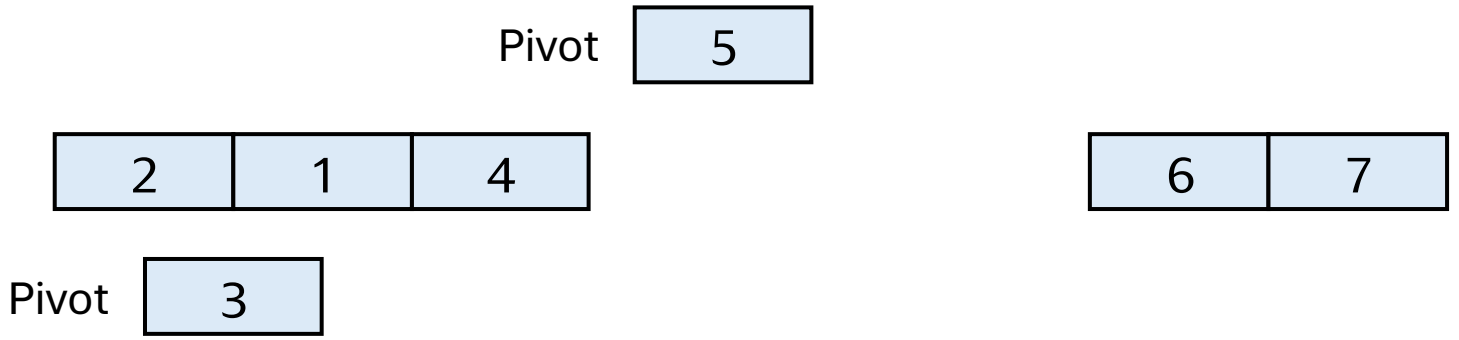
Quick Sort



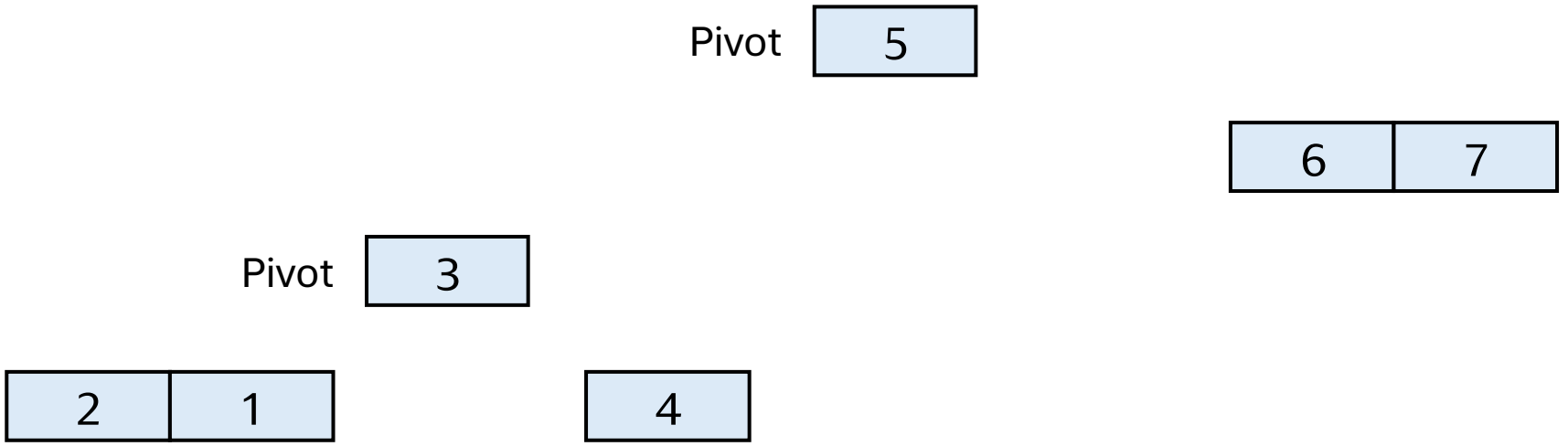
Quick Sort



