ECE 312 (SP26):

# Recursion

Neil Zhao

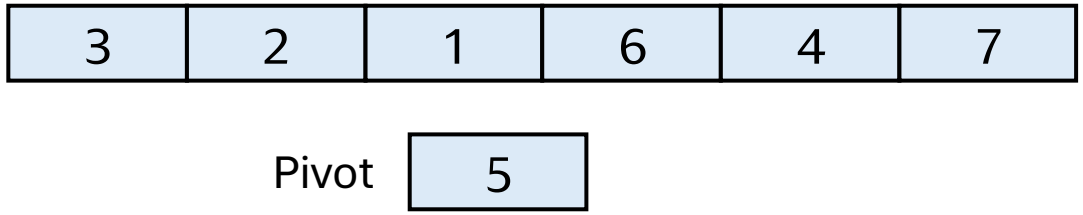
neil.zhao@utexas.edu

# Recursion

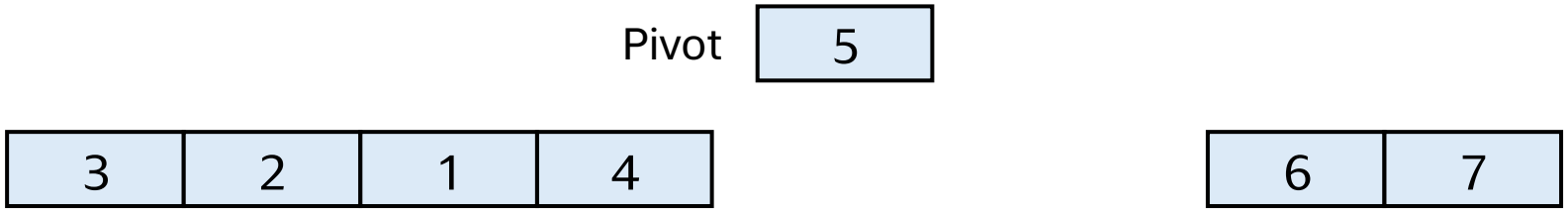
The power of realizing it’s the same problem at a different scale

