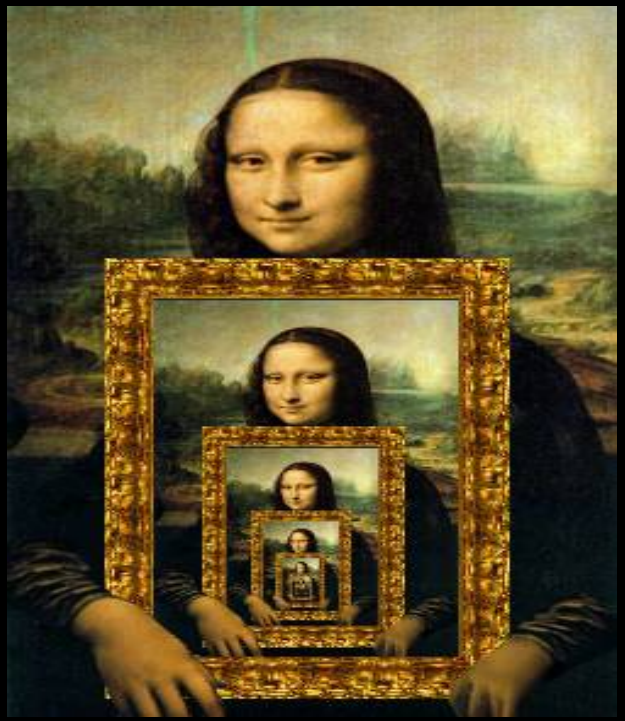
Recursion



Adding all numbers from 1 to n

$$sum(n) = n + \mathbf{(n - 1) + (n - 2) + \ldots + 1}$$

Adding all numbers from 1 to n

$$sum(n) = n + \mathbf{(n - 1) + (n - 2) + \ldots + 1}$$

$$sum(n) = n + \mathbf{sum(n - 1)}$$

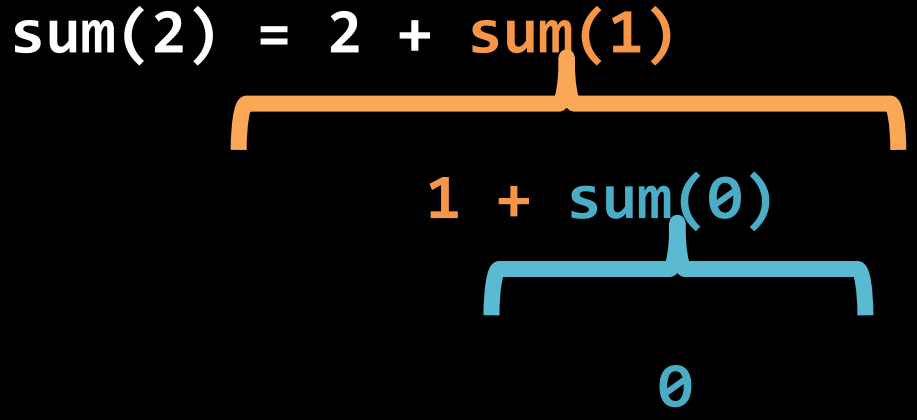
Adding all numbers from 1 to n

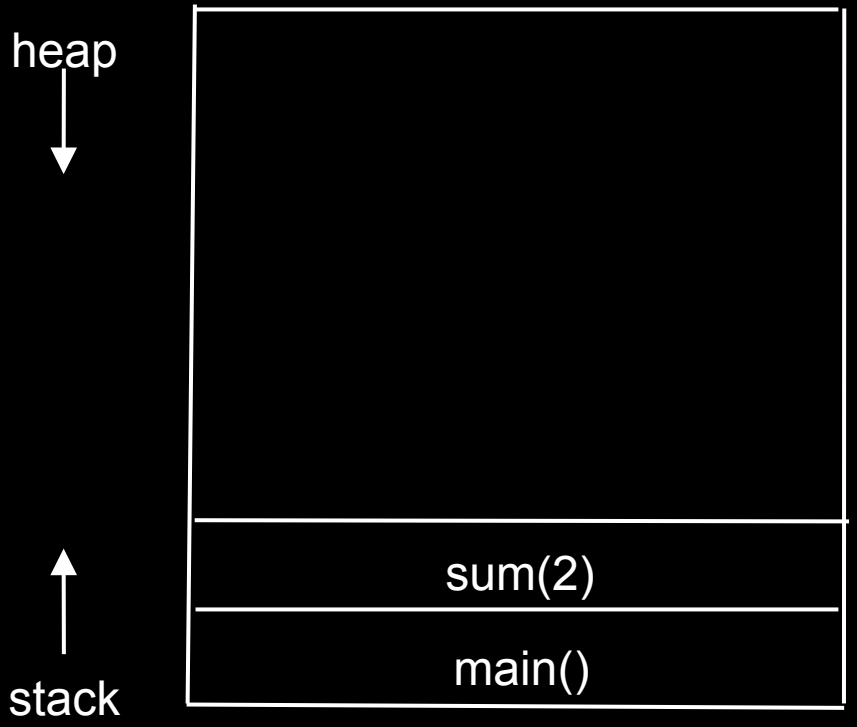
sum(n) = n + **(n - 1) + (n - 2) + ... + 1**