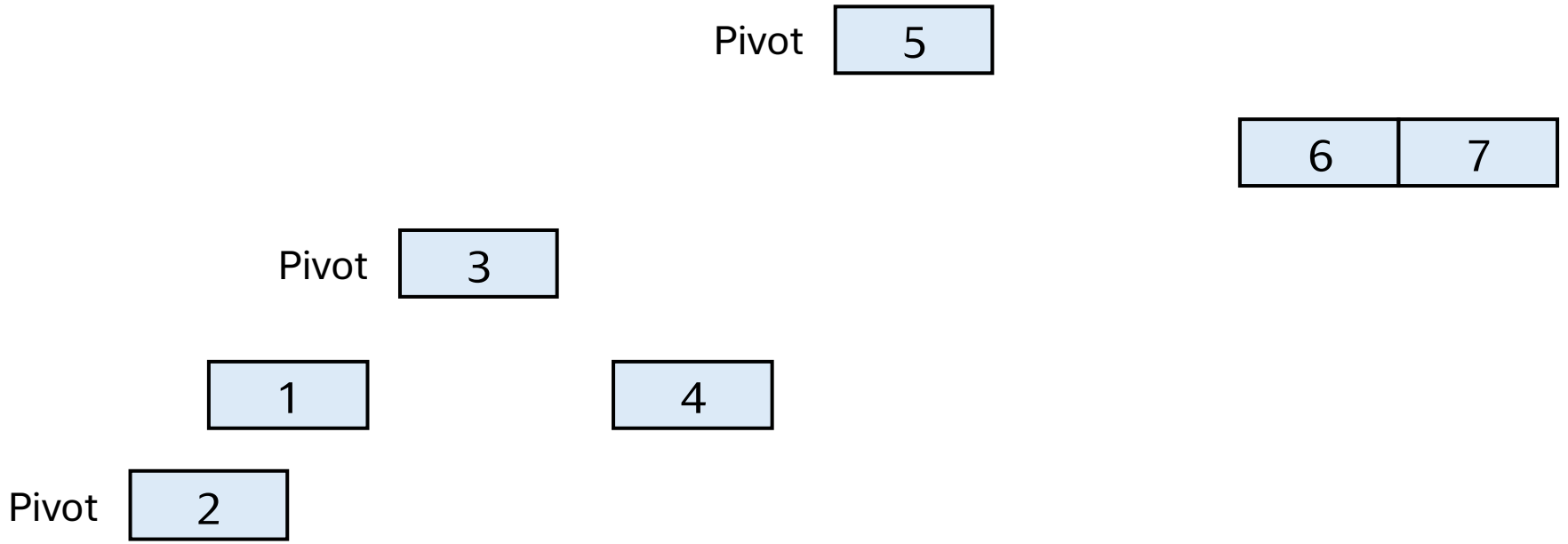
Quick Sort



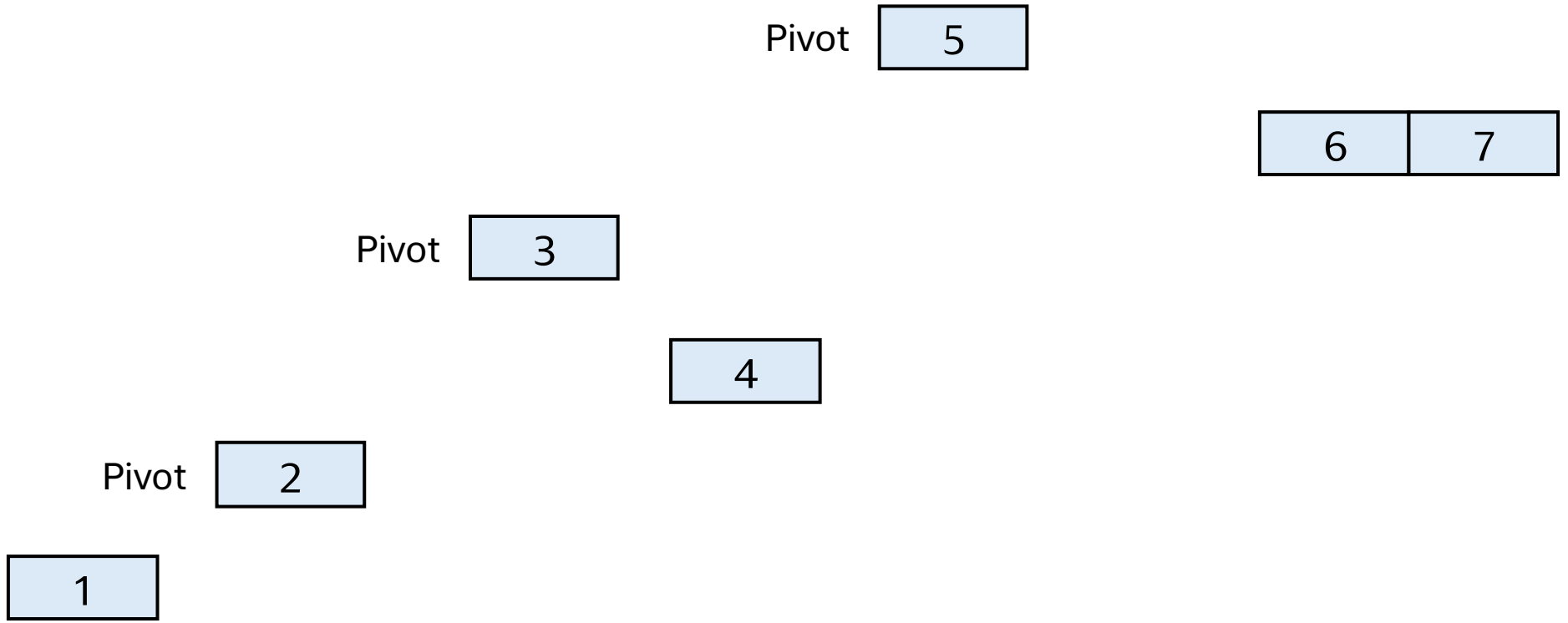
Quick Sort



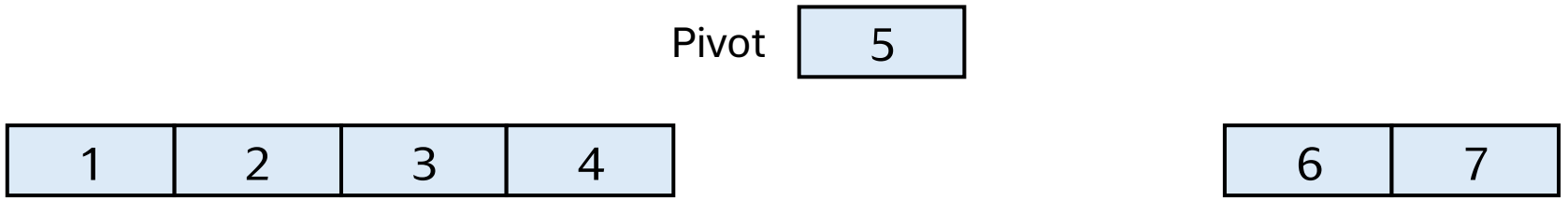