# Quick Sort

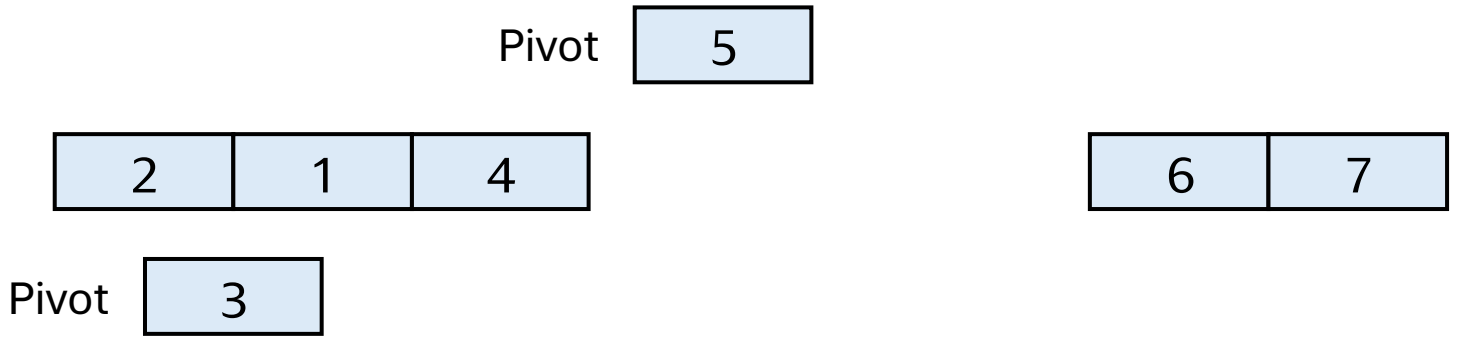
# Quick Sort



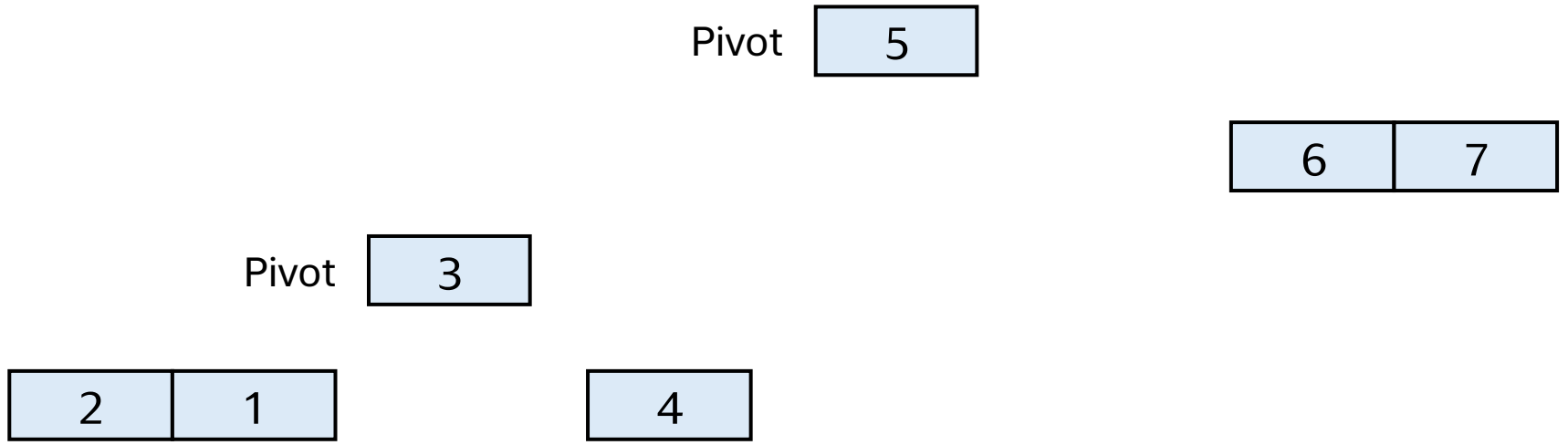
# Quick Sort



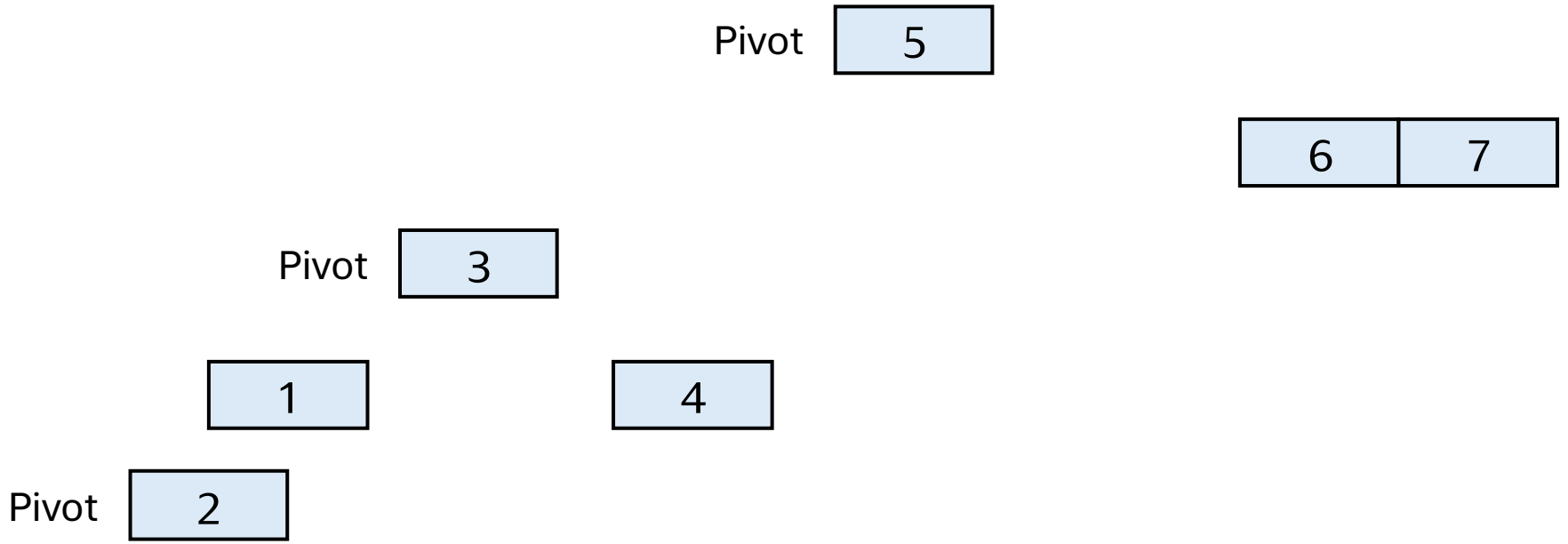
# Quick Sort



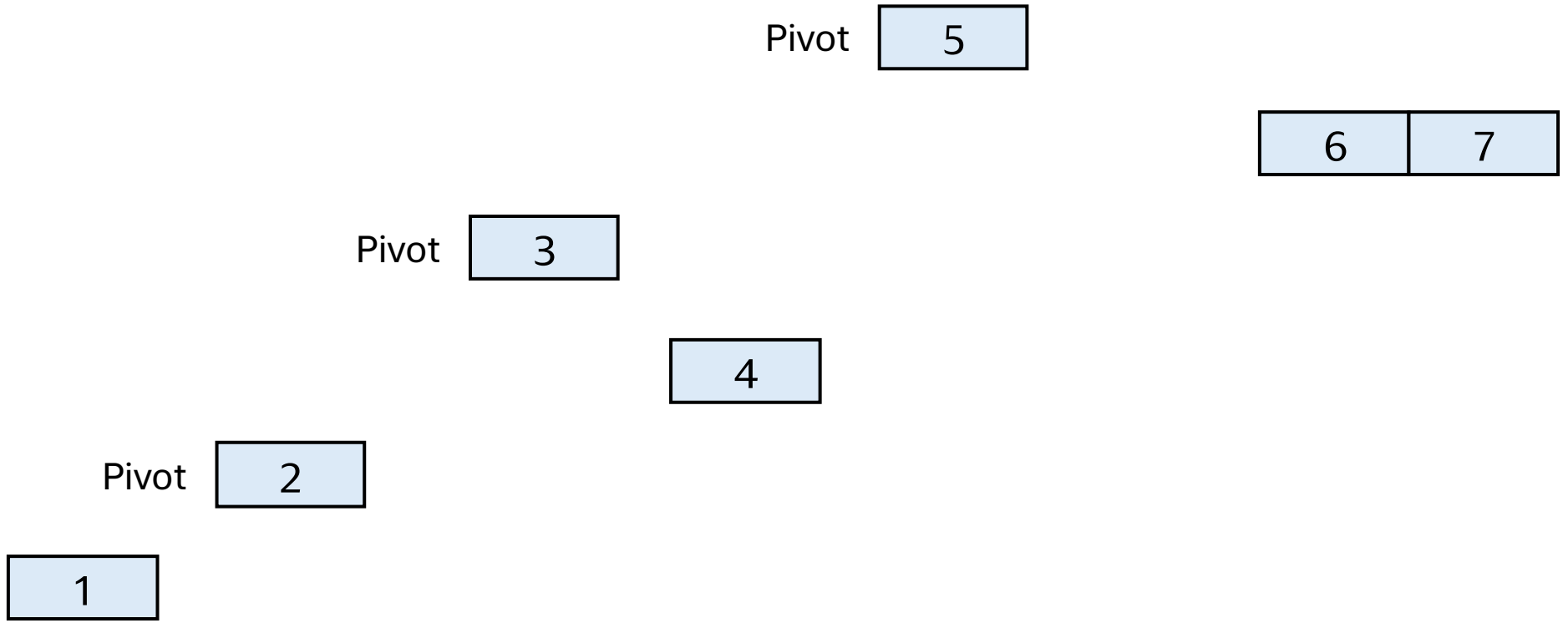
# Quick Sort



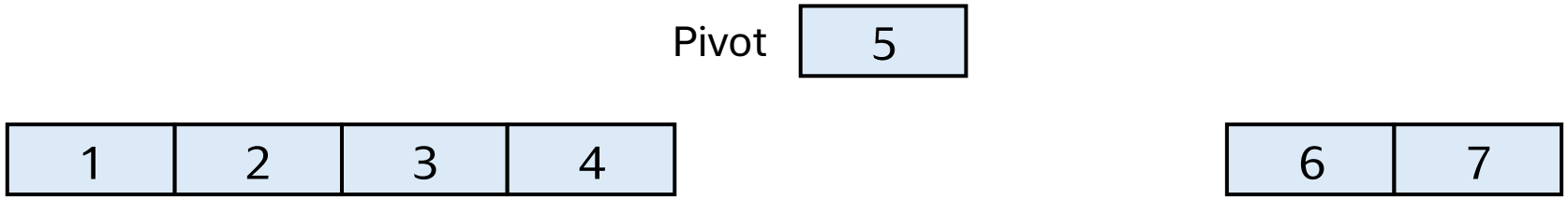
# Quick Sort



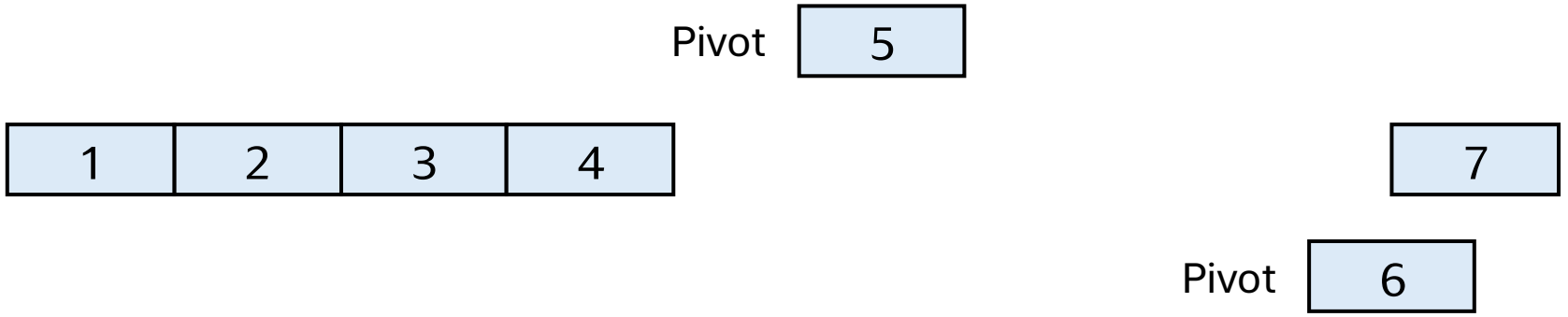
# Quick Sort



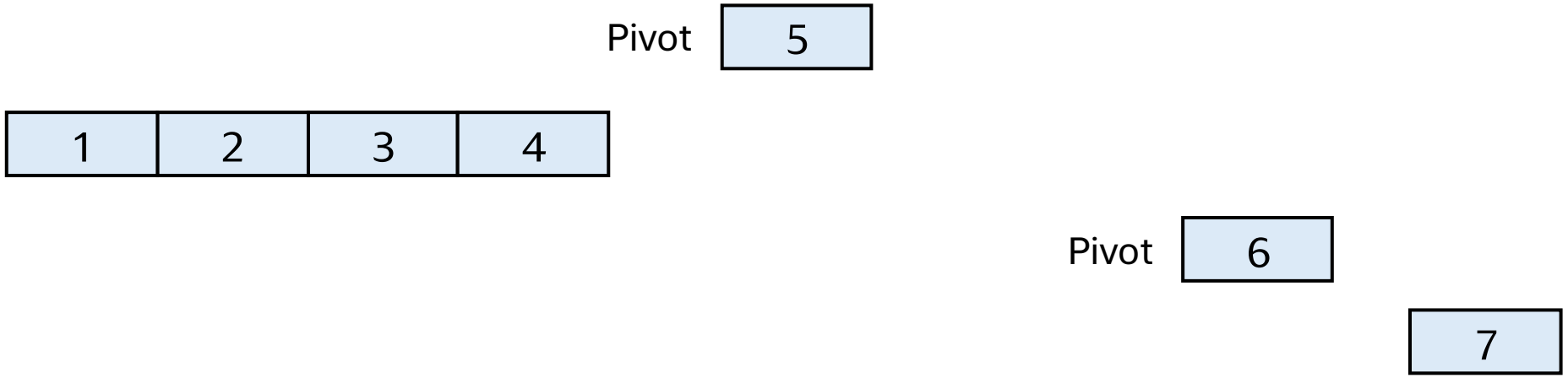
# Quick Sort



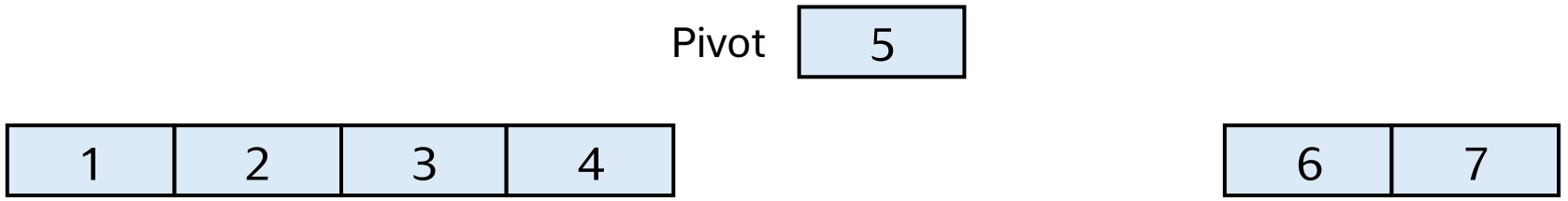
# Quick Sort



# Quick Sort

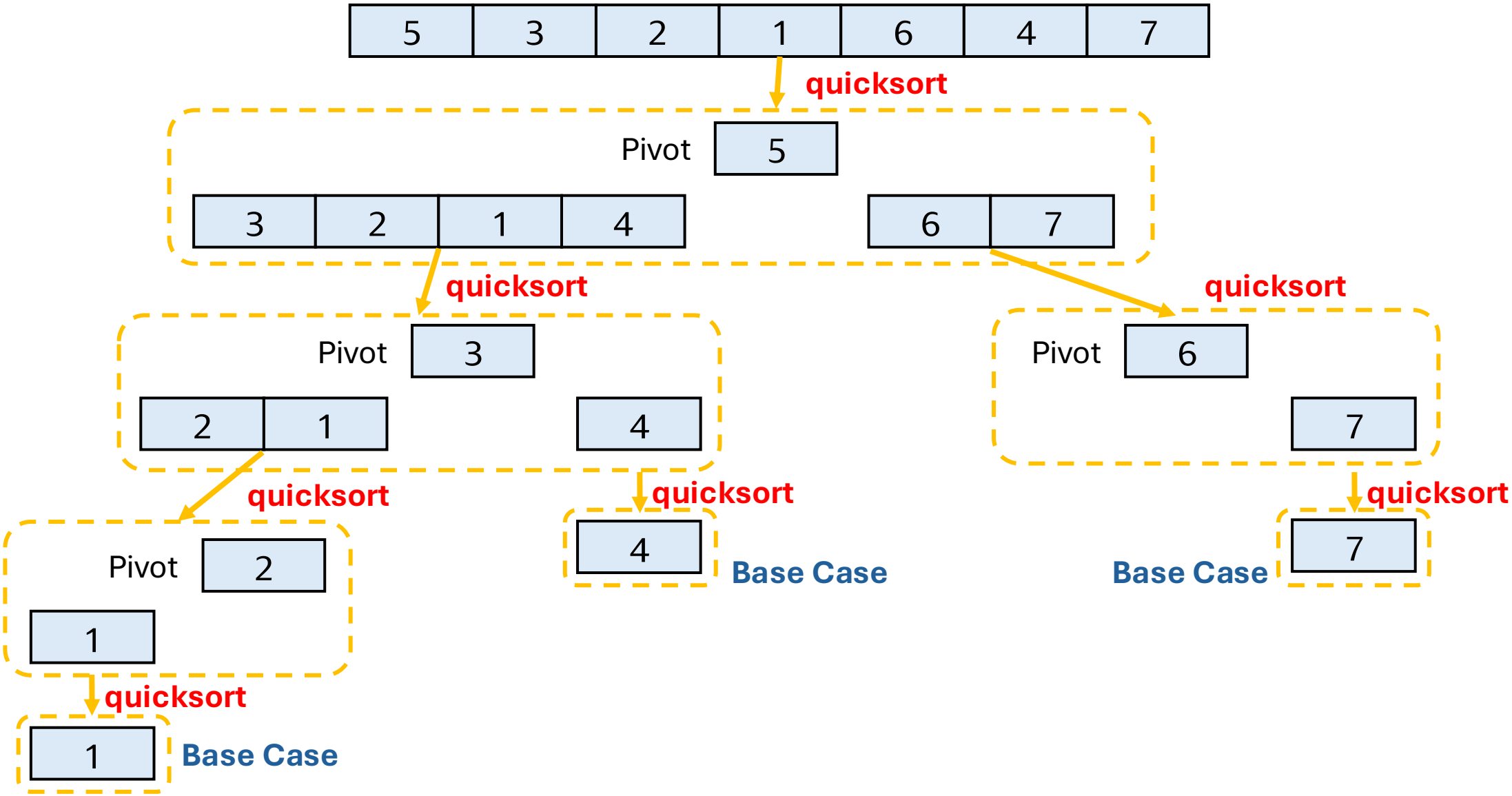


# Quick Sort



# Quick Sort

# Quick Sort


5 3 2 1 6 4 7
quicksort
Pivot 5
3 2 1 4
6 7
quicksort
quicksort
Pivot 3
2 1
4
Pivot 6
7
quicksort
quicksort
quicksort
Pivot 2
4
Base Case
