Welcome back!

Announcements

- Grading
  - Be sure to use check50 and style50
  - Uploading to the correct folder
- Quiz 0: October 15th!
- Problem Set #3: Breakout!
  - Start early -> make sure the appliance is configured correctly!

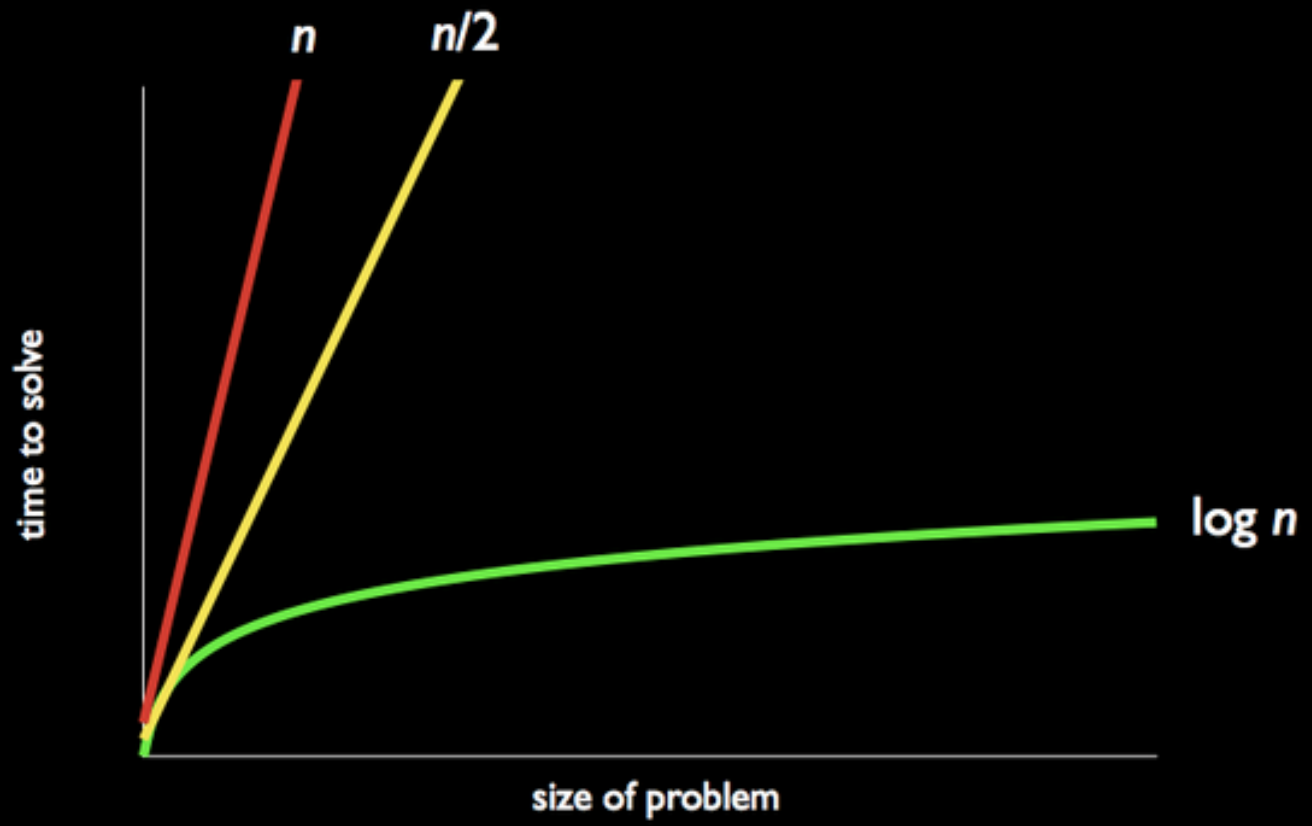
Agenda

- GDB (or the best friend you just heard about)
- Binary Search
- Sorts!
  - Bubble
  - Selection
  - Insertion
  - Merge

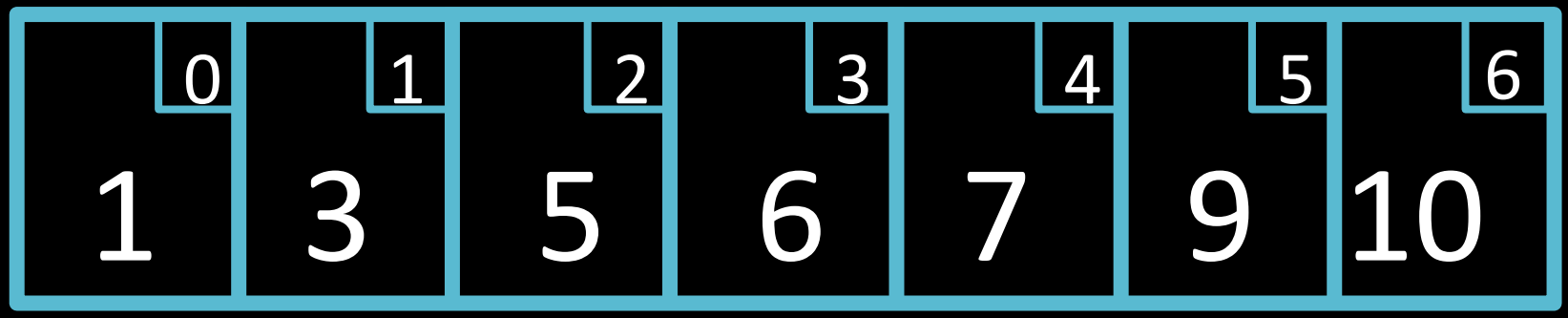
GDB

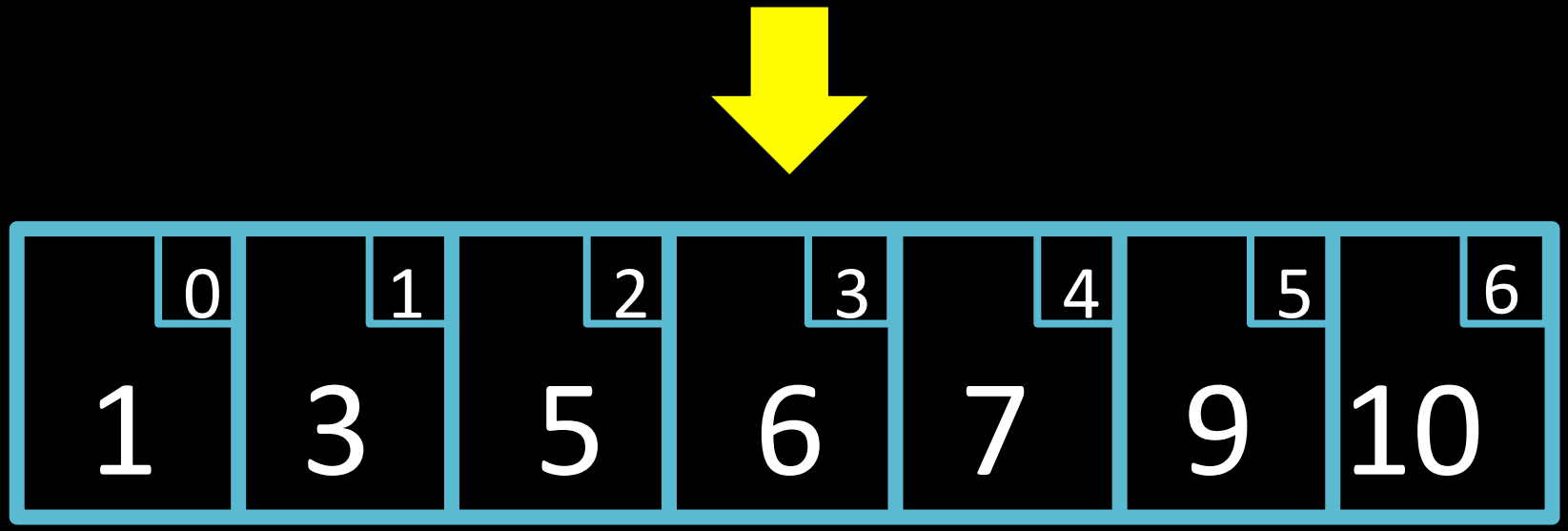
- break [line/function]
  - Sets a breakpoint
- next
  - Skip to next line; over functions
- step
  - Step into a function
- list
  - Print surrounding code
- print [variable]
  - Print a specified variable
- info locals
  - Prints out local variables
- display [variable]
  - Displays the variable for the duration of debugging

Binary Search


n
n/2
log n
time to solve
size of problem

Does the array contain 7?