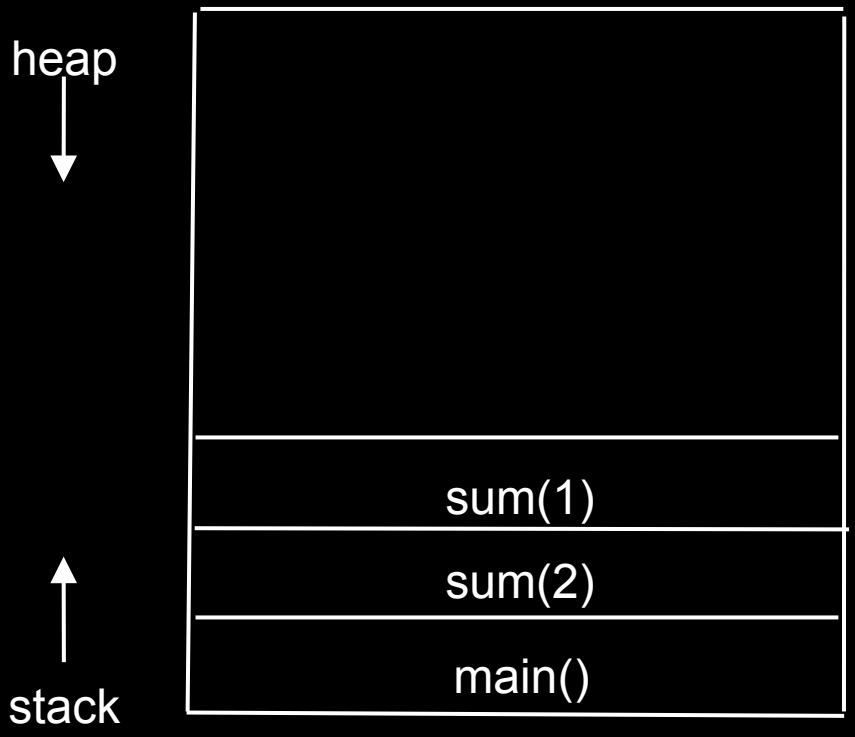
sum(n) = n + **sum(n - 1)**

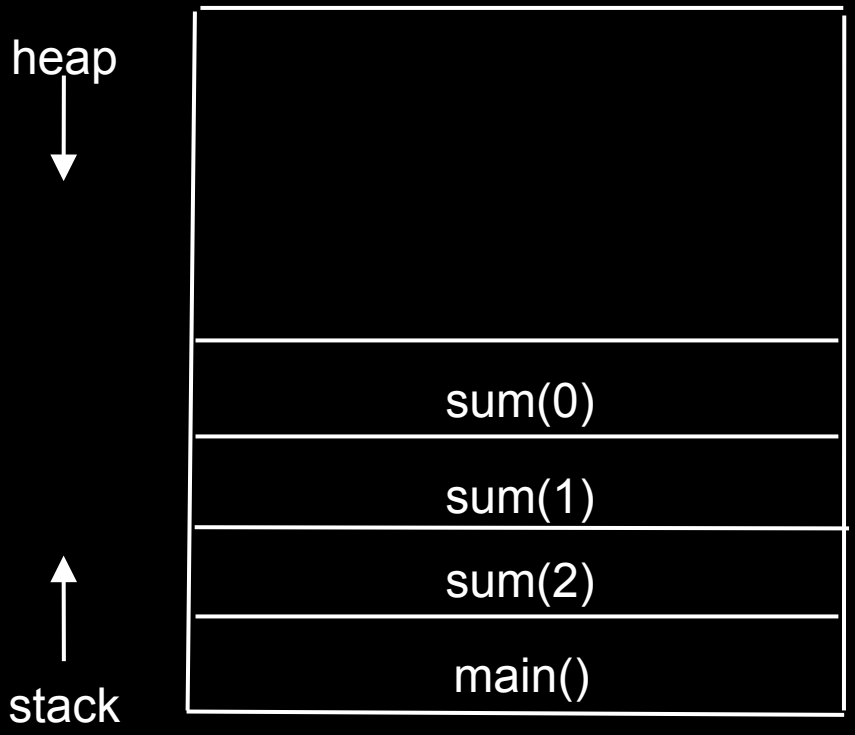
sum(0) = 0 (base case)

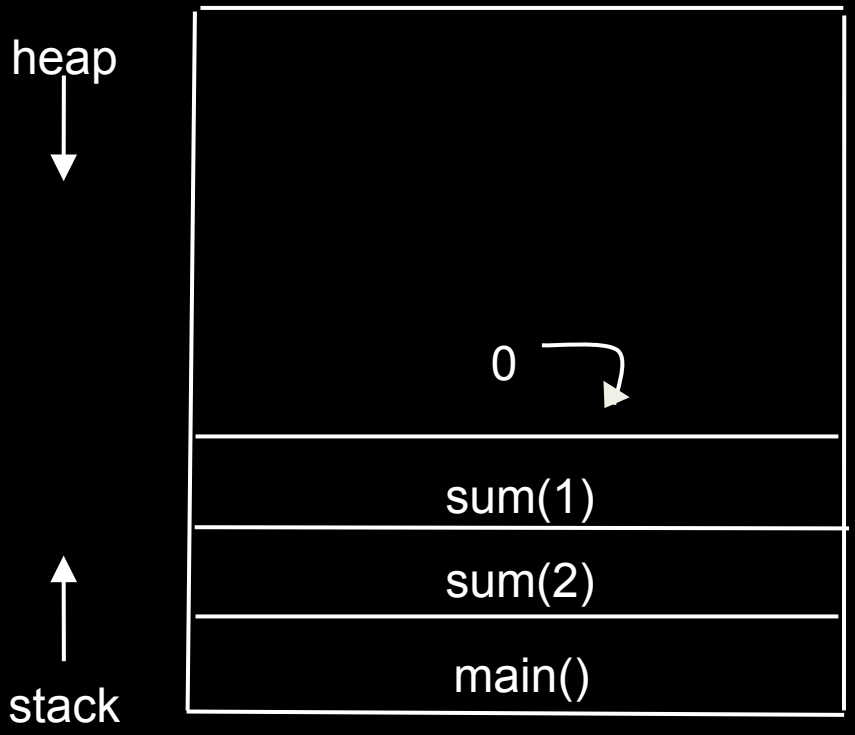
```
int sum(int n)
{
    if (n <= 0)
    {
        return 0;
    }
    else
    {
        return n + sum(n - 1);
    }
}
```

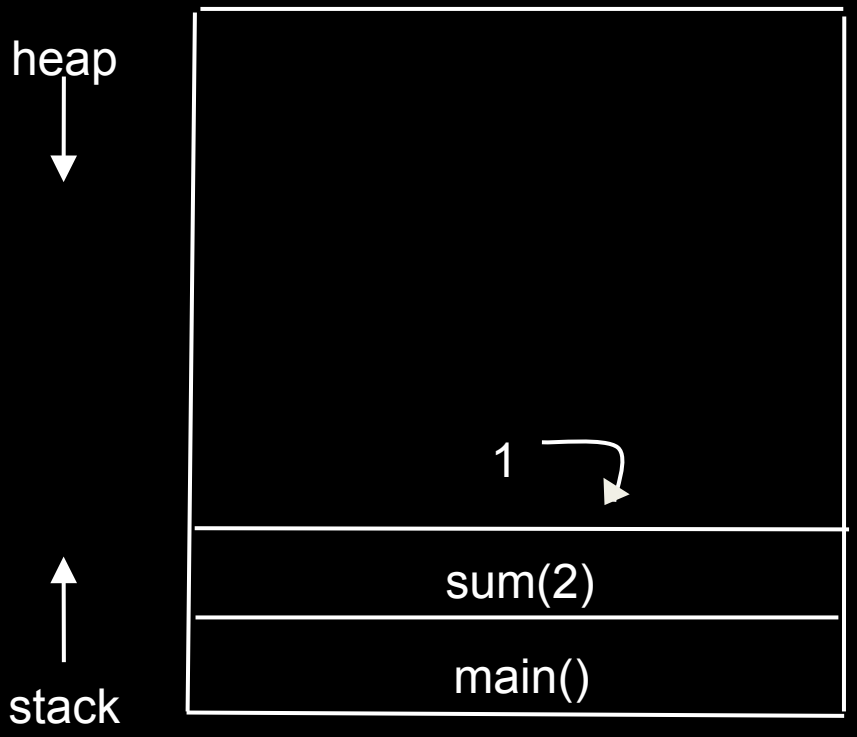

sum(2) = 2 + sum(1)
1 + sum(0)
0

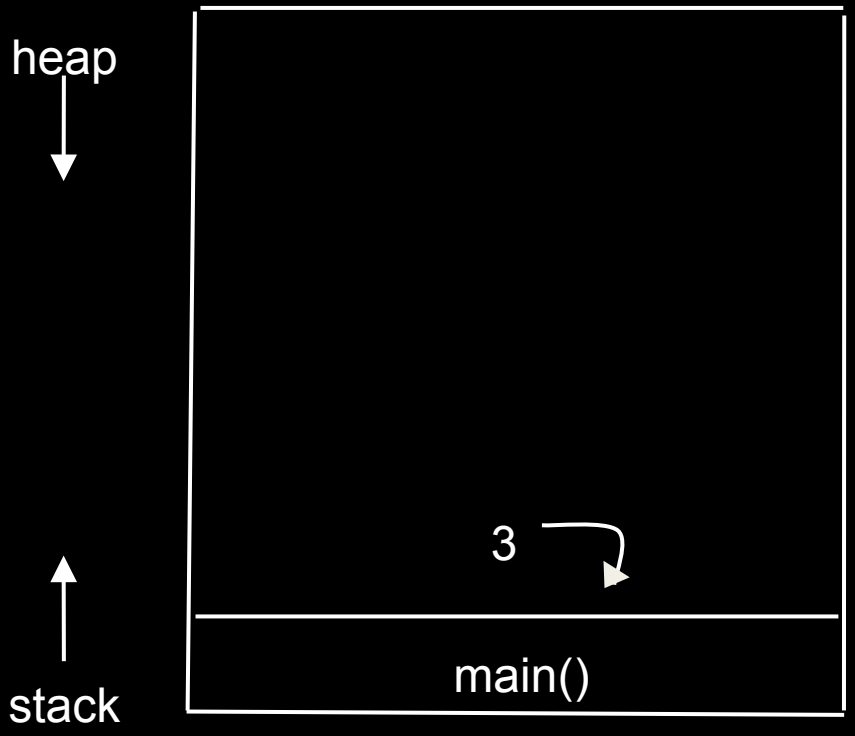

heap
sum(2)
main()
stack


heap
sum(1)
sum(2)
main()
stack


heap
sum(0)
sum(1)
sum(2)
main()
stack


heap
0
sum(1)
sum(2)
main()
stack


heap
1
sum(2)
main()
stack


heap
3
main()
stack

Sorting

- Dictionary
- Facebook
- Organizing data
- Pokémon!