7
Base Case
1
quicksort
1
Base Case

# Example: Sum an Array


**Recursive case:**
The sum of an array is its first element plus the sum of the rest of the array

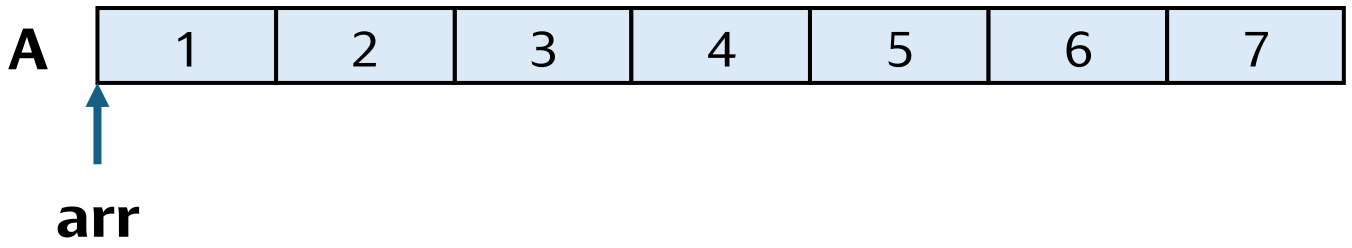
$$sum(arr[0..N]) = arr[0] + sum(arr[1..N])$$

**Base case:**
The sum of an empty array is zero

```
int sum_arr(int *arr, size_t len) {
  if (len == 0) return 0; // base case
  return arr[0] + sum_arr(arr + 1, len - 1); // recursive case
}
```

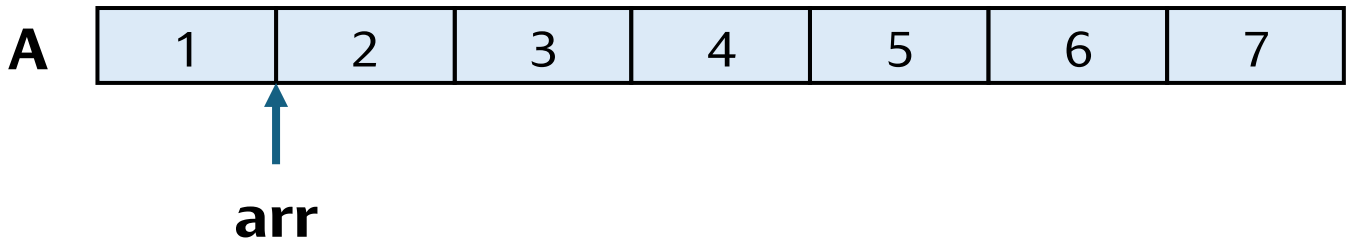
The problem is getting smaller

# Example: Sum an Array

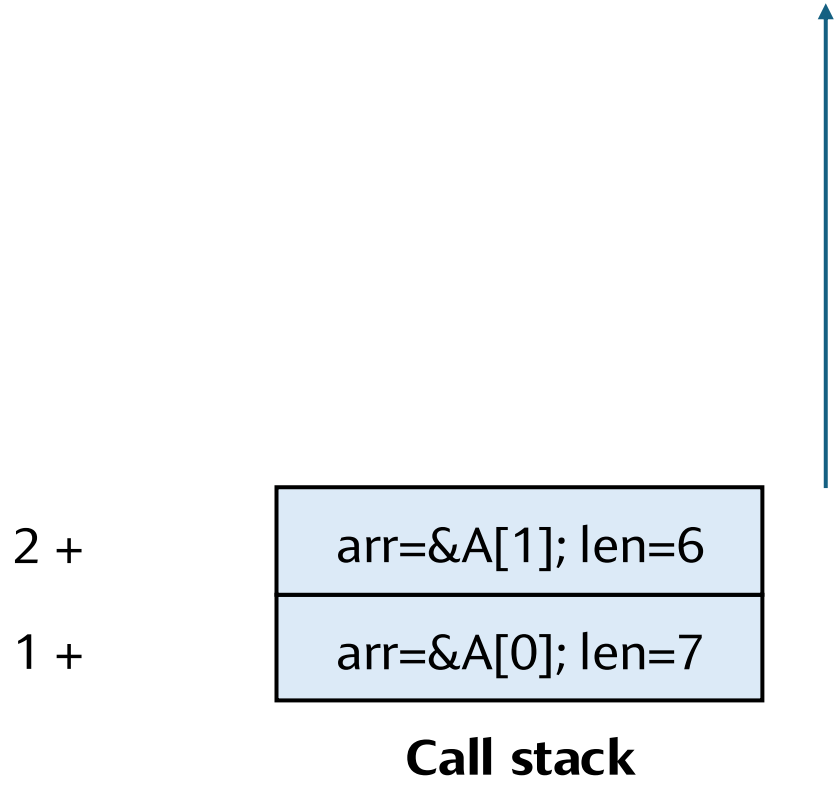



```
int sum_arr(int *arr, size_t len) {
  if (len == 0) return 0;
  return arr[0] + sum_arr(arr + 1, len - 1);
}
```