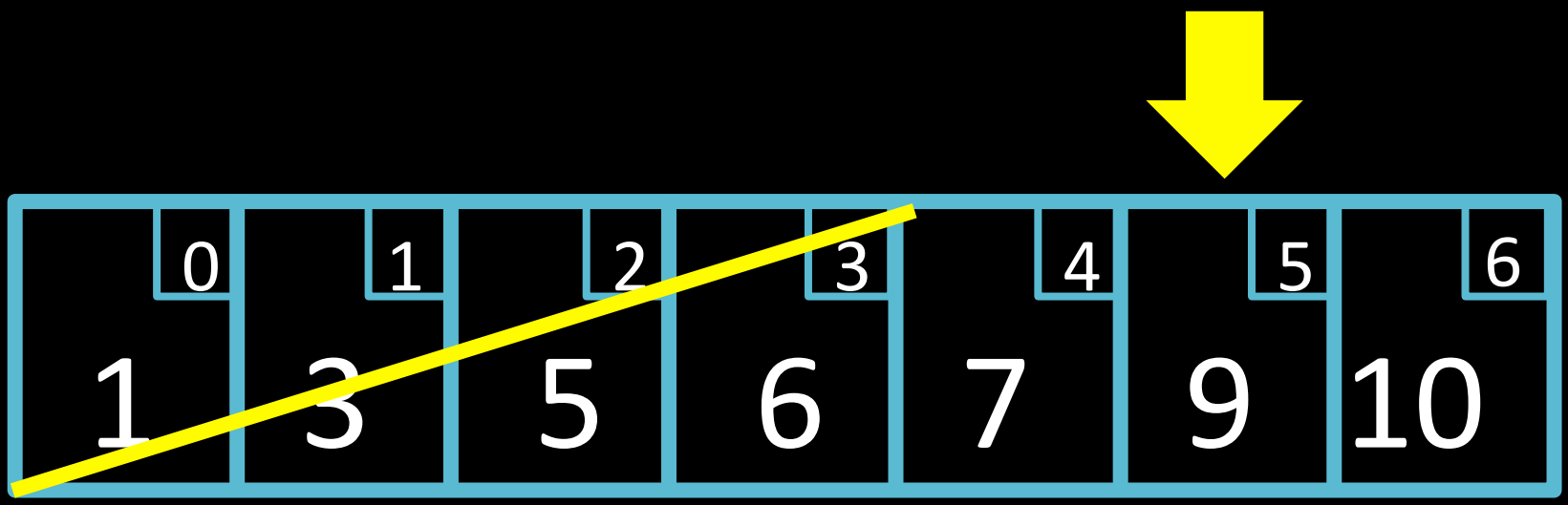




Is array[3] == 7?

Is array[3] < 7?

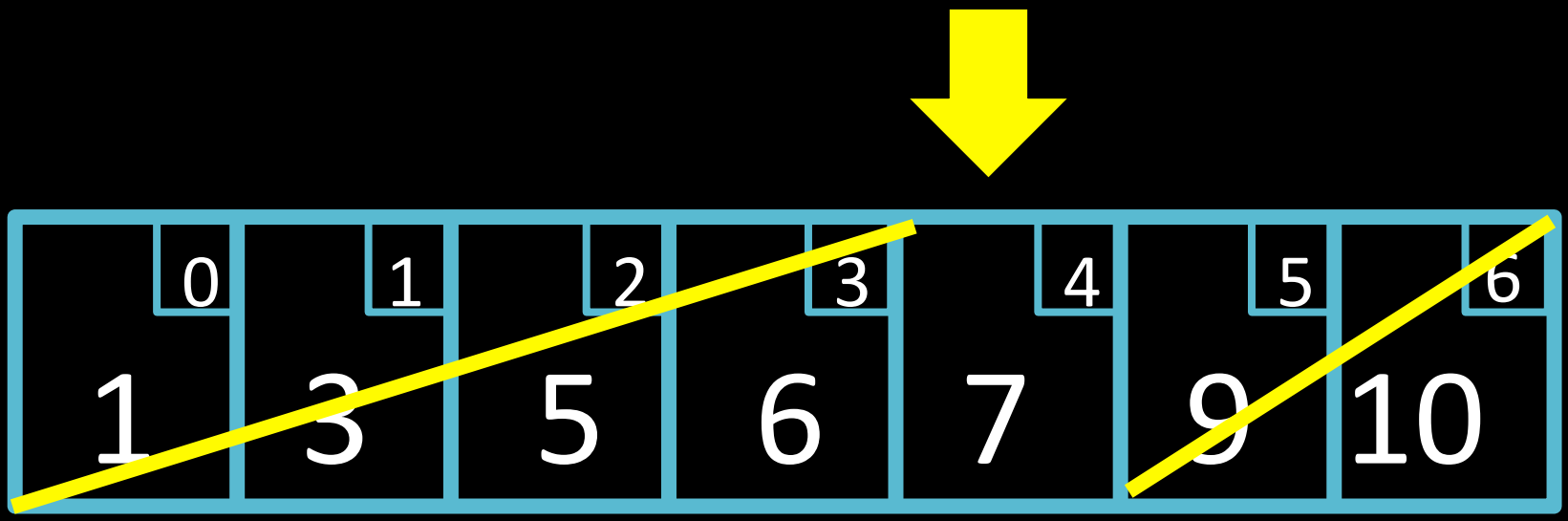
Is array[3] > 7?



Is array[5] == 7?

Is array[5] < 7?

Is array[5] > 7?



Is array[4] == 7?

Is array[4] < 7?

Is array[4] > 7?

TIME FOR SOME CODE!

Binary Sort Recap

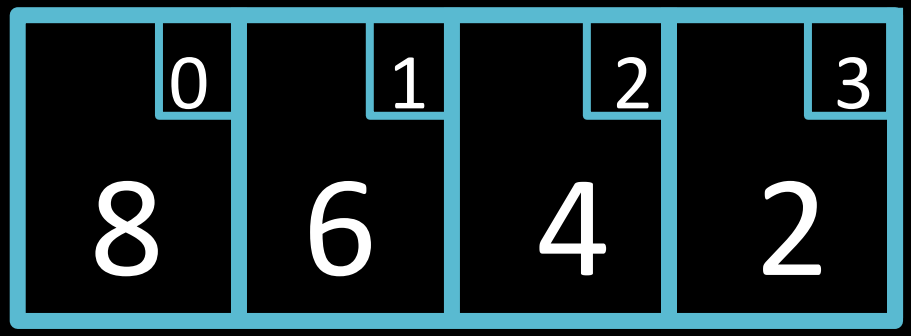
- Best case: $\Omega(1)$
- Worst case: $O(\log n)$
- Assumptions:
  - SORTED list
  - Random access