Selection Sort

1. **Select** the smallest unsorted value

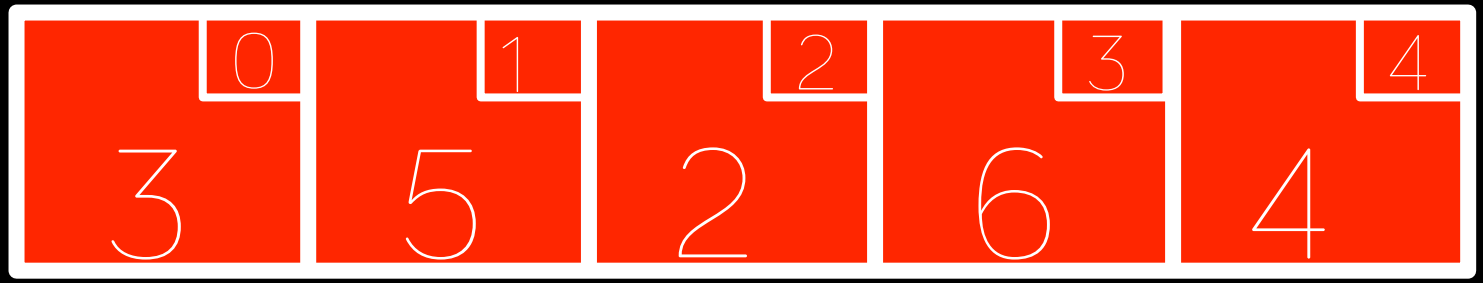
2. Move that value to the end of the “sorted” part of the list