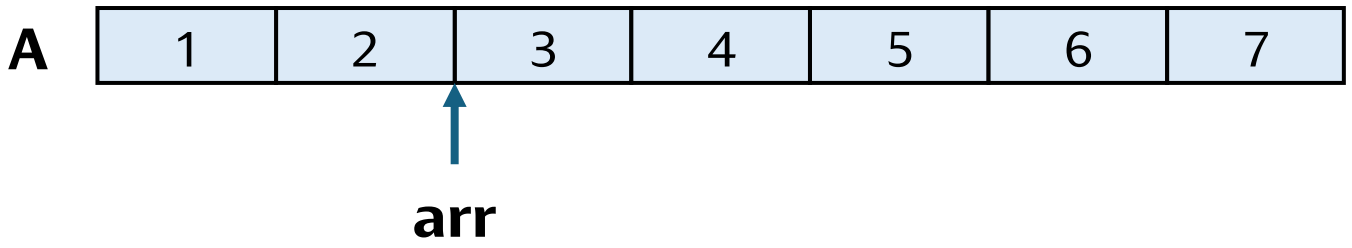
# Example: Sum an Array



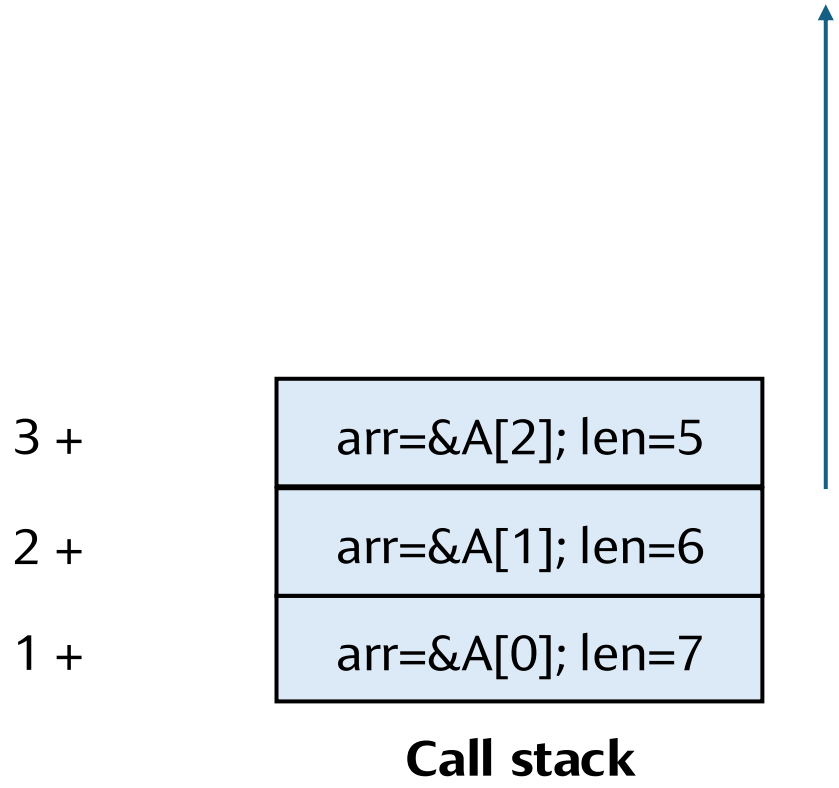

```
int sum_arr(int *arr, size_t len) {
  if (len == 0) return 0;
  return arr[0] + sum_arr(arr + 1, len - 1);
}
```



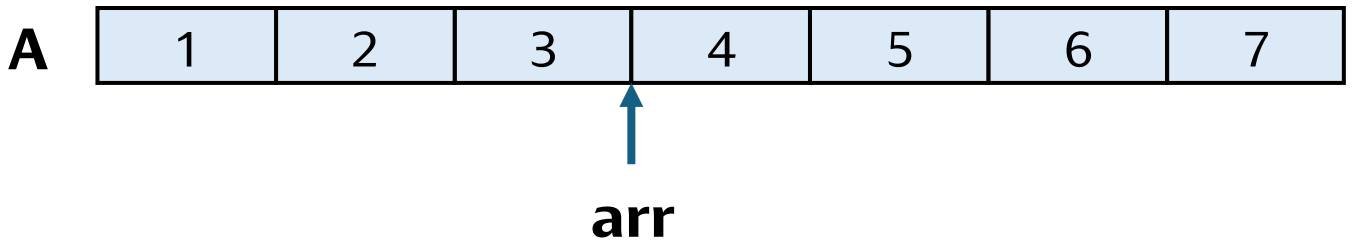
# Example: Sum an Array



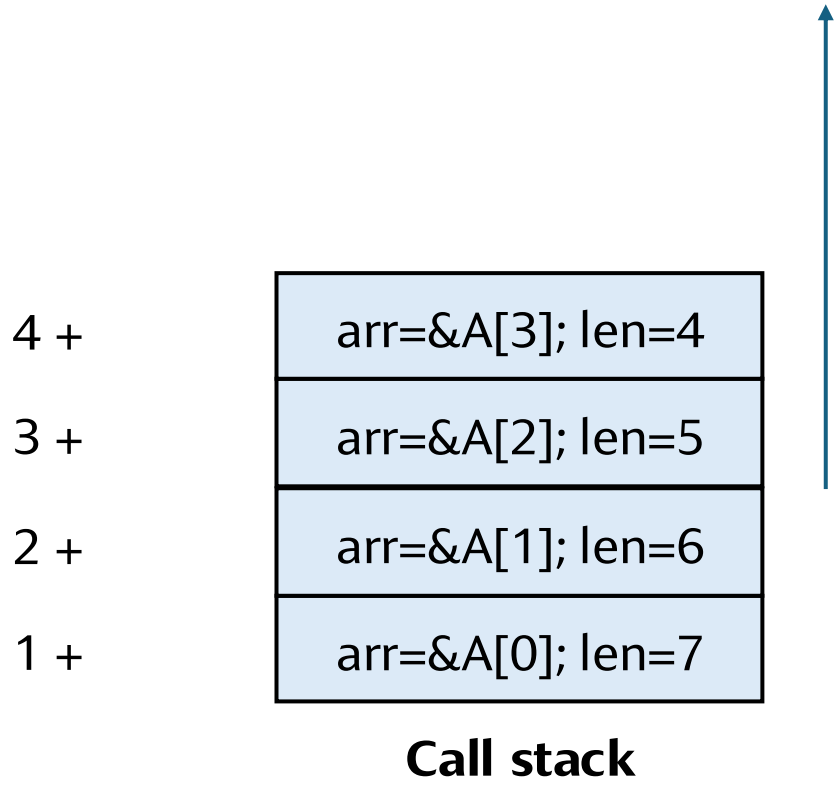

```
int sum_arr(int *arr, size_t len) {
  if (len == 0) return 0;
  return arr[0] + sum_arr(arr + 1, len - 1);
}
```



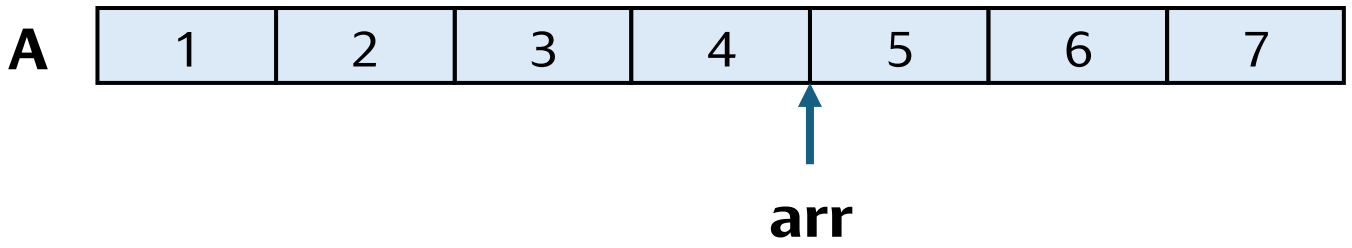
# Example: Sum an Array



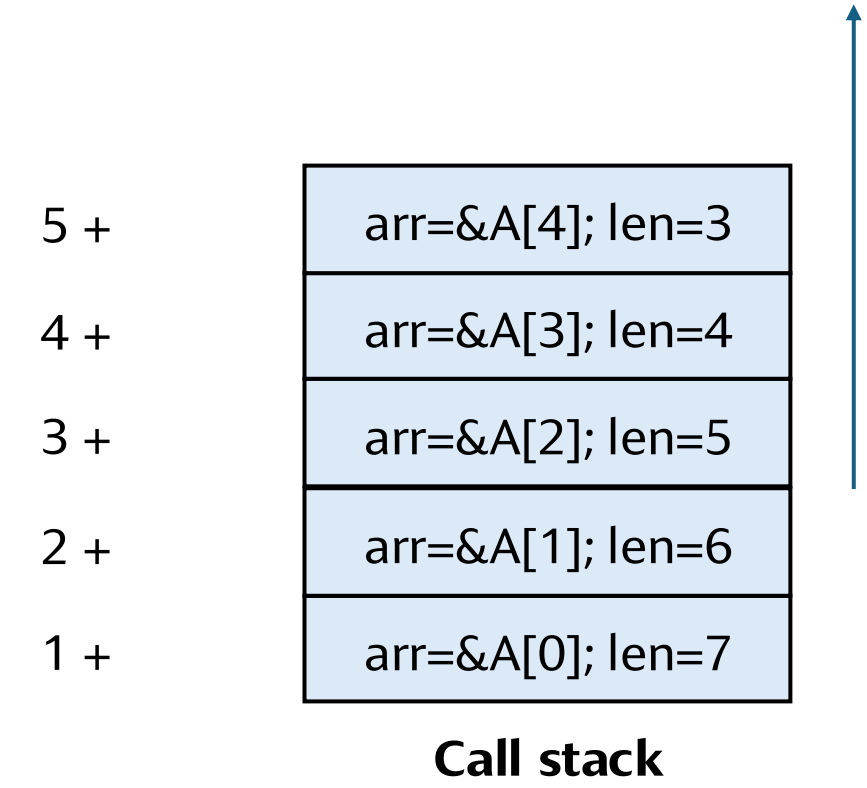

```
int sum_arr(int *arr, size_t len) {
  if (len == 0) return 0;
  return arr[0] + sum_arr(arr + 1, len - 1);
}
```