Bubble Sort

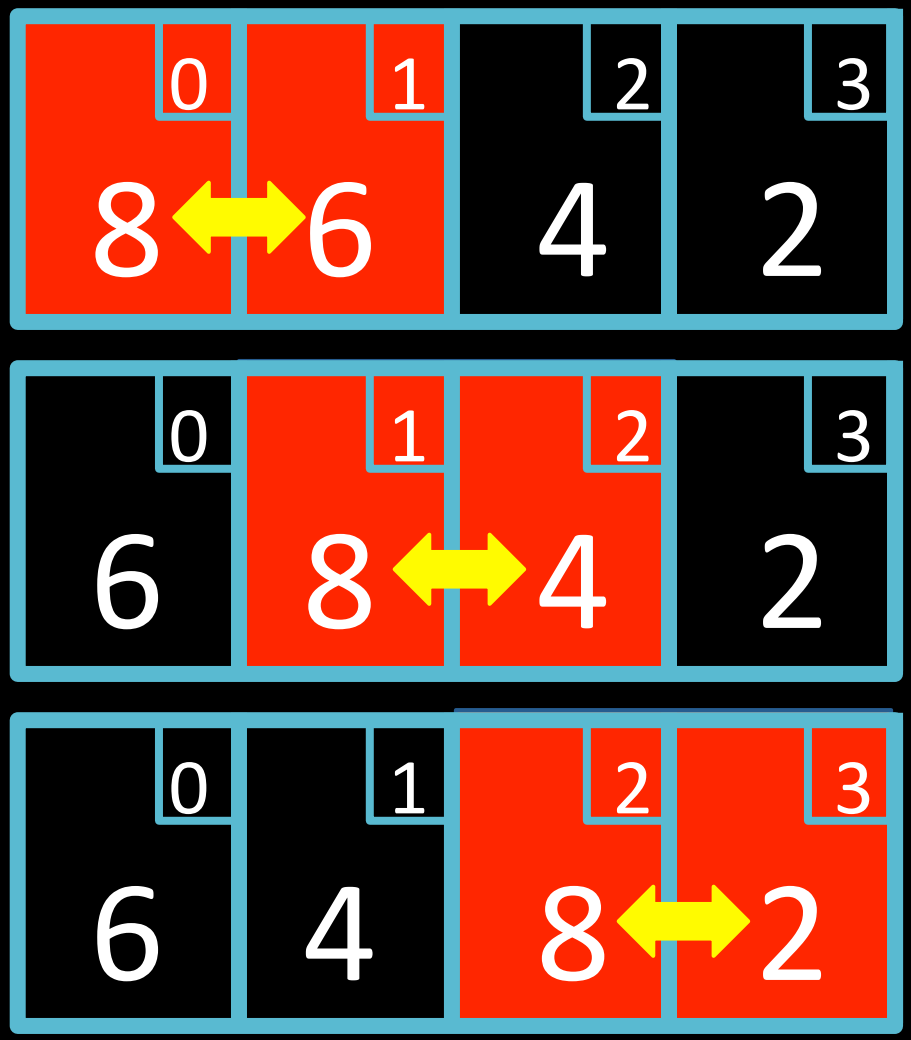
Algorithm

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

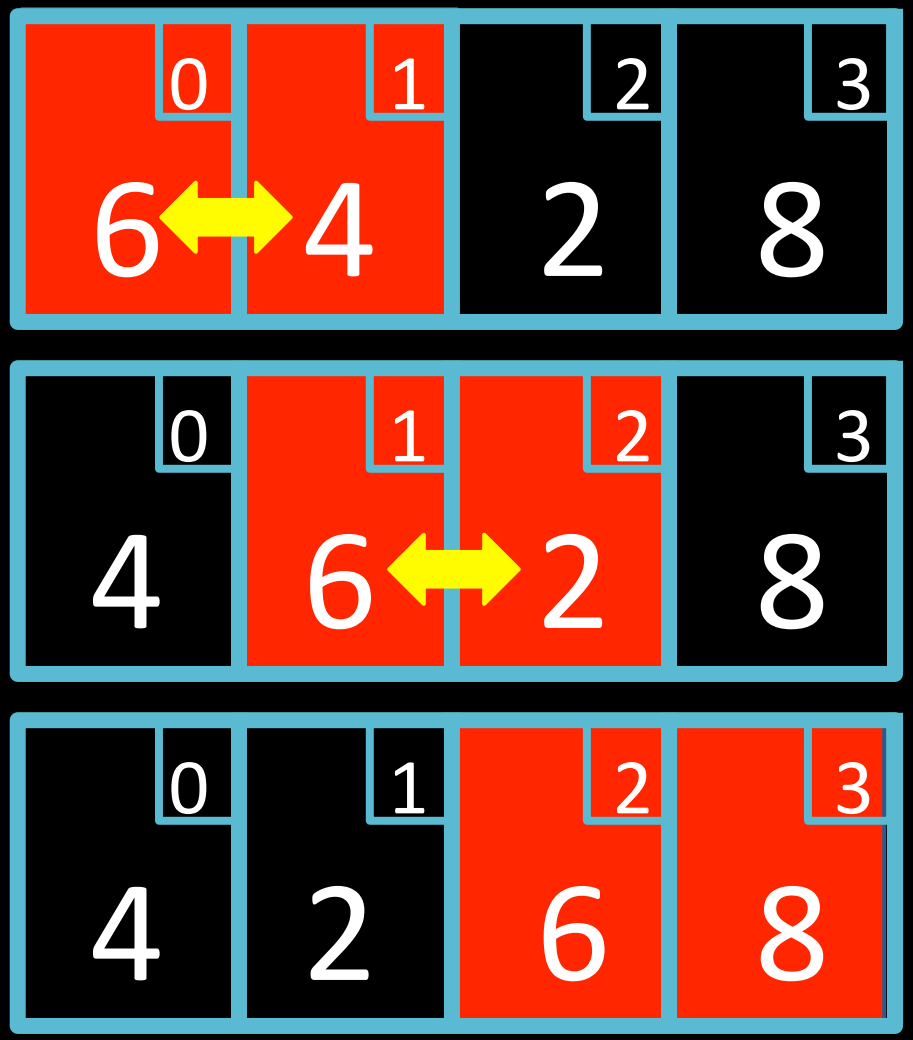
2. Repeat from step 1 if any swaps have been made