3. Repeat from step 1 if there are still unsorted items

All values start as Unsorted

Sorted

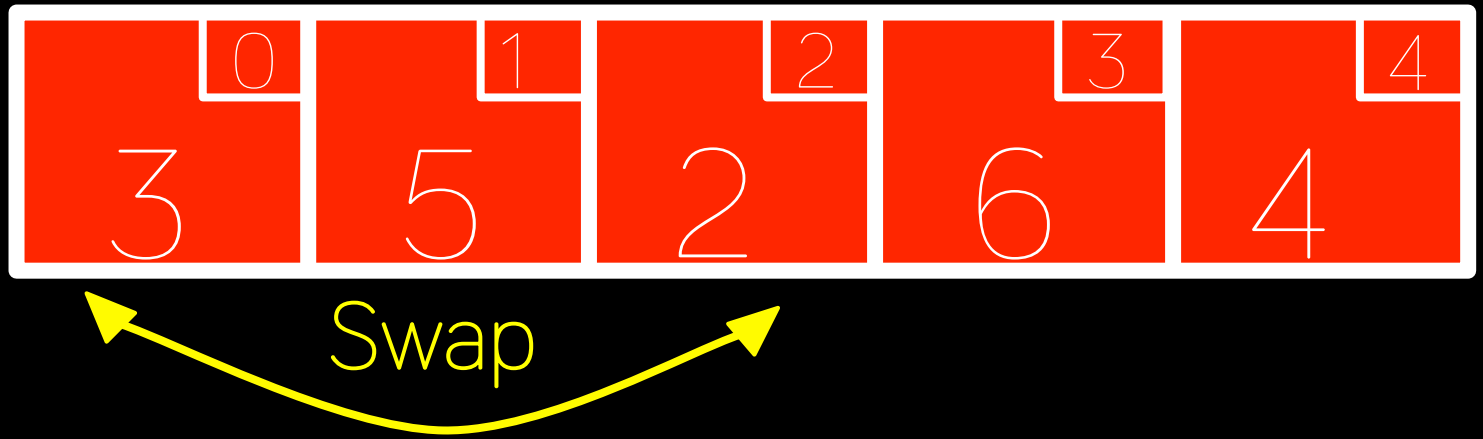
Unsorted



First pass:
2 is smallest, swap with 3

Sorted

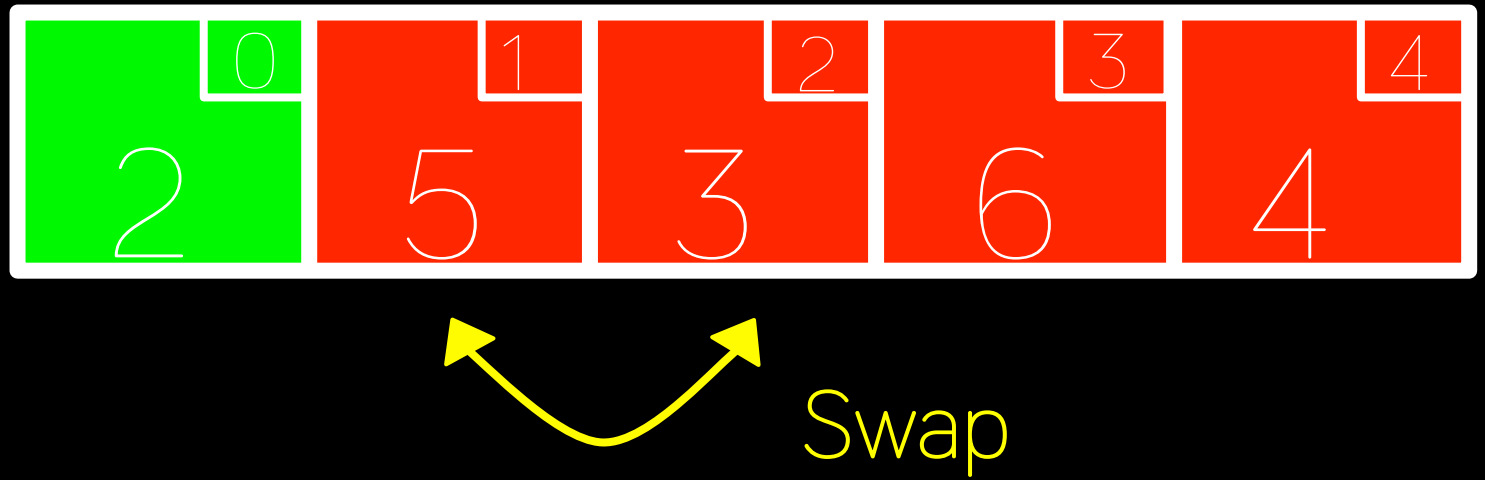
Unsorted



Second pass:
3 is smallest, swap with 5

Sorted

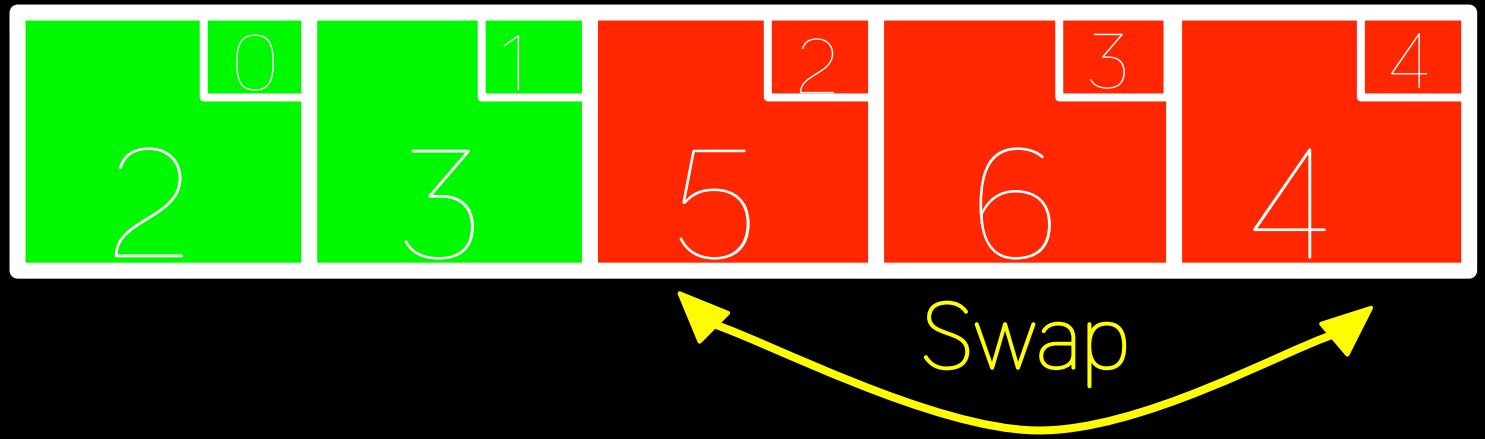
Unsorted



Third pass:
4 is smallest, swap with 5

Sorted

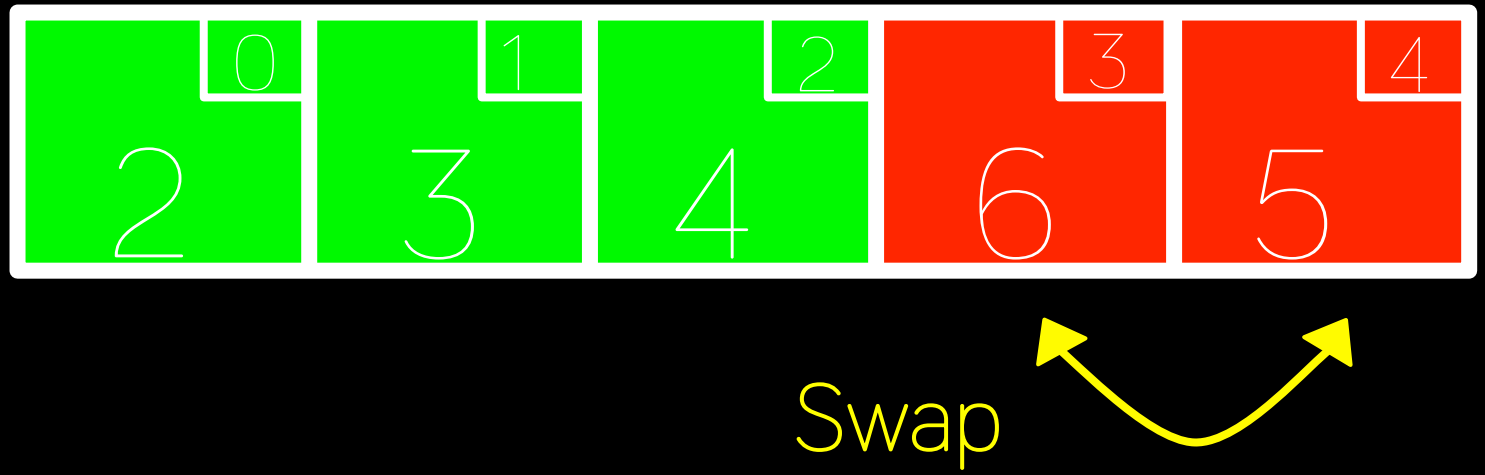
Unsorted



Fourth pass:
5 is smallest, swap with 6

Sorted

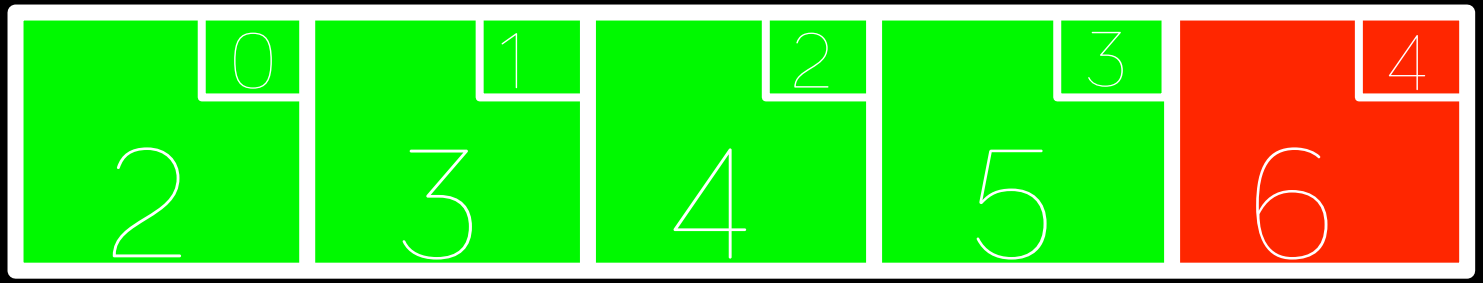
Unsorted



Fifth pass:
6 is the only value left, done!

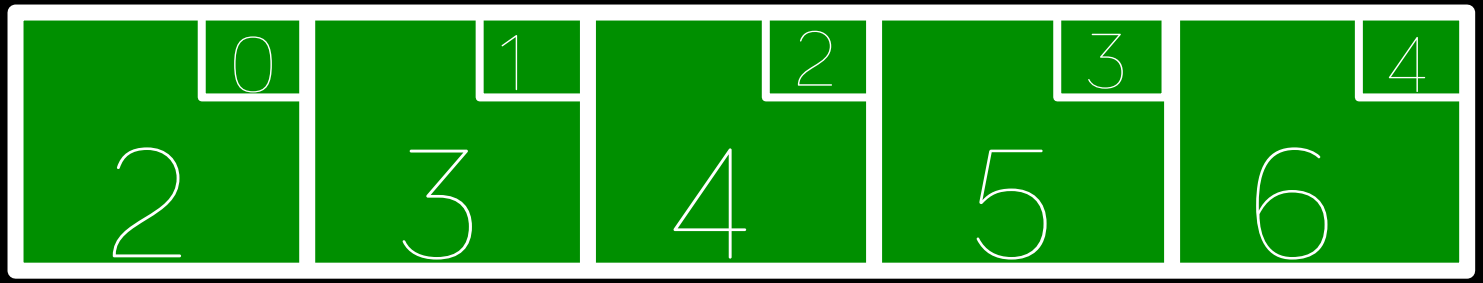
Sorted

Unsorted



Fifth pass:
6 is the only value left, done!