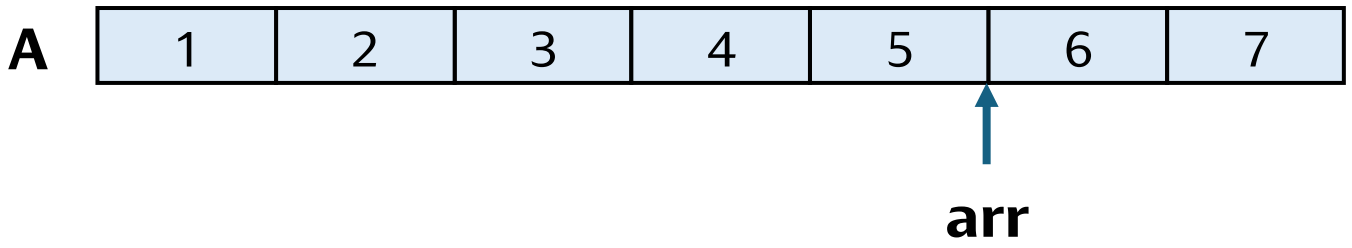
# Example: Sum an Array



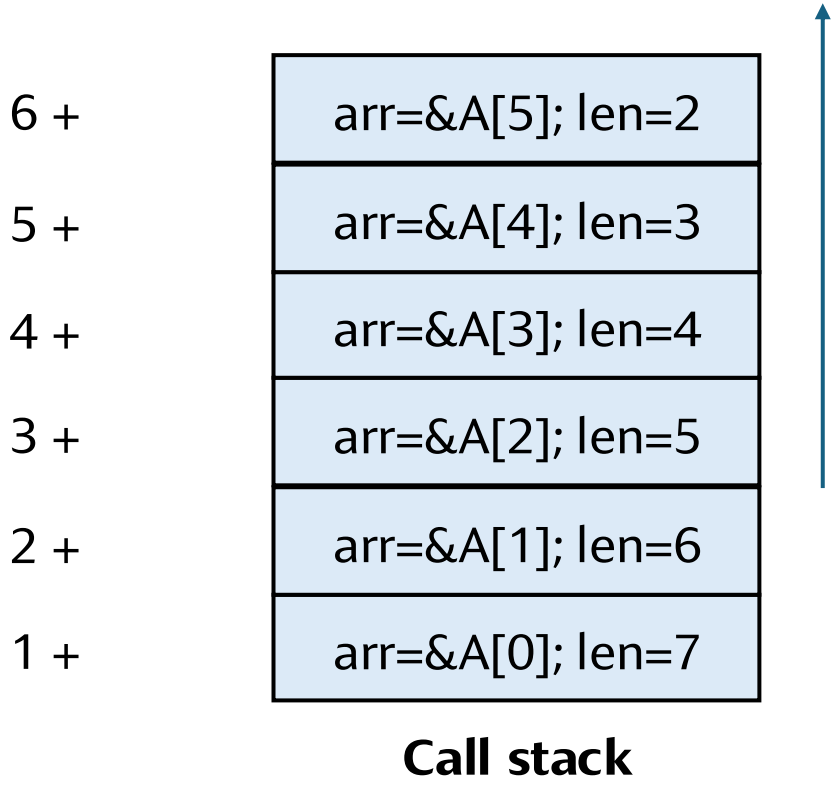

```
int sum_arr(int *arr, size_t len) {
  if (len == 0) return 0;
  return arr[0] + sum_arr(arr + 1, len - 1);
}
```



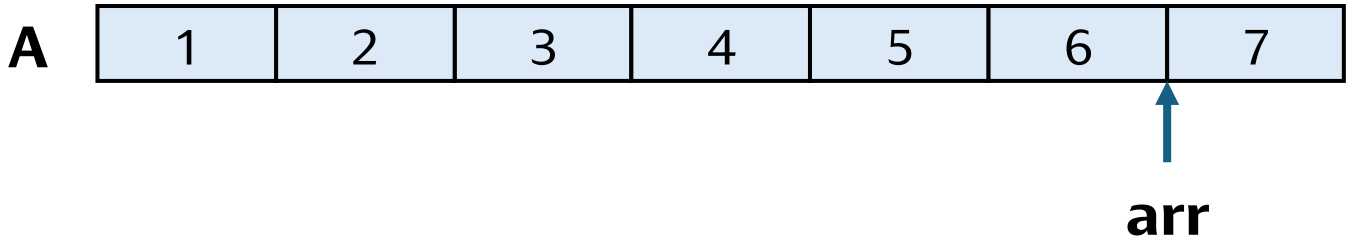
# Example: Sum an Array



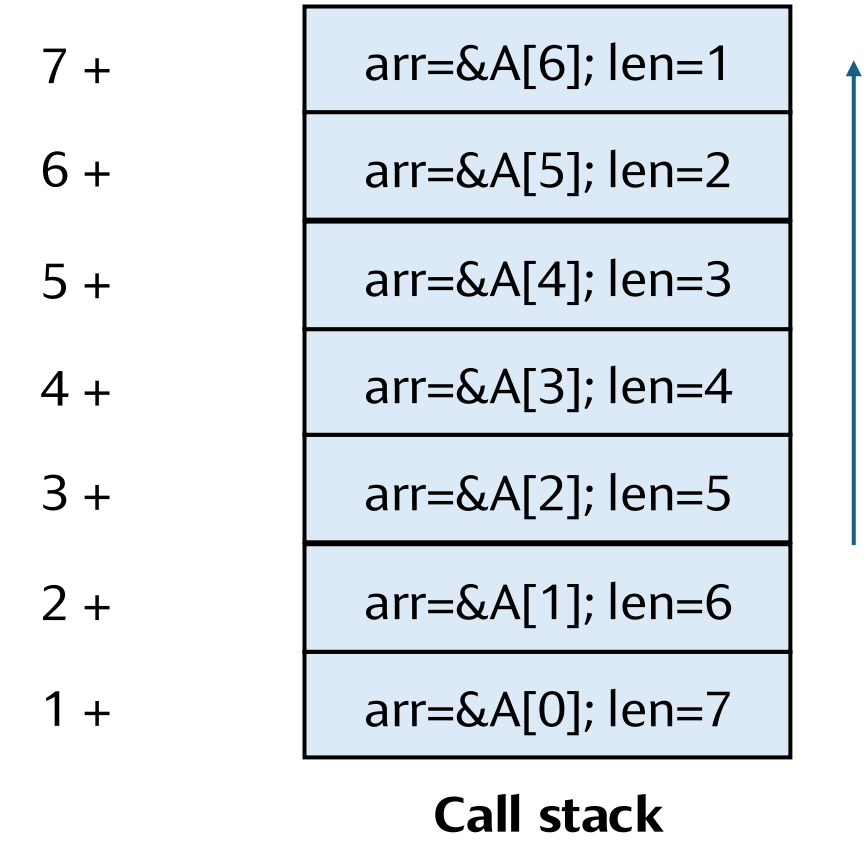

```
int sum_arr(int *arr, size_t len) {
  if (len == 0) return 0;
  return arr[0] + sum_arr(arr + 1, len - 1);
}
```



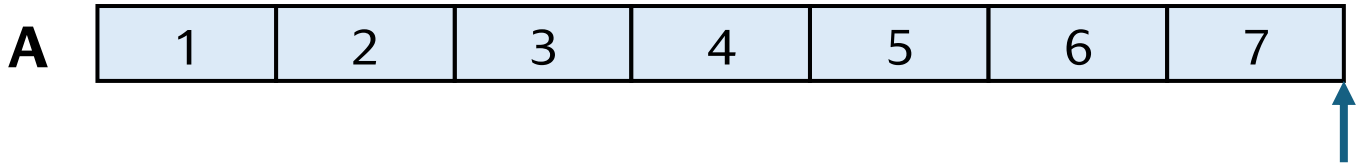
# Example: Sum an Array




```
int sum_arr(int *arr, size_t len) {
  if (len == 0) return 0;
  return arr[0] + sum_arr(arr + 1, len - 1);
}
```



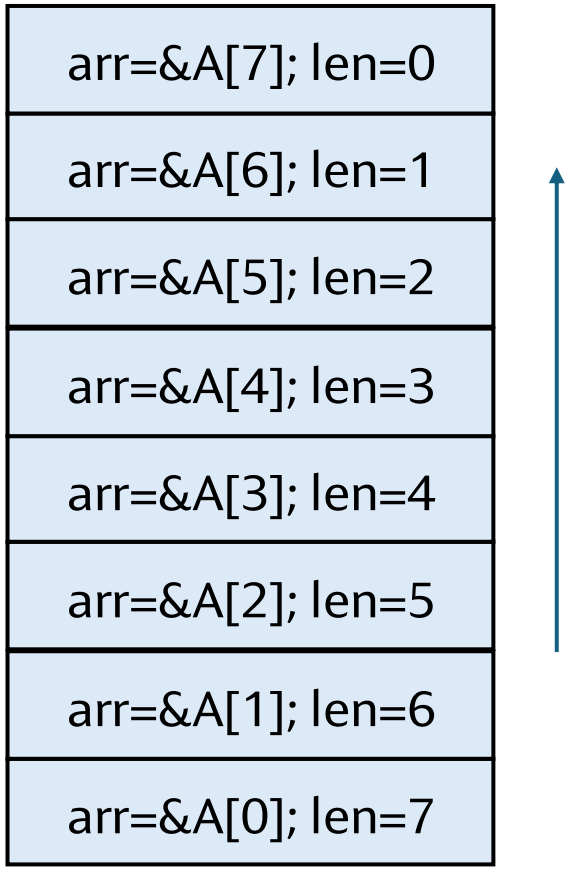
# Example: Sum an Array




**arr**

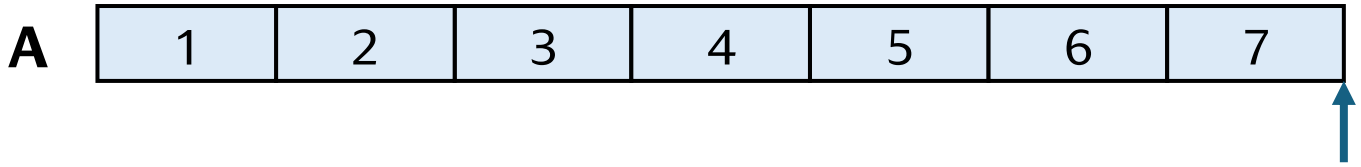
```
int sum_arr(int *arr, size_t len) {
  if (len == 0) return 0;
  return arr[0] + sum_arr(arr + 1, len - 1);
}
```