0
1
2
3
8
6
4
2

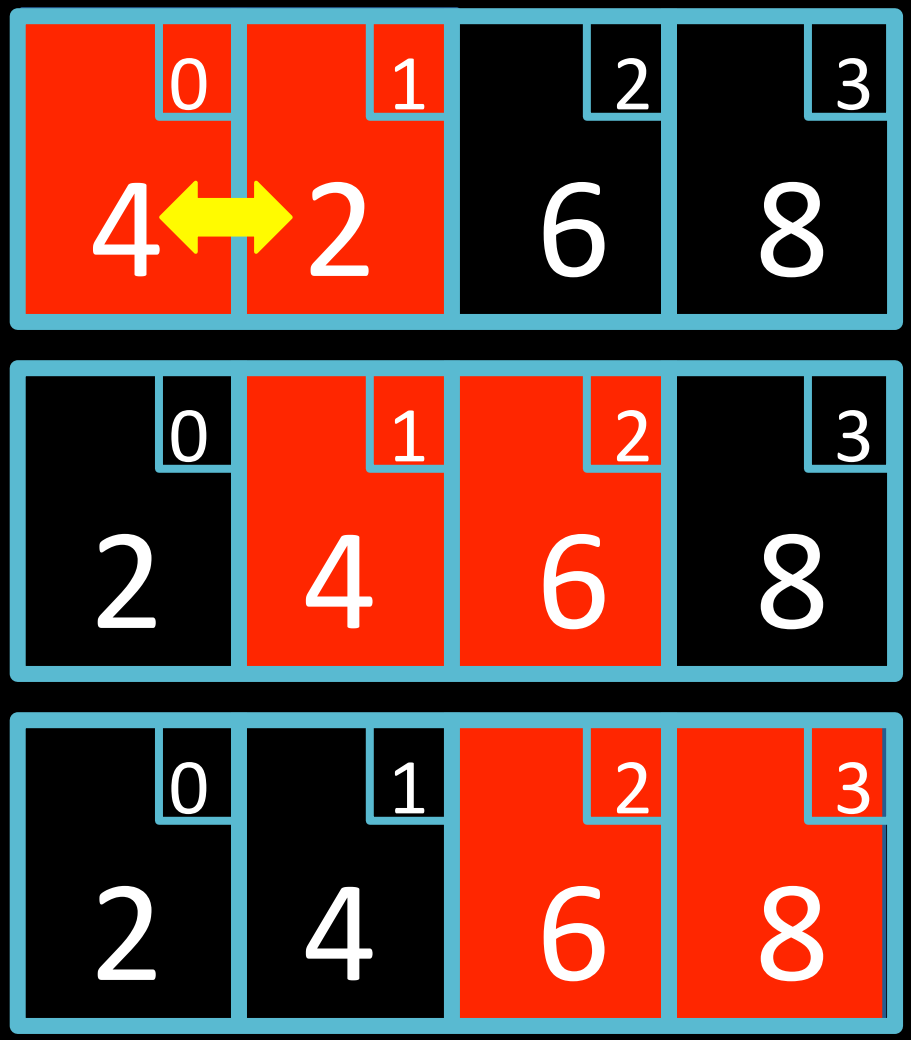
First pass: 3 swaps



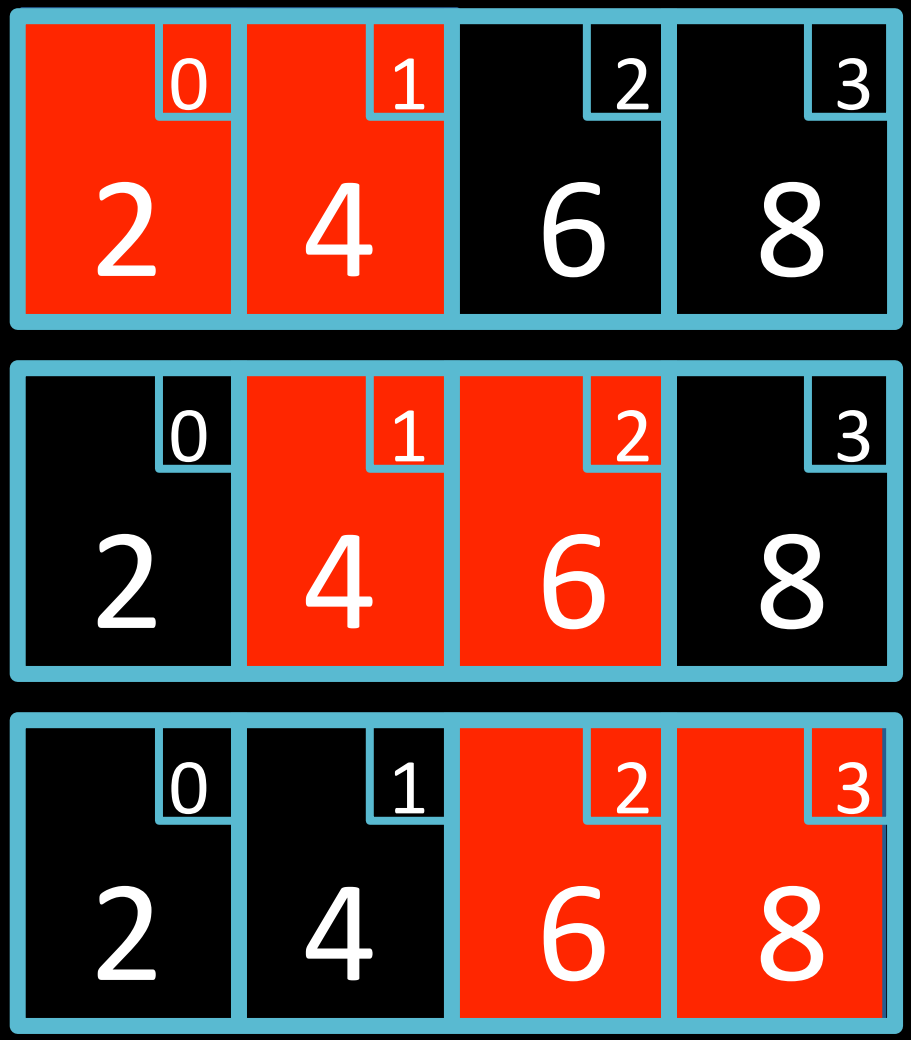
Second pass: 2 swaps



Third pass: 1 swap



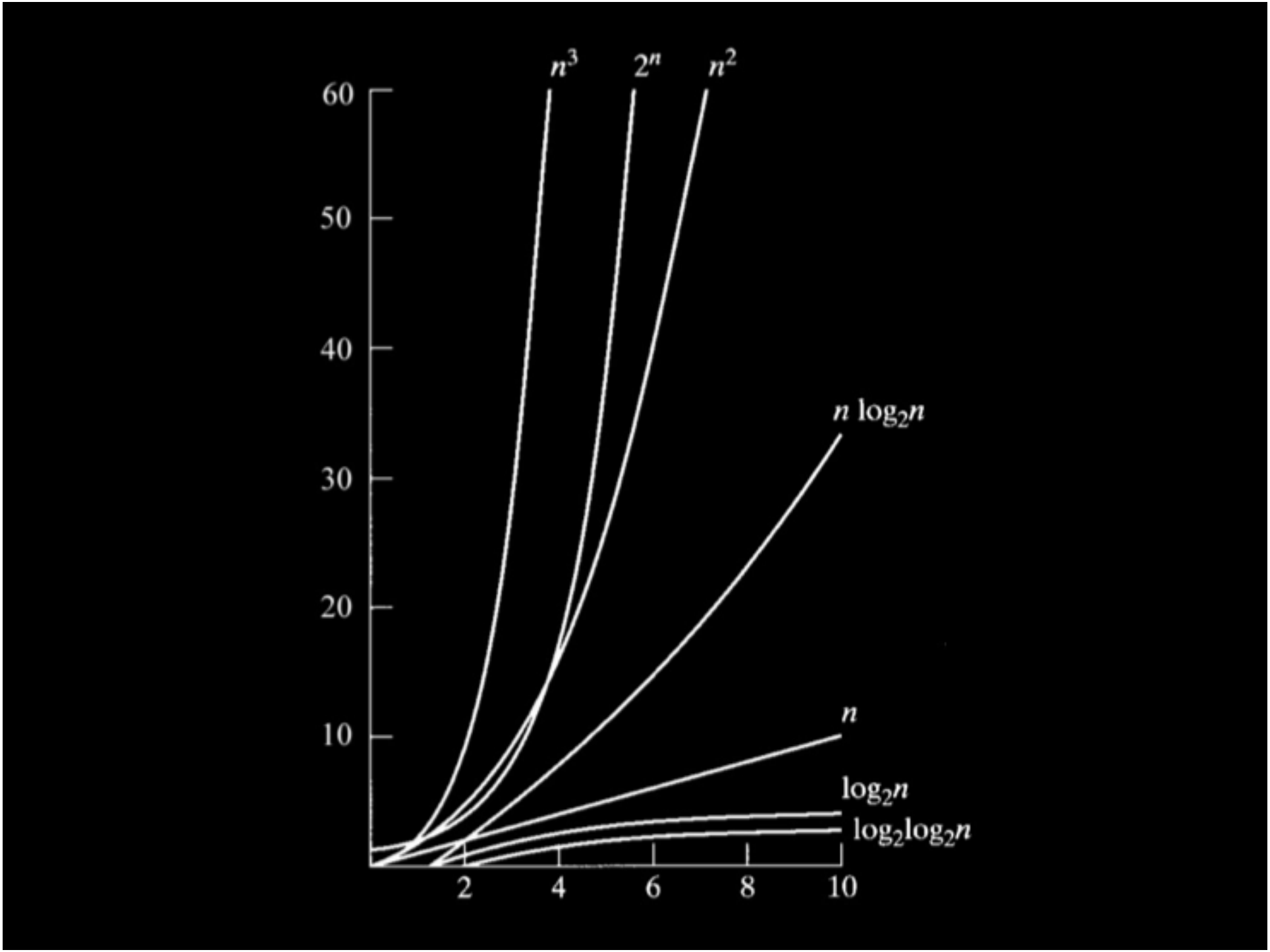
Fourth pass: 0 swaps



```
initialize counter
do
{
        set counter to 0

        iterate through entire array
                if array[n] > array[n+1]
                        swap them
                        increment counter
}
while (counter > 0)
```

What's the worst case runtime of bubble sort?

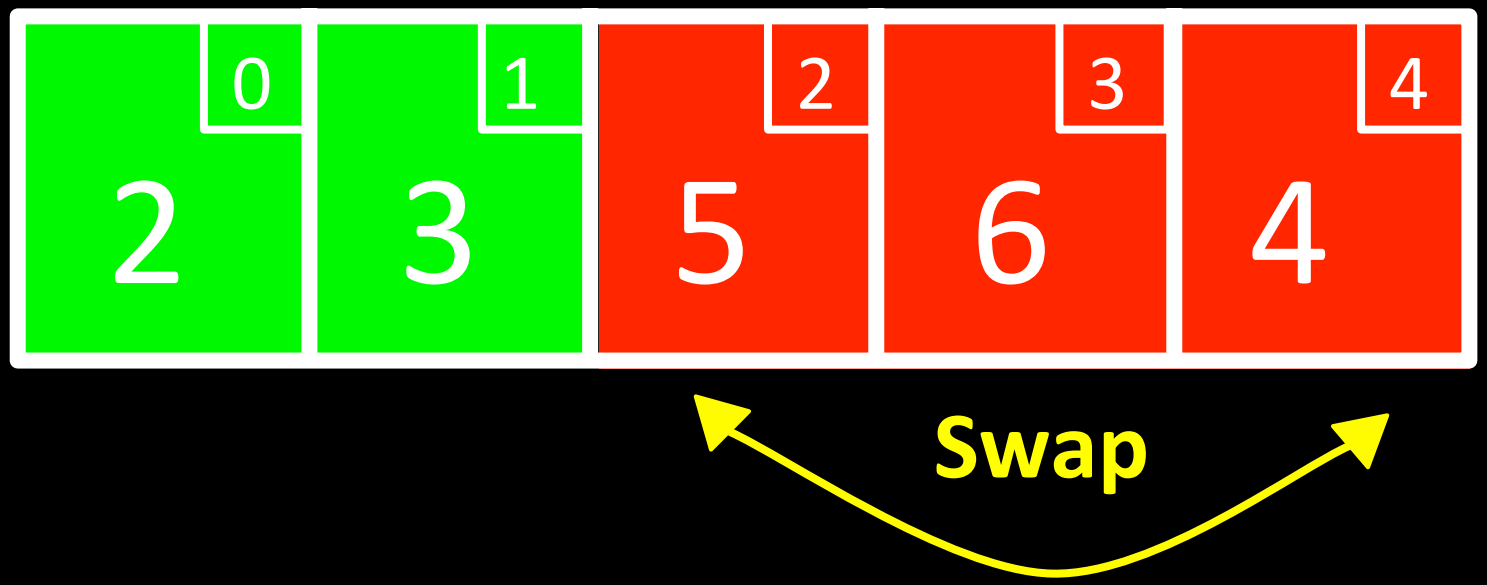
What's the best case runtime of bubble sort?