All Sorted



Insertion Sort

For each unsorted element n:

1. Determine where to insert n on the sorted portion of the list

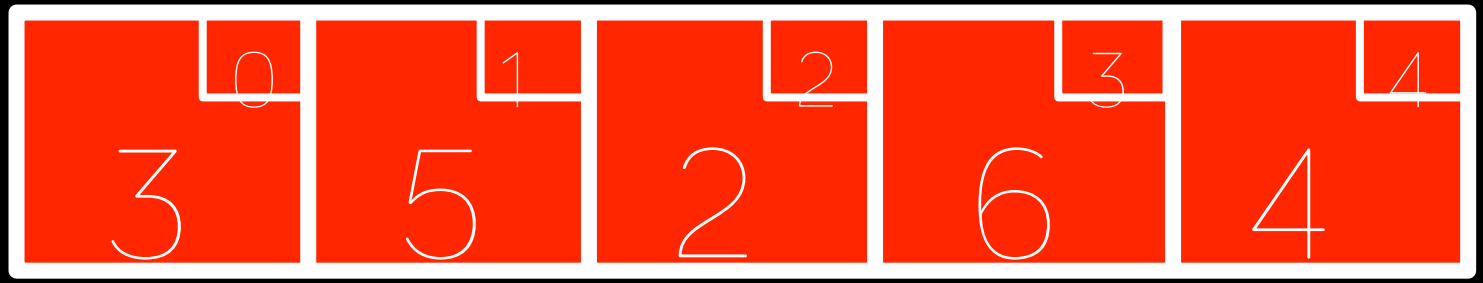
2. Shift sorted elements rightwards as necessary to make room for n