7 +
6 +
5 +
4 +
3 +
2 +
1 +



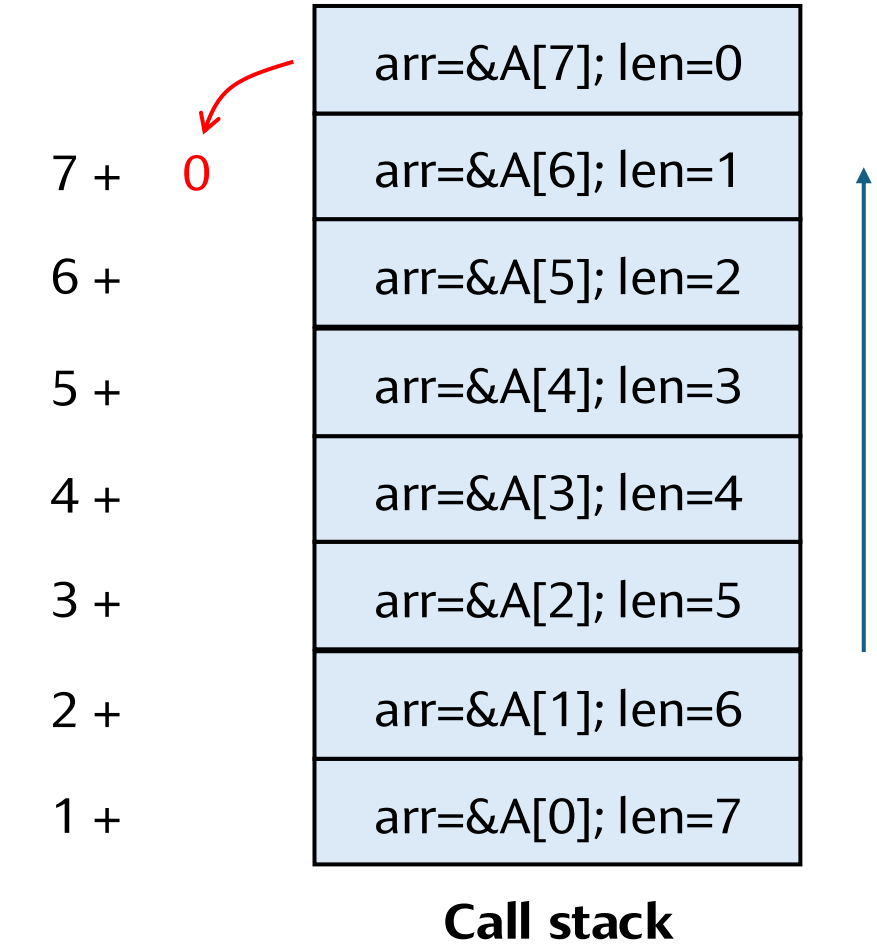

**Call stack**

# Example: Sum an Array

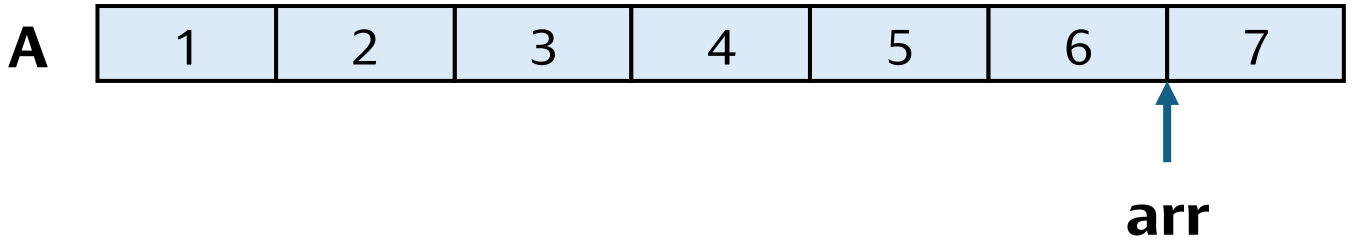



arr

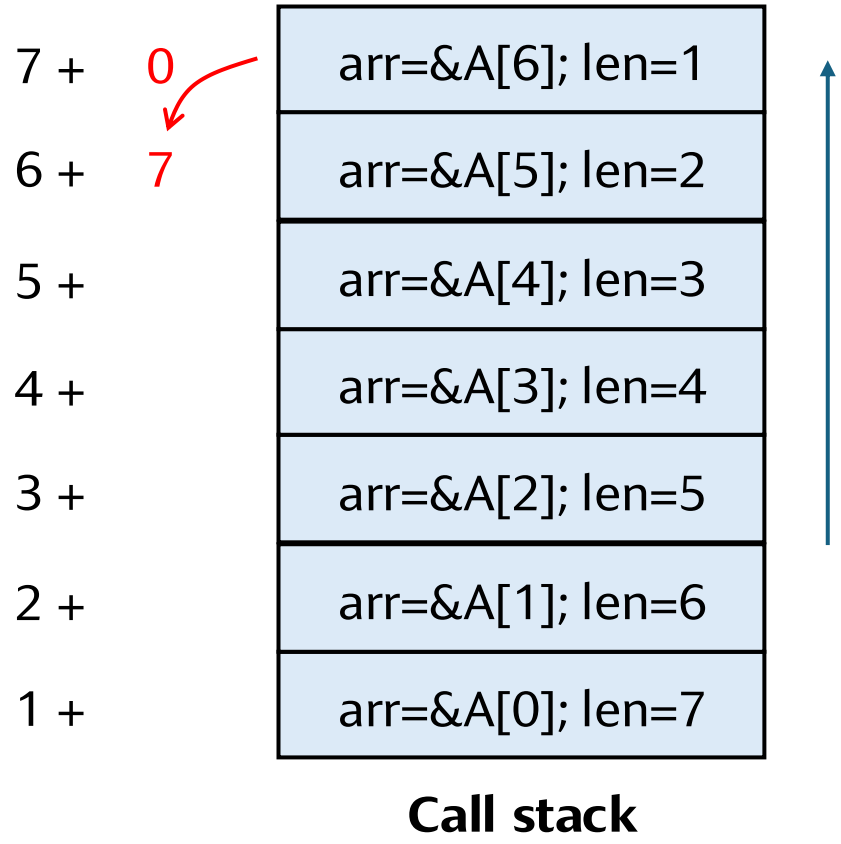
```
int sum_arr(int *arr, size_t len) {
  if (len == 0) return 0;
  return arr[0] + sum_arr(arr + 1, len - 1);
}
```



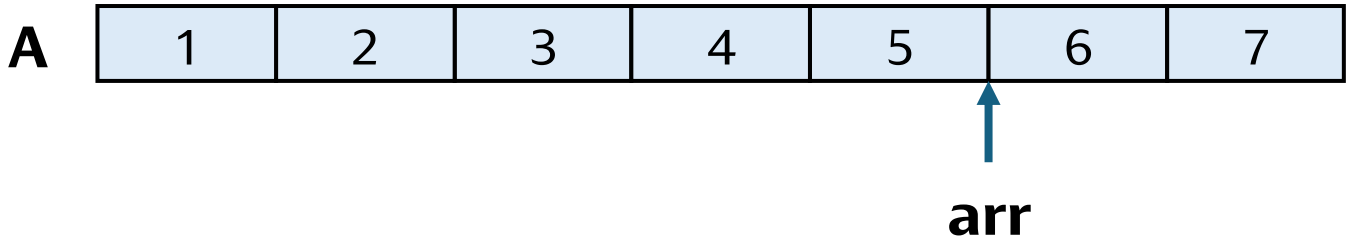
# Example: Sum an Array



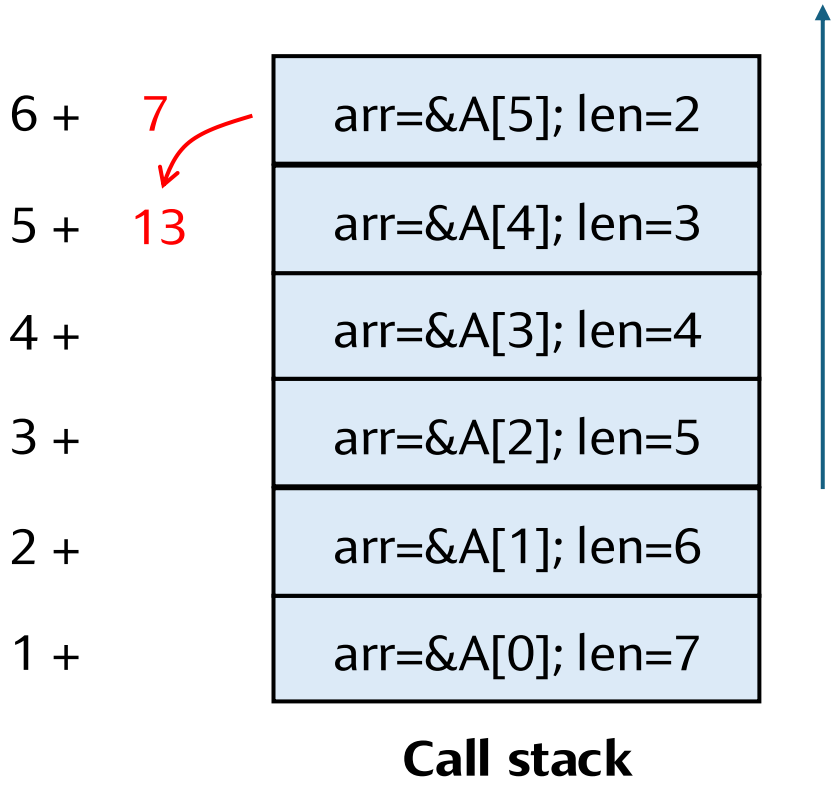

```
int sum_arr(int *arr, size_t len) {
  if (len == 0) return 0;
  return arr[0] + sum_arr(arr + 1, len - 1);
}
```