n^3
2^n
n^2
60
50
40
30
20
10
$n \log_2 n$
n
$\log_2 n$
$\log_2 \log_2 n$
2
4
6
8
10

	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort
O	n^2	n^2	n^2	$nlogn$
Ω	n	n^2	n	$nlogn$
Θ		n^2		$nlogn$

Selection Sort

Sorted **Unsorted**



Algorithm

1. Find the smallest unsorted value

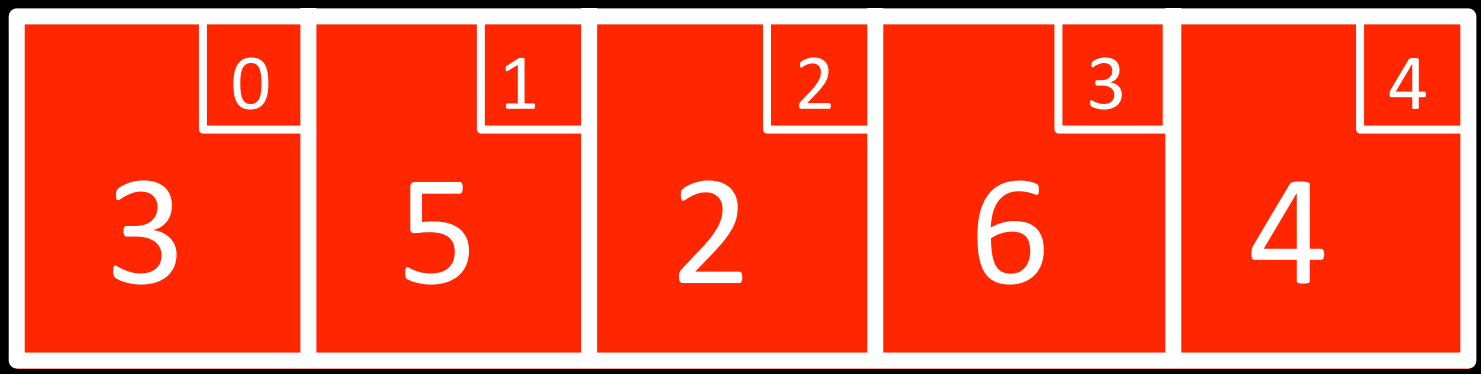
2. Swap that value with the first unsorted value