3. **Insert** n into sorted portion of the list

All values start as Unsorted

Sorted

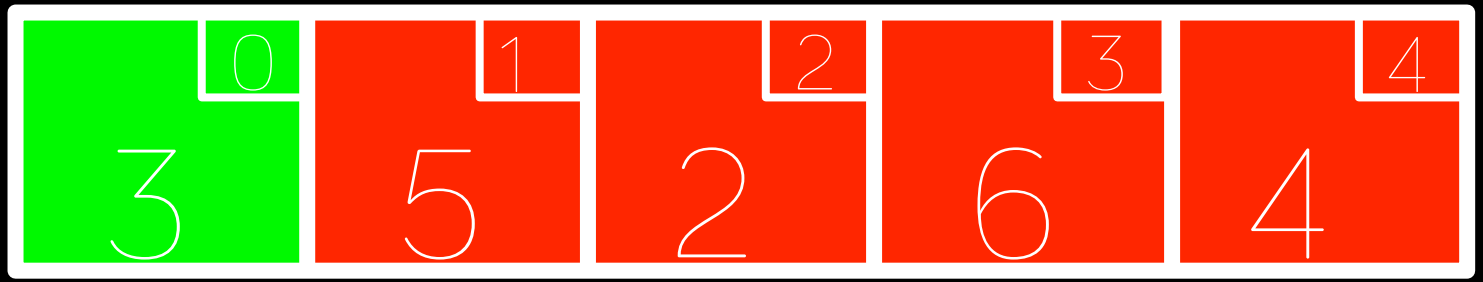
Unsorted



Add first value to Sorted

Sorted

Unsorted

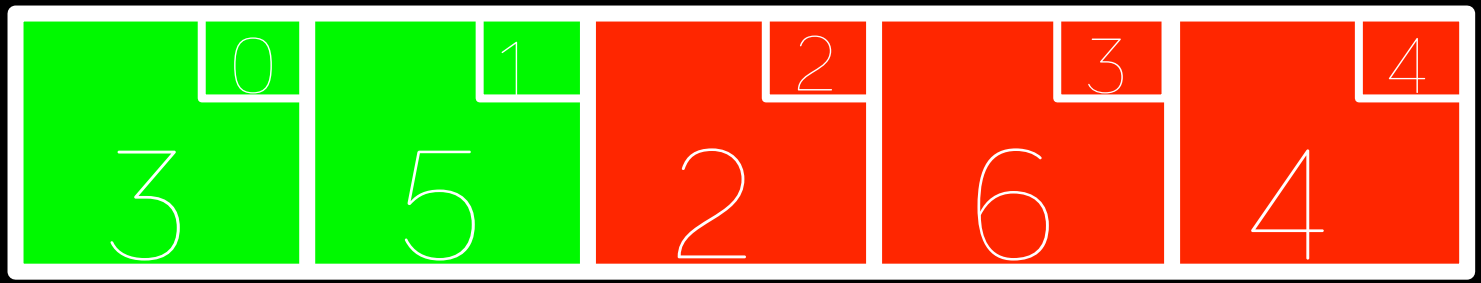


5 > 3

insert 5 to right of 3

Sorted

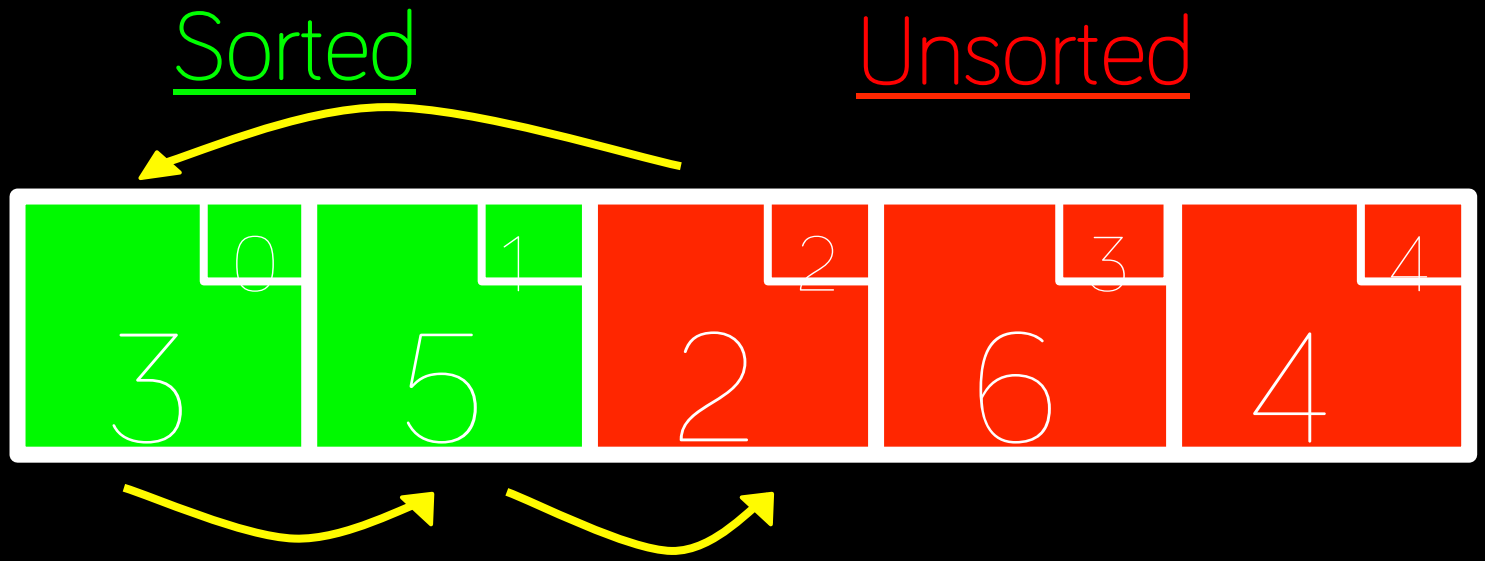
Unsorted



$2 < 5$ and $2 < 3$

shift 3 and 5

insert 2 to left of 3

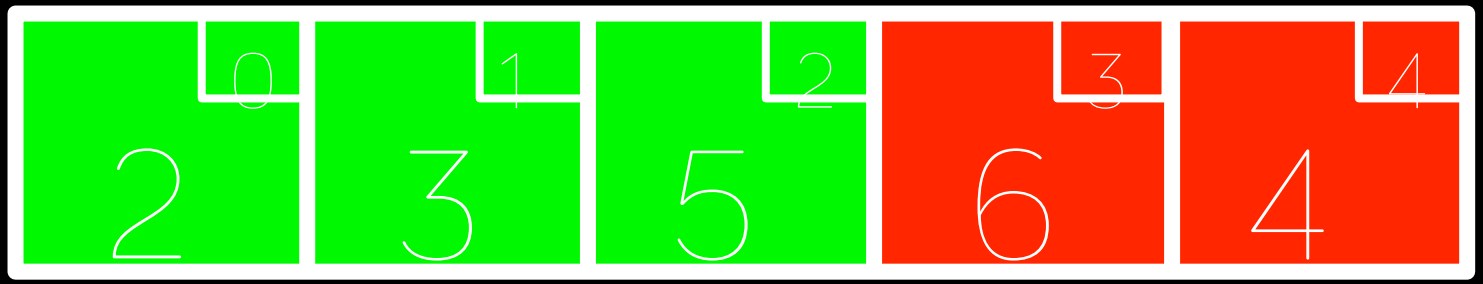


6 > 5

insert 6 to right of 5

Sorted

Unsorted



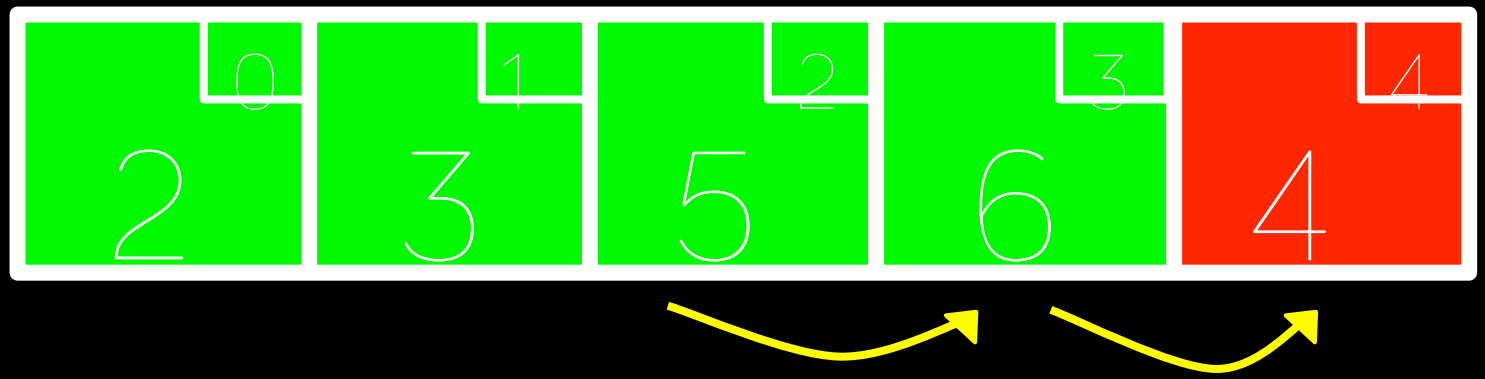
4 < 6, 4 < 5, and 4 > 3

shift 5 and 6

insert 4 to right of 3

Sorted

Unsorted

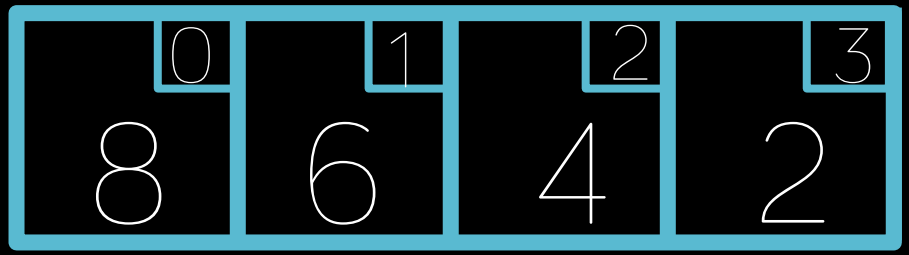


Bubble Sort

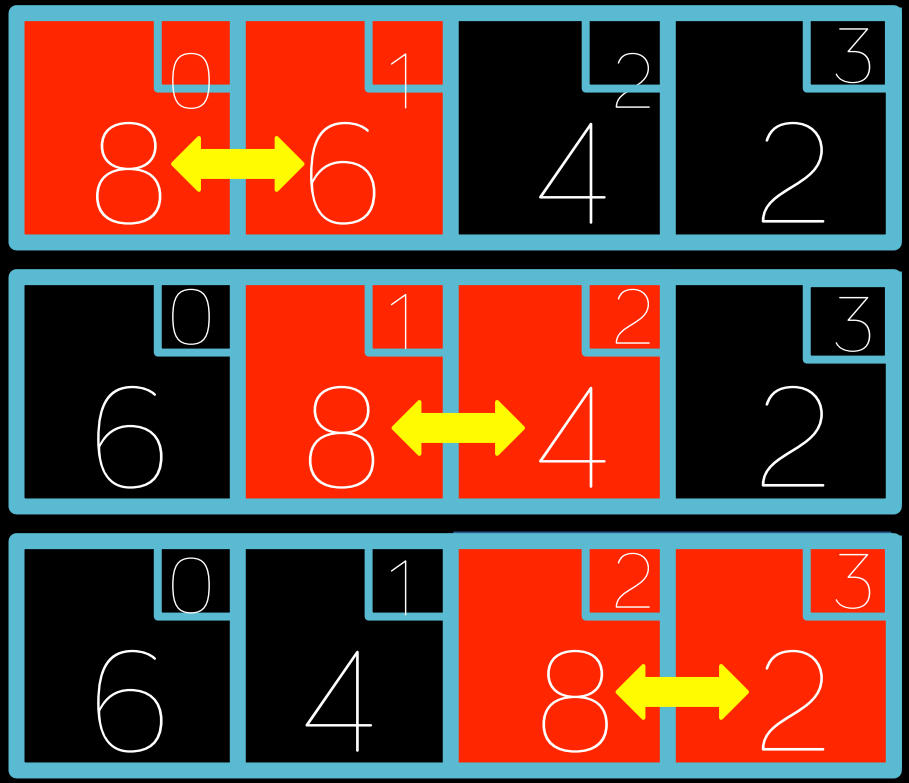
1. Step through entire list, swapping adjacent values if not in order