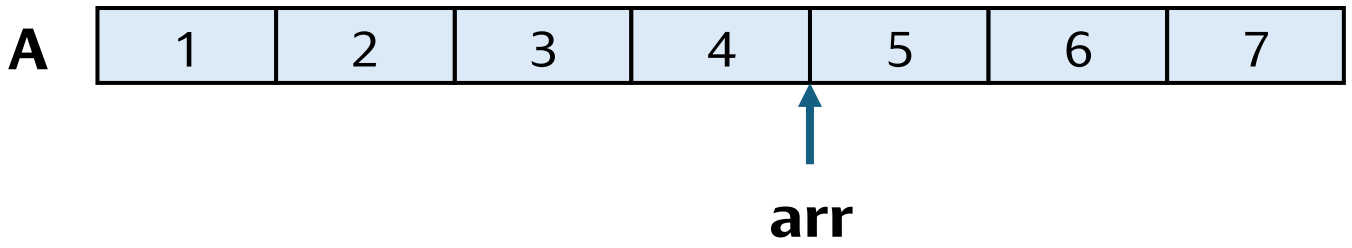
# Example: Sum an Array



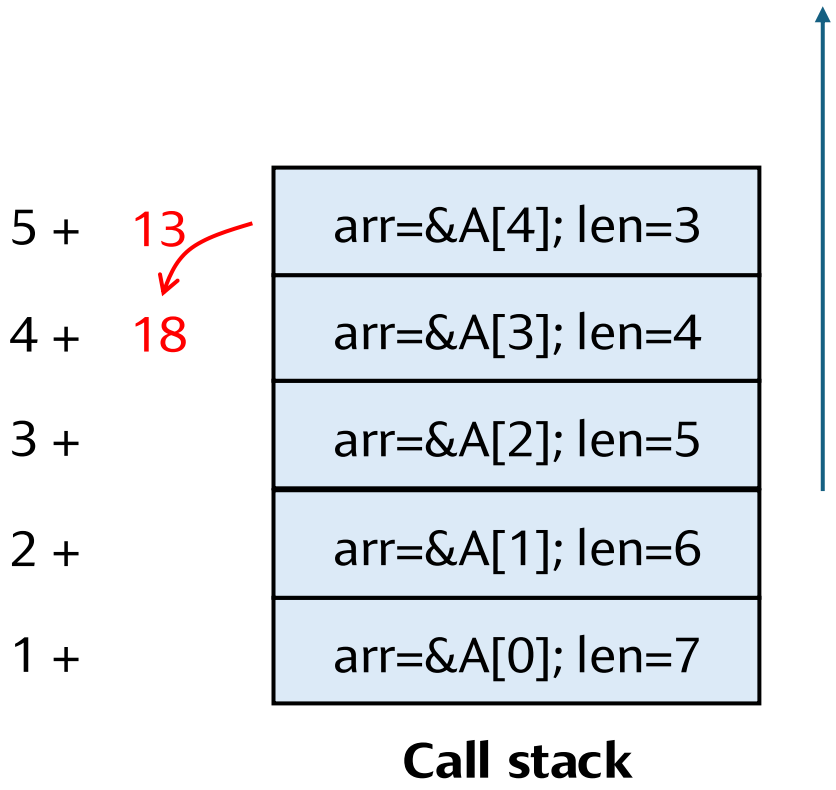

```
int sum_arr(int *arr, size_t len) {
  if (len == 0) return 0;
  return arr[0] + sum_arr(arr + 1, len - 1);
}
```



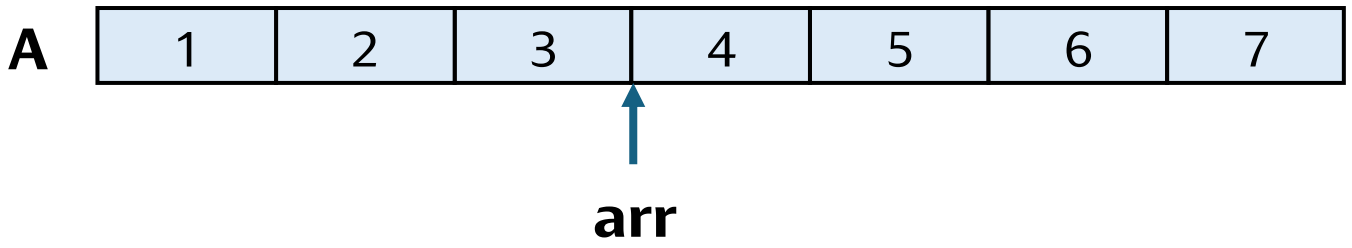
# Example: Sum an Array



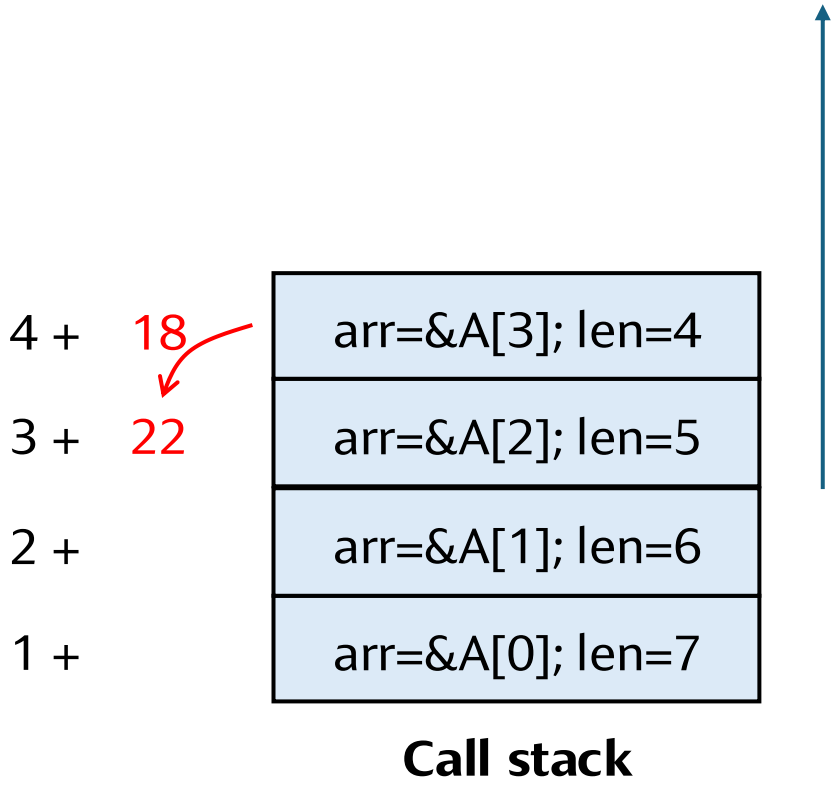

```
int sum_arr(int *arr, size_t len) {
  if (len == 0) return 0;
  return arr[0] + sum_arr(arr + 1, len - 1);
}
```