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

All values start as Unsorted

Sorted

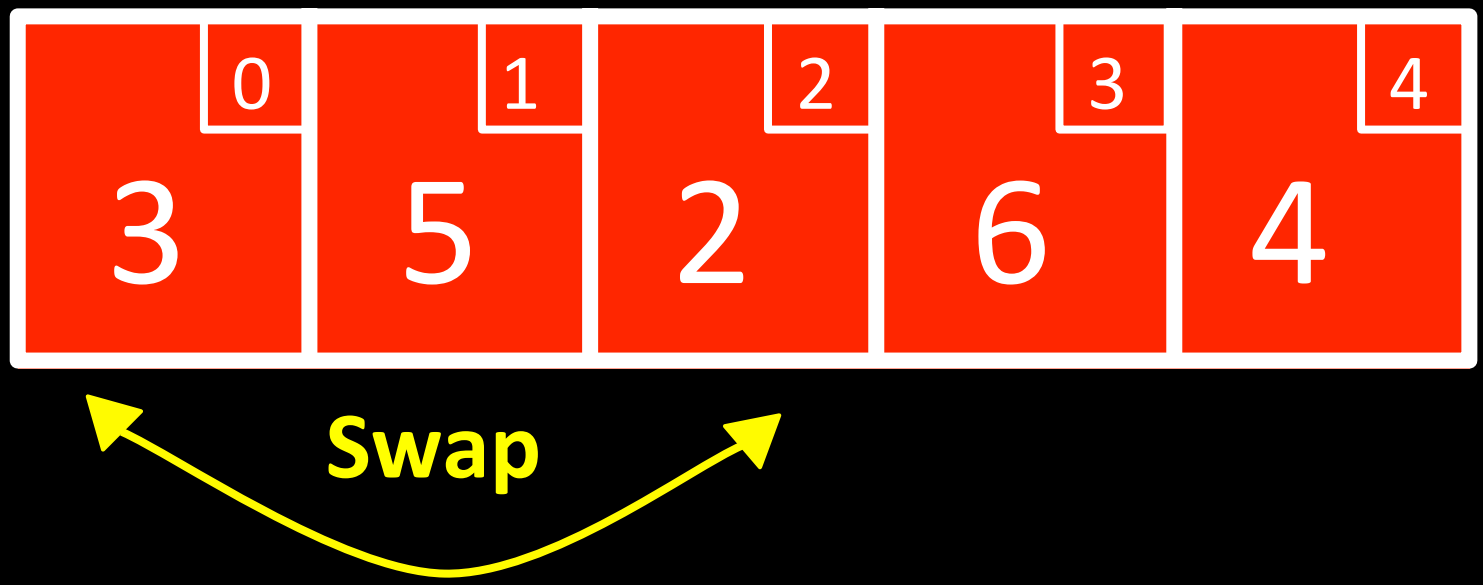
Unsorted



First pass:
2 is smallest, swap with 3

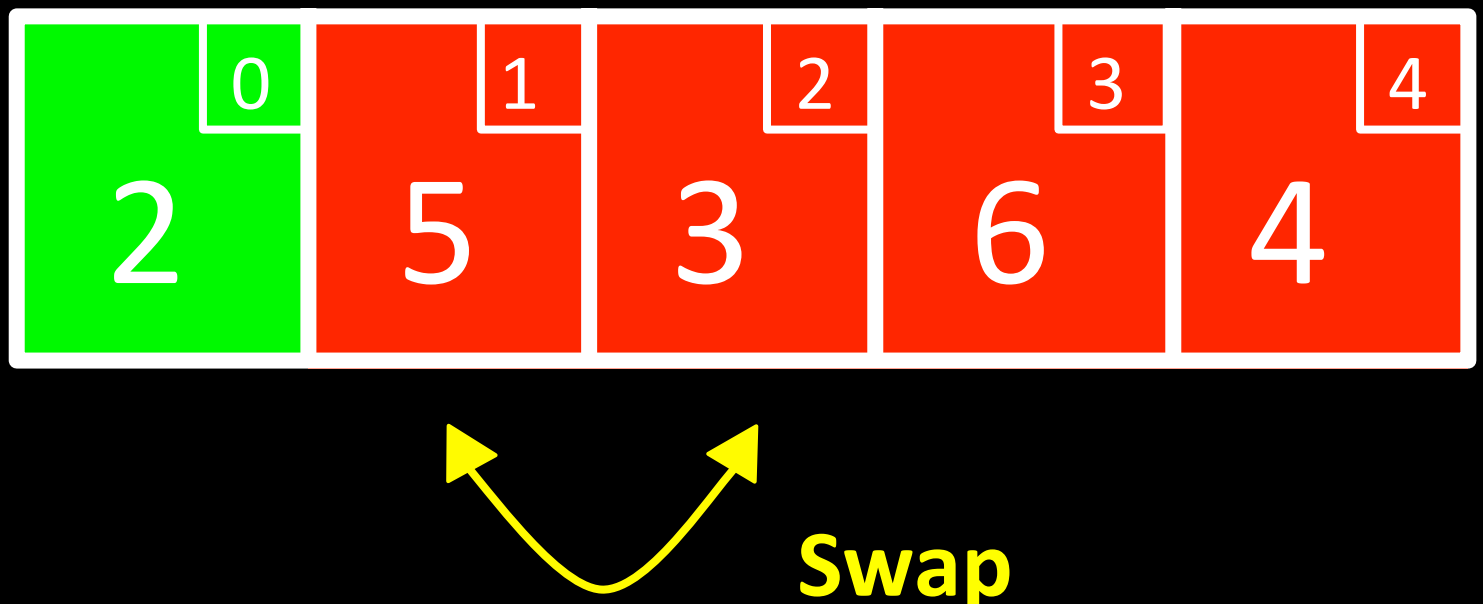
Sorted

Unsorted



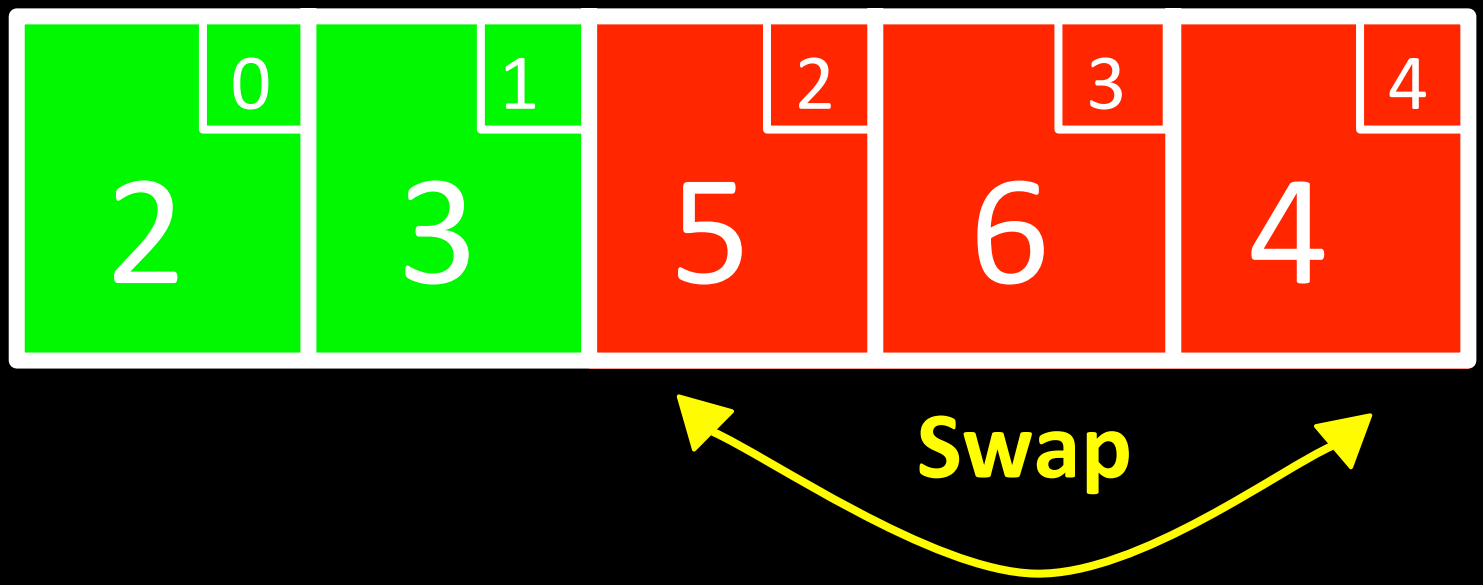
Second pass:
3 is smallest, swap with 5

Sorted **Unsorted**



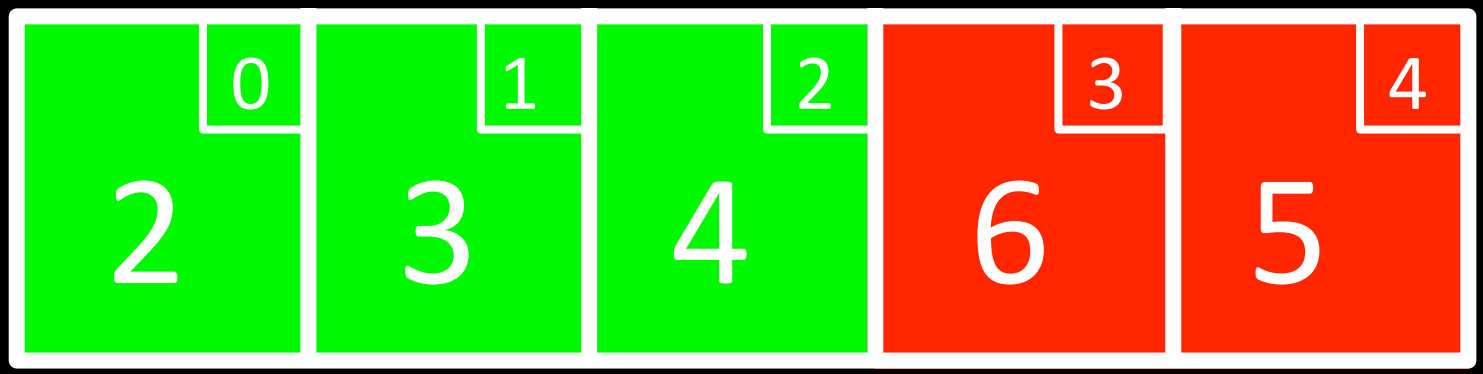
Third pass:
4 is smallest, swap with 5

Sorted **Unsorted**



Fourth pass:
5 is smallest, swap with 6

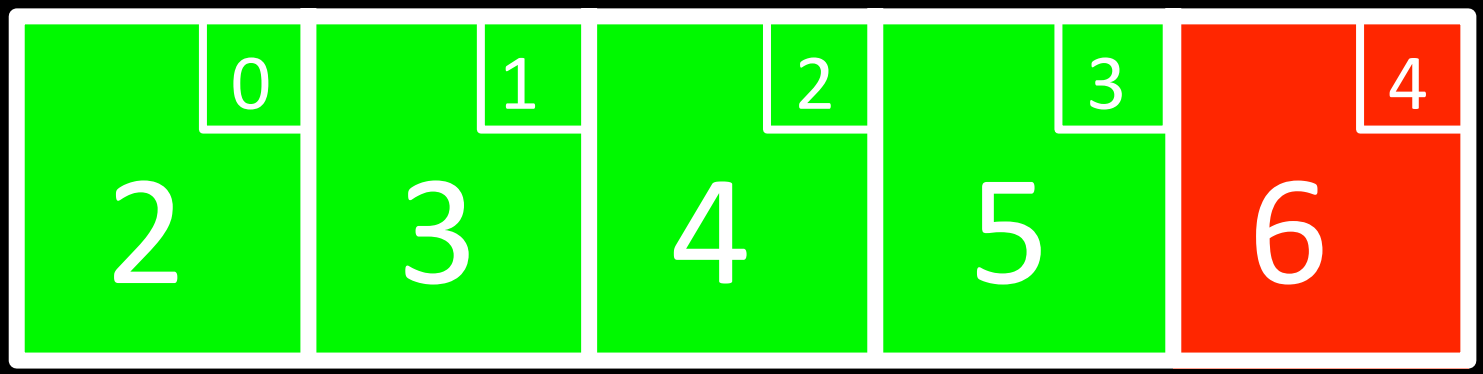
Sorted **Unsorted**



Swap

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

Sorted **Unsorted**

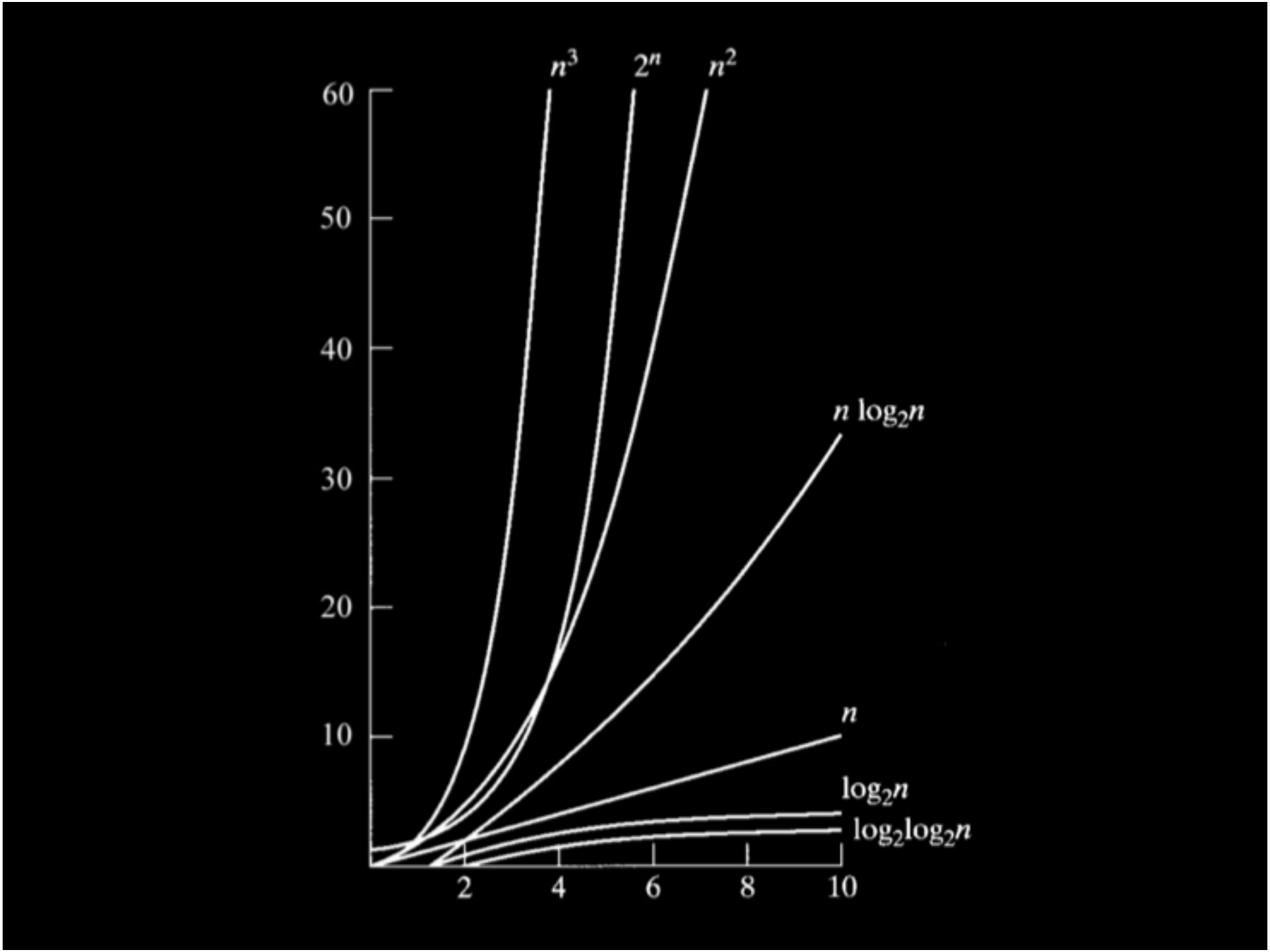


```
for i = 0 to n - 2
        min = i
        for j = i + 1 to n - 1
            if array[j] < array[min]
                    min = j;
        if min != i
                swap array[min] and array[i]
```

What's the best case runtime of selection sort?

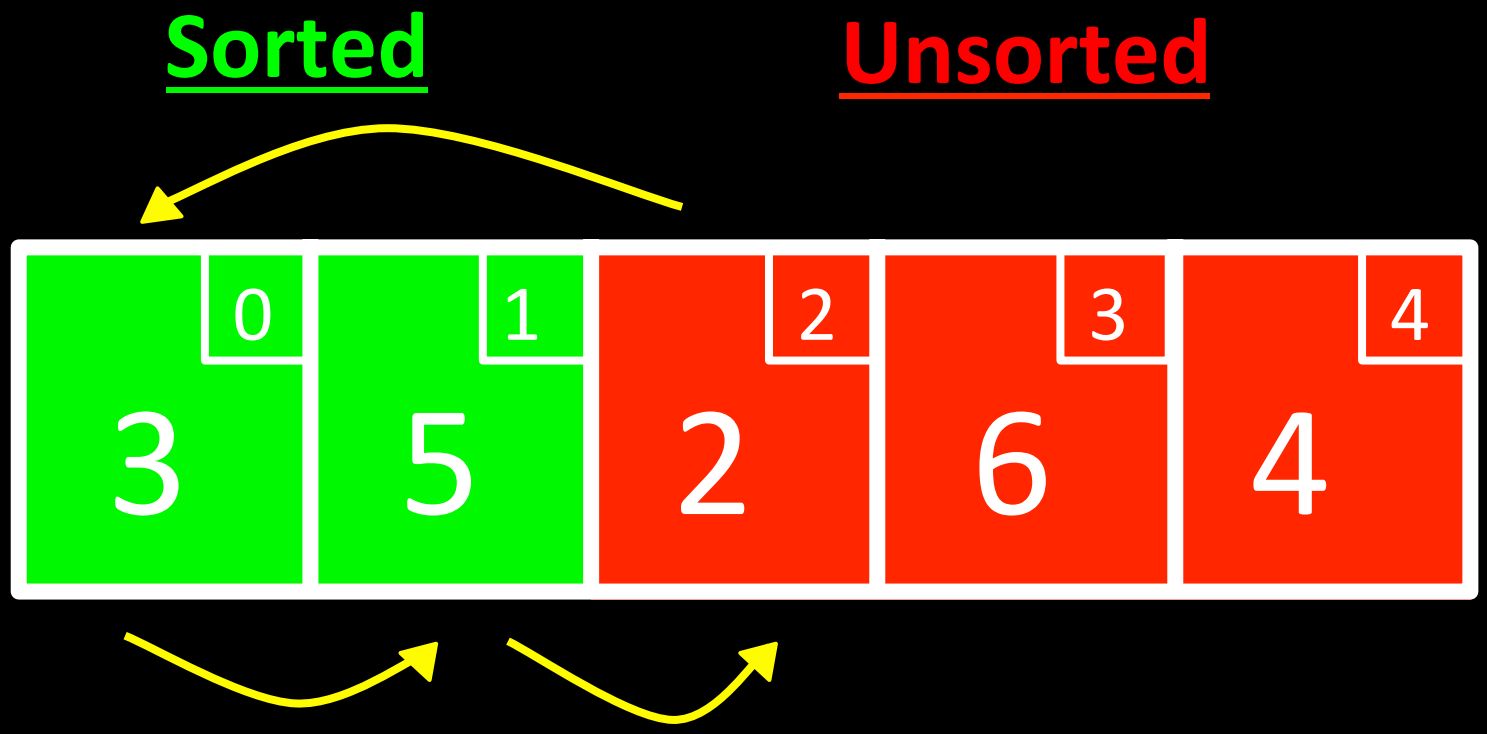
What's the worst case runtime of selection sort?

What's the expected runtime of selection sort?


n^3
2^n
n^2
60
50
40
30
20
10
$n \log_2 n$
n
$\log_2 n$
$\log_2 \log_2 n$
2
4
6
8
10

	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort
O	n^2	n^2	n^2	$nlogn$
Ω	n	n^2	n	$nlogn$
Θ		n^2		$nlogn$

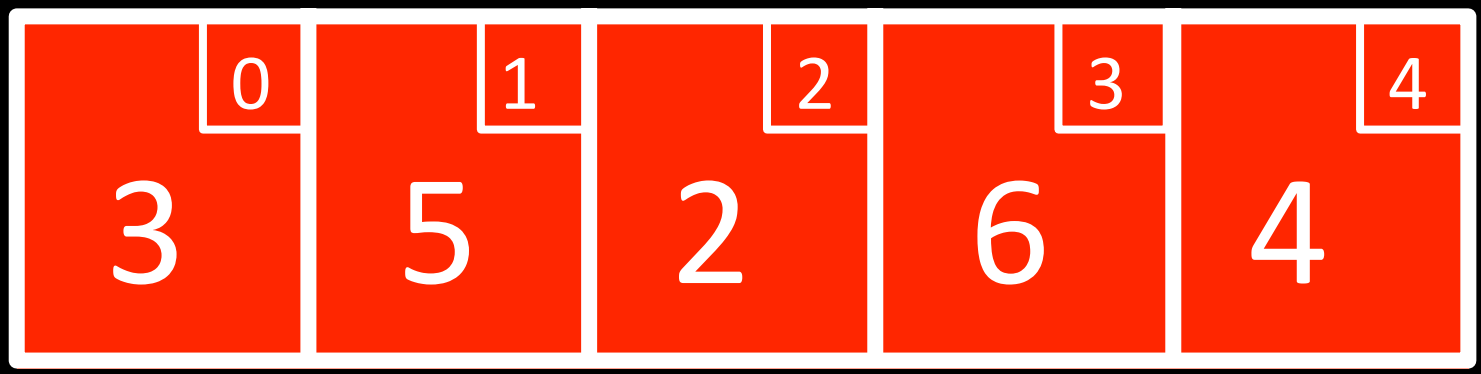
Insertion Sort



All values start as Unsorted

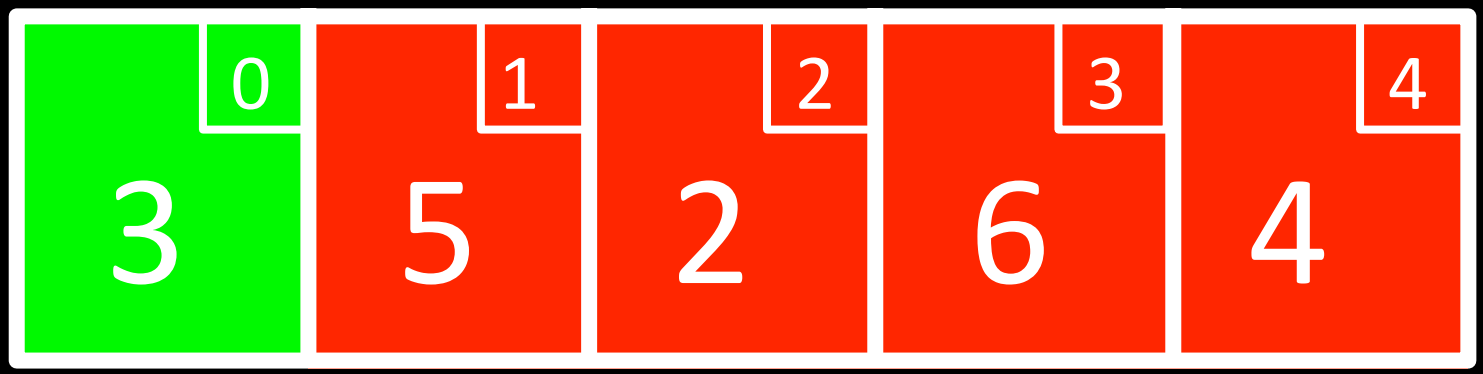
Sorted

Unsorted



Add first value to Sorted

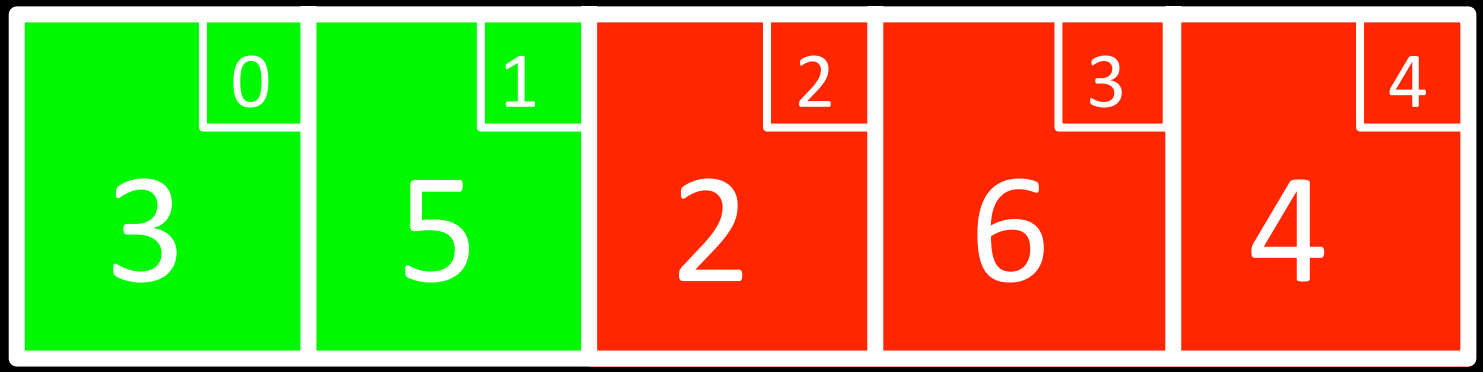
Sorted **Unsorted**



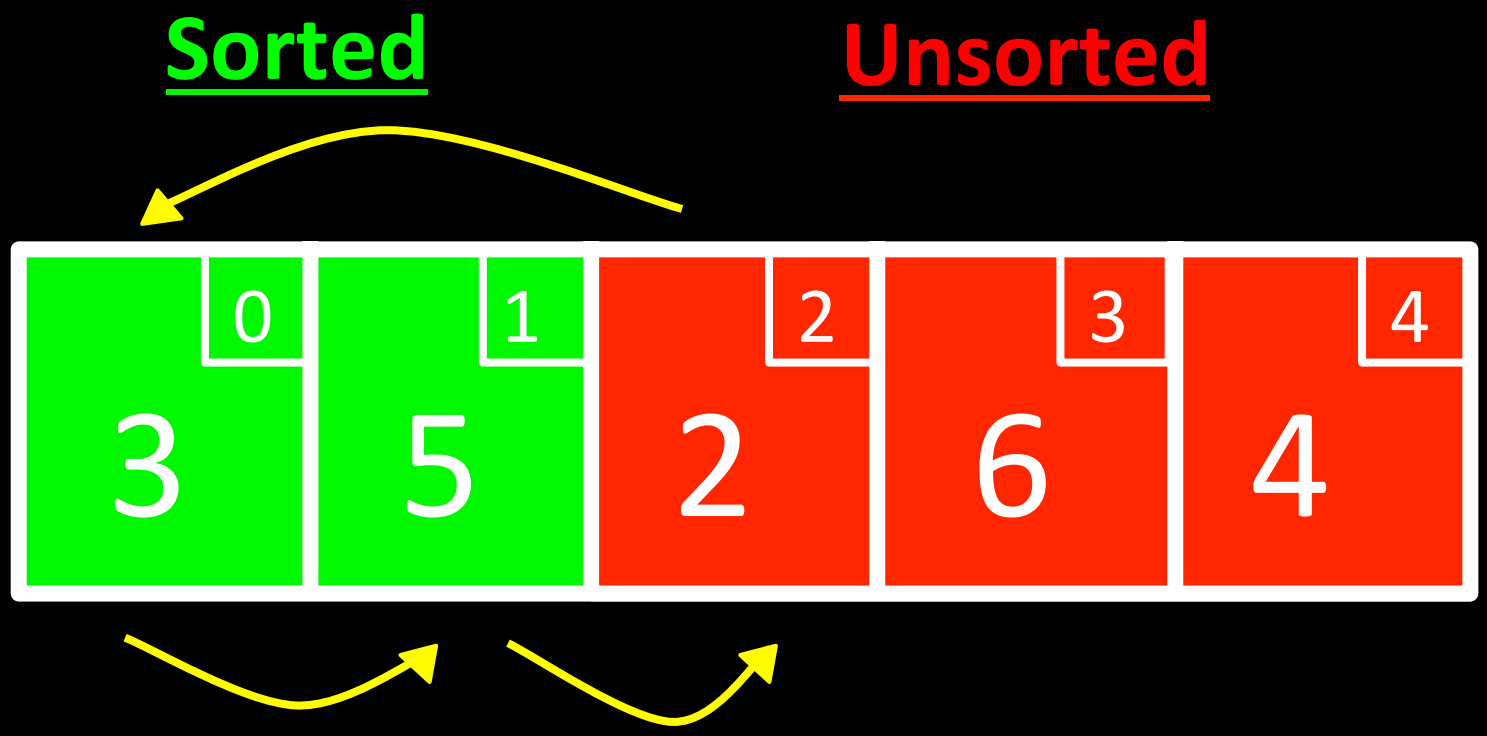
5 > 3

insert 5 to right of 3

Sorted