Quick Sort



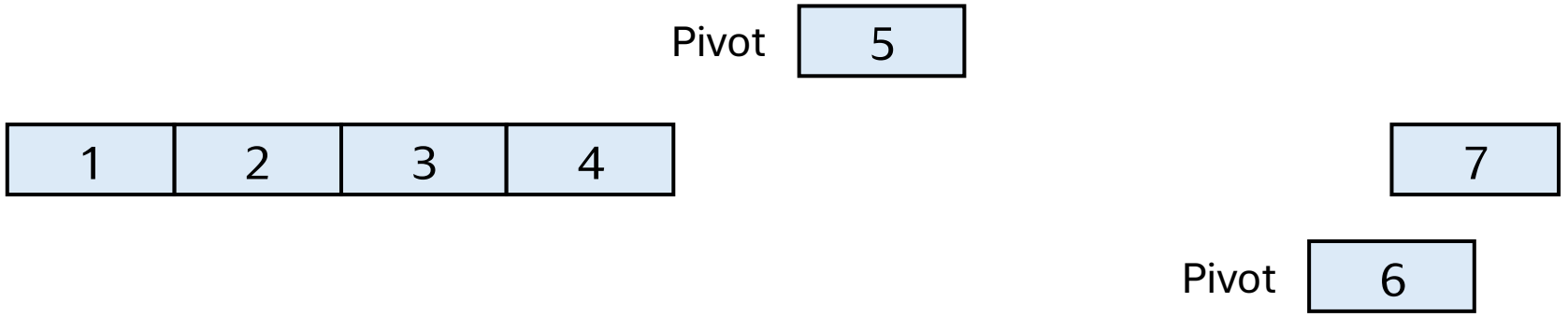
Quick Sort



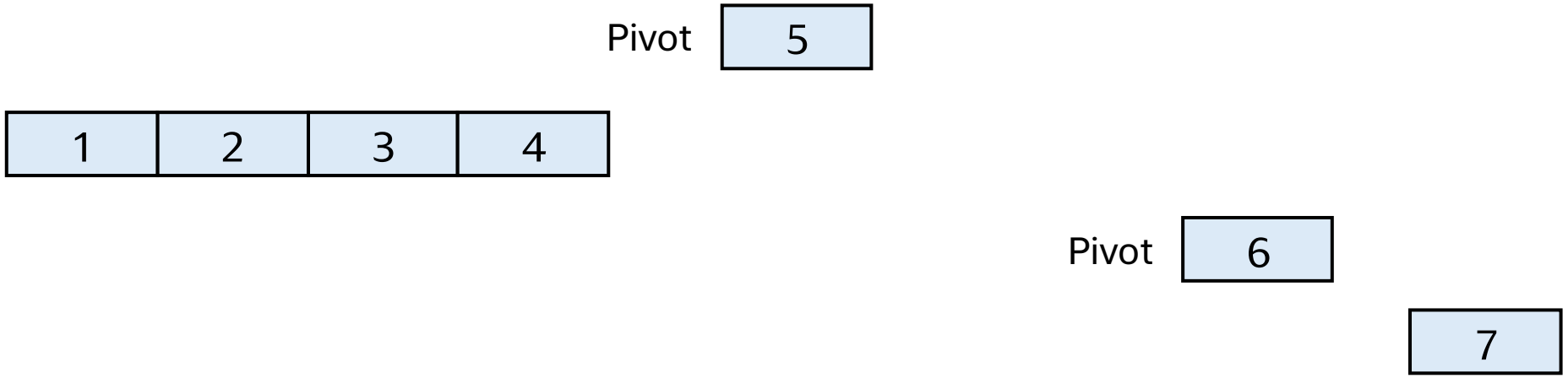
Quick Sort



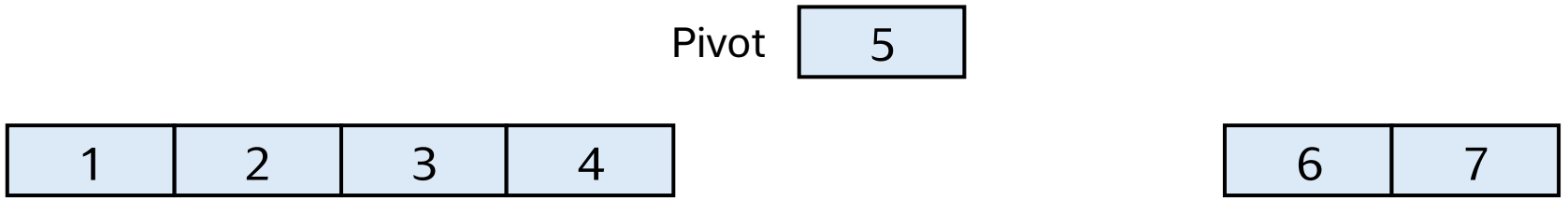
Quick Sort



Quick Sort



Quick Sort



Quick Sort

Quick Sort