2. Repeat from step 1 if any swaps have been made

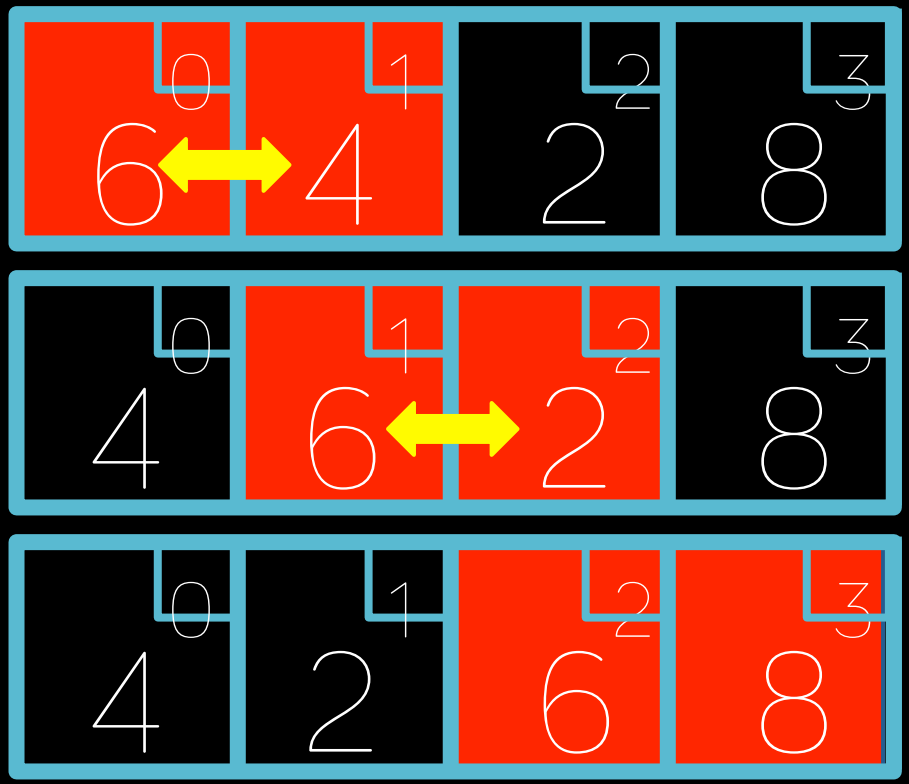
At each step, the largest value **bubbles** to the end of the list