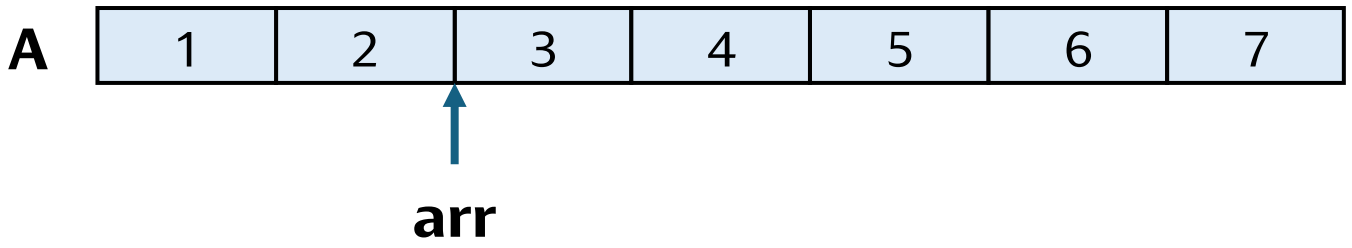
# Example: Sum an Array



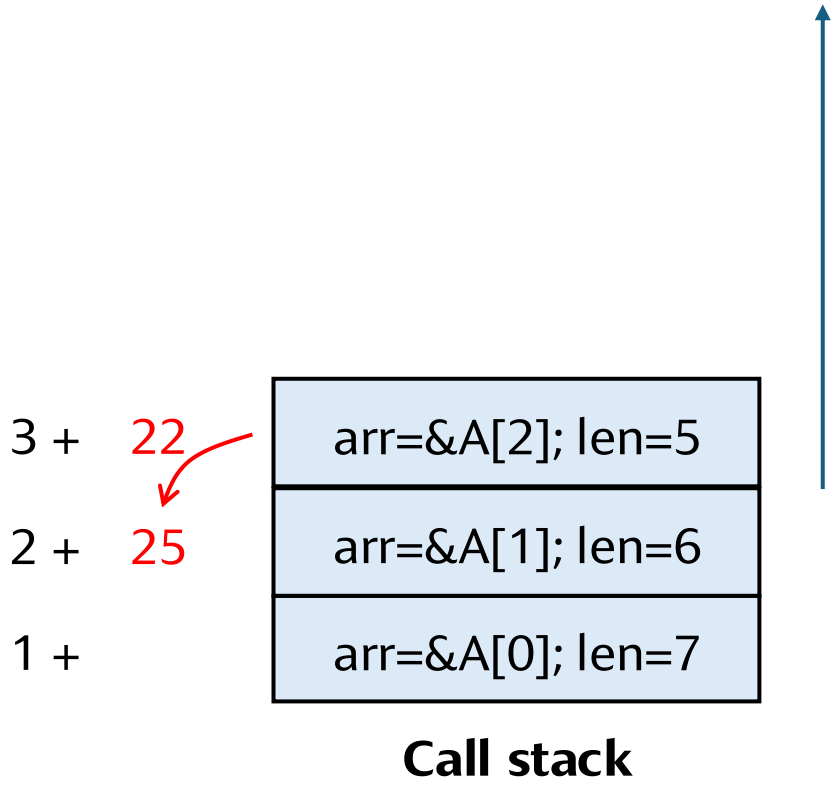

```
int sum_arr(int *arr, size_t len) {
  if (len == 0) return 0;
  return arr[0] + sum_arr(arr + 1, len - 1);
}
```



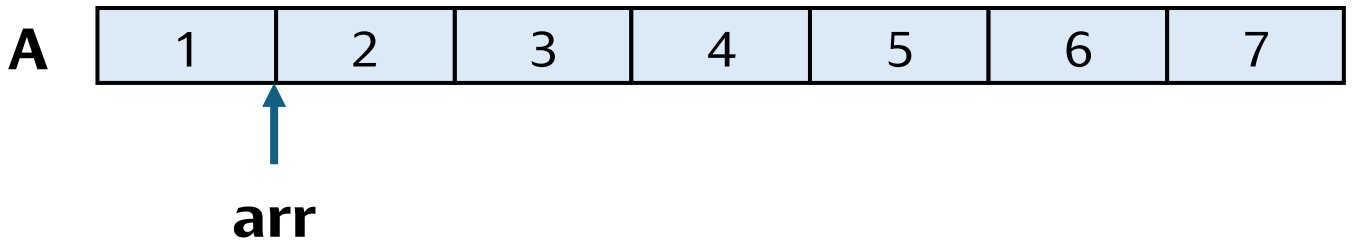
# Example: Sum an Array



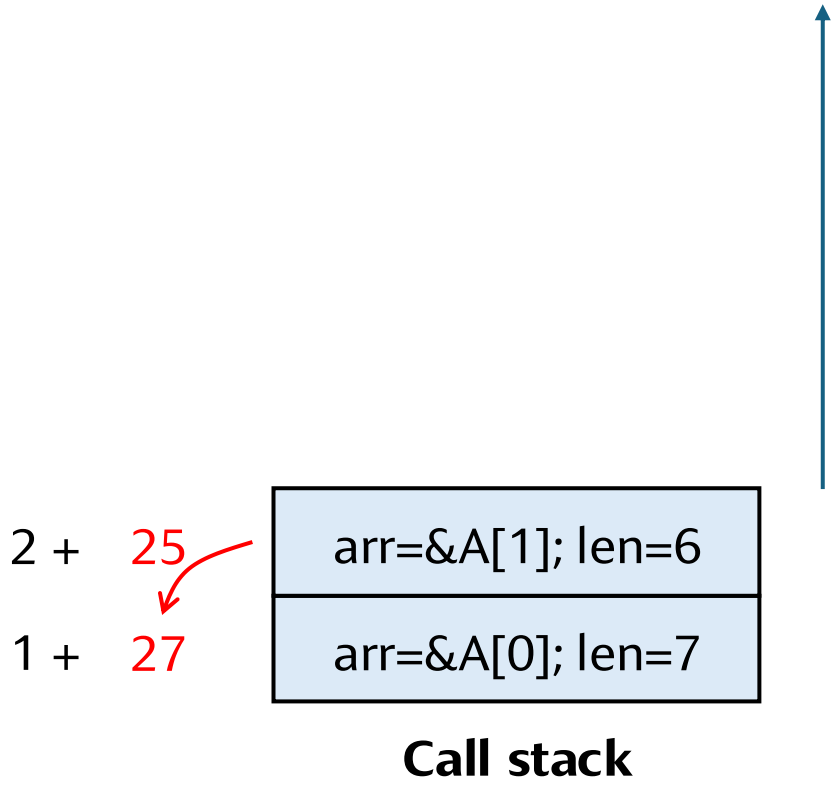

```
int sum_arr(int *arr, size_t len) {
  if (len == 0) return 0;
  return arr[0] + sum_arr(arr + 1, len - 1);
}
```



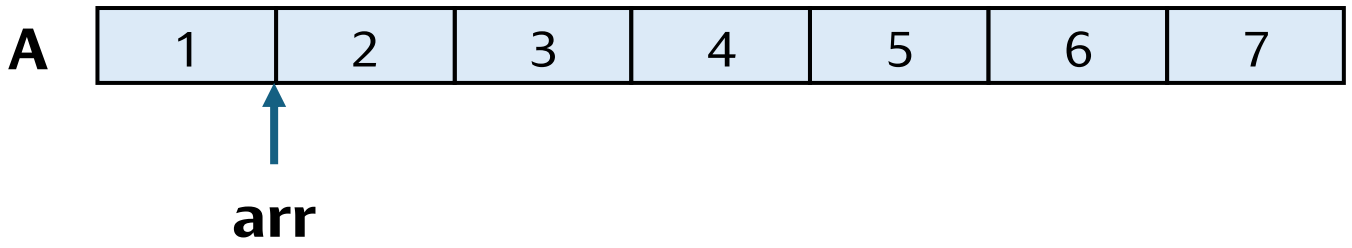
# Example: Sum an Array



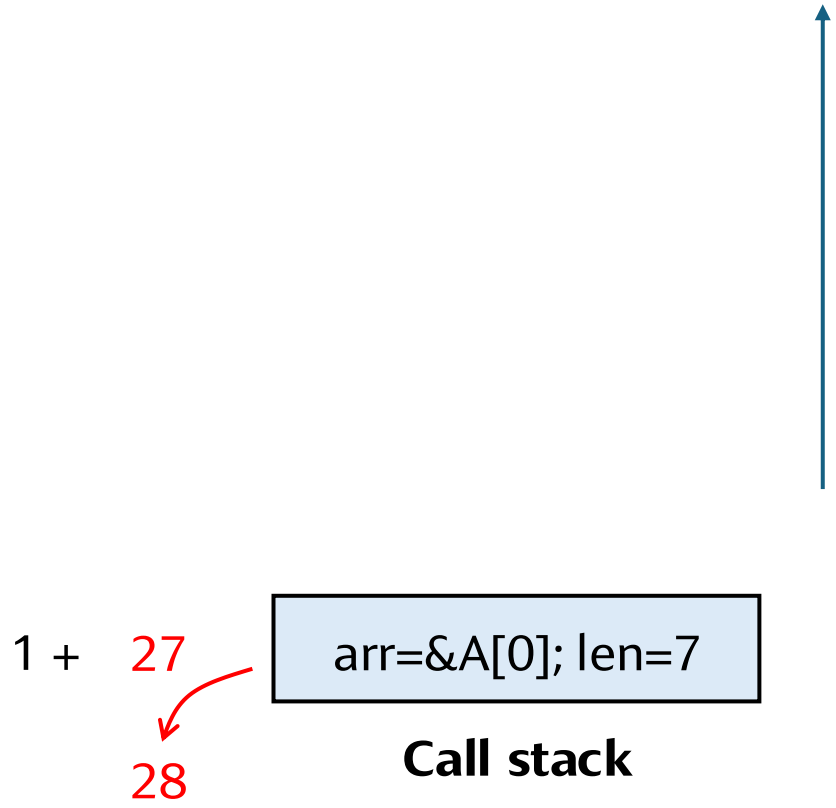

```
int sum_arr(int *arr, size_t len) {
  if (len == 0) return 0;
  return arr[0] + sum_arr(arr + 1, len - 1);
}
```



# Example: Sum an Array




```
int sum_arr(int *arr, size_t len) {
  if (len == 0) return 0;
  return arr[0] + sum_arr(arr + 1, len - 1);
}
```



# Knapsack Problem (0/1)

| **Loots** | Diamond | Gold | Hammer | Sword | … |
|---|---|---|---|---|---|
| **Value** | 1000 | 300 | 10 | 100 | … |
| **Weight** | 10 | 100 | 500 | 200 | … |

**Suppose you can only carry X units of weight, what's the maximum value you can take?**

**Recursive case:**

```
max_value(items, X) =
        MAX(
                value(items[0]) + max_value(remaining items, X – weight[items[0]]),
                max_value(remaining items, X)
        )
```

# Flooding