0
1
2
3
8
6
4
2

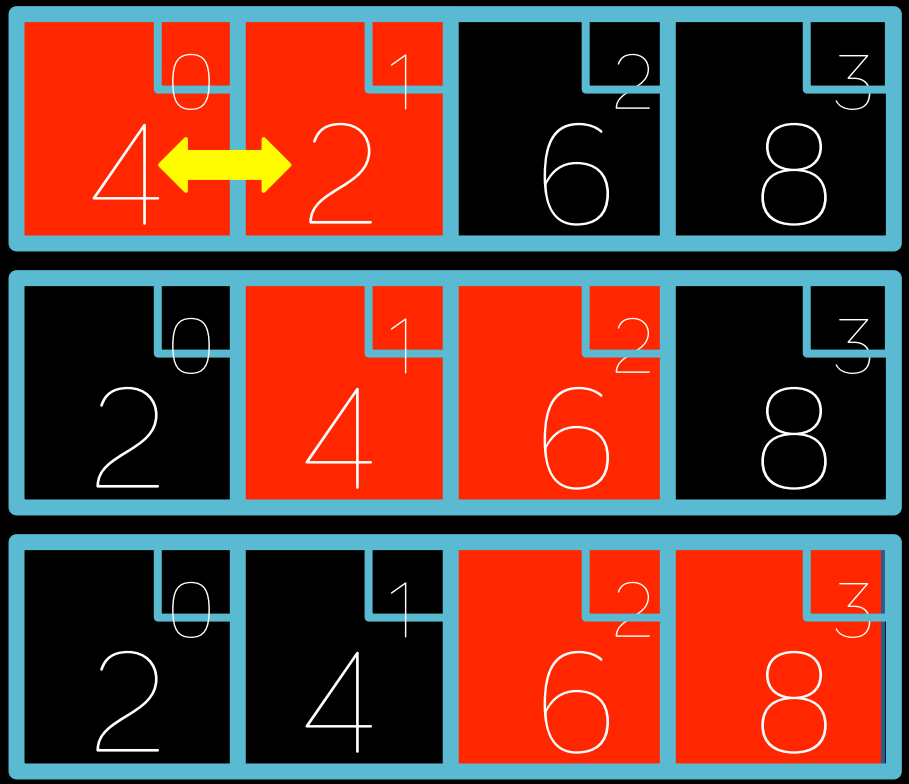
First pass: 3 swaps



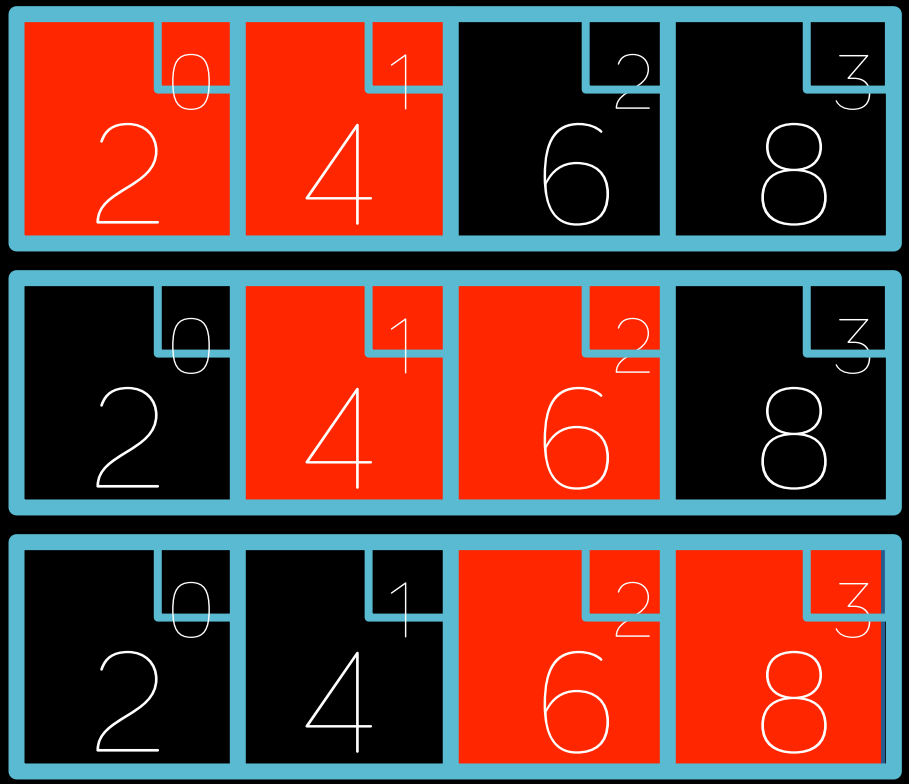
Second pass: 2 swaps



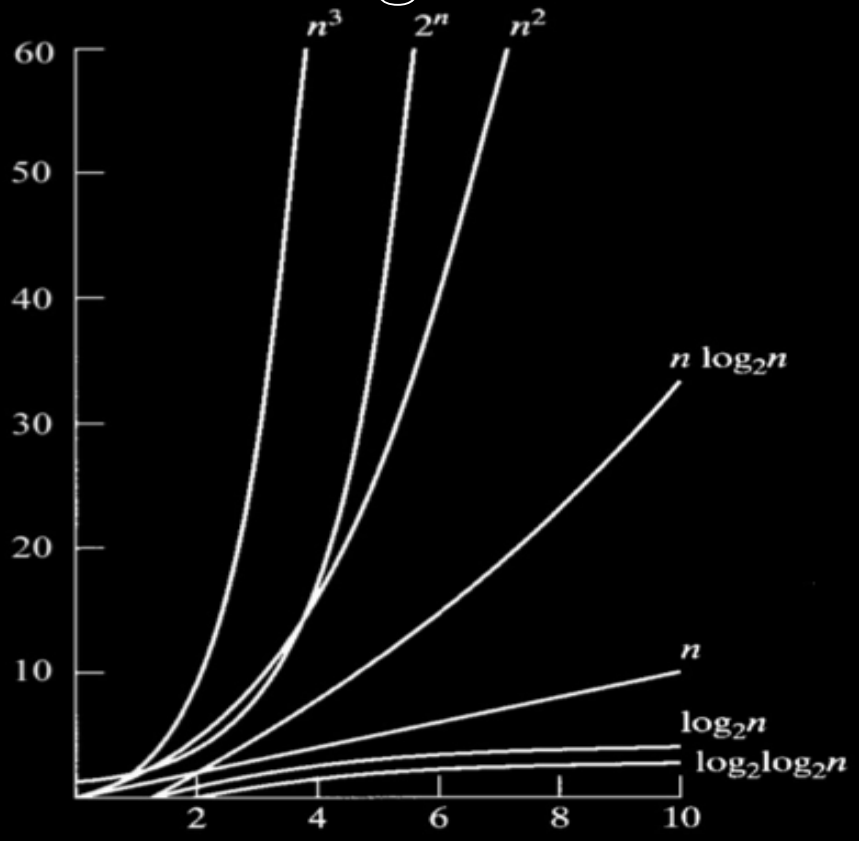
Third pass: 1 swap



Fourth pass: 0 swaps



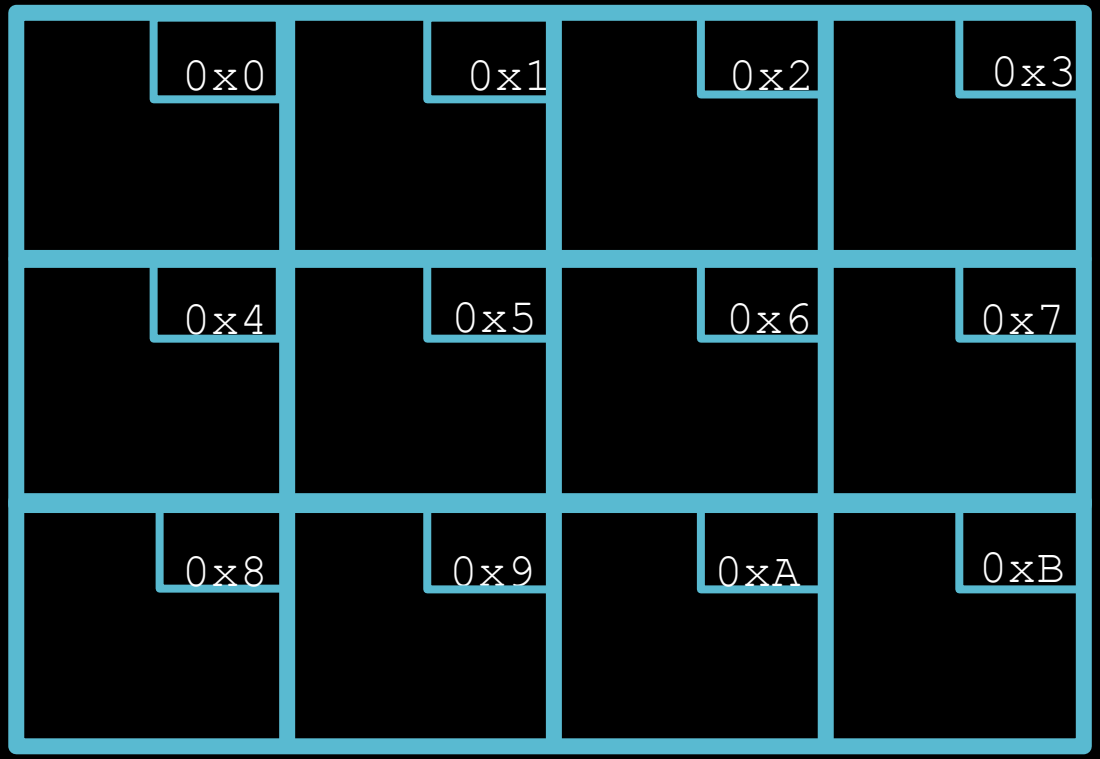
Running Time



	O	Ω	Θ
Bubble sort	n^2	n	
Selection sort	n^2	n^2	n^2
Insertion sort	n^2	n	
Merge sort	$n\log n$	$n\log n$	$n\log n$

Pointers

Memory



Creating Pointers

Declaring pointers:

<type>* <variable name>

Examples:

```
int* x;
char* y;
float* z;
```

Size: 4 bytes for 32-bit machine

Referencing and Dereferencing

Referencing:
&<variable name>
& is the same as saying “address of”

Dereferencing:
*<pointer name>
***** is the same as saying “content of”

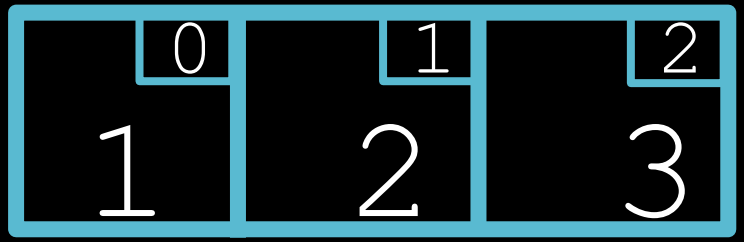
Let's see this in memory

```
int a = 3;
int b = 4;
int c = 5;
int* pa = &a;
int* pb = &b;
int* pc = &c;
```

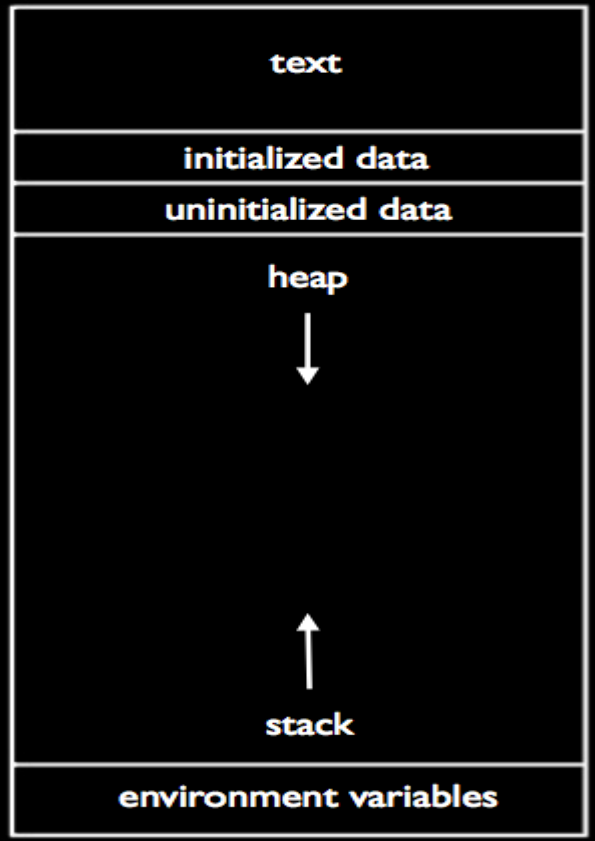
Pointers and Arrays

```
int array[3];

*array = 1;
*(array + 1) = 2;
*(array + 2) = 3;
```



Dynamic Memory Allocation



m(emory)alloc(ation)

- Allocates memory on the heap
- Returns pointer to allocated memory
- Returns **`NULL`** if can't allocate memory

malloc

```
void* malloc(size in bytes);
```

example:

```
int* ptr;
ptr = malloc(sizeof(int) * 10);
```

Check for NULL!

```
int* ptr = malloc(sizeof(int) * 10);

if (ptr == NULL)
{
      printf("Error -- out of memory.\n");
      return 1;
}
```

Don’t forget to free!

```
free(ptr);
```

```c
#include <cs50.h>
#include <stdio.h>

int main(void)
{
    int* ptr = malloc(sizeof(int));
    if (ptr == NULL)
    {
        printf("Error -- out of memory.\n");
        return 1;
    }

    *ptr = GetInt();
    printf("You entered %i.\n", *ptr);

    free(ptr);

    return 0;
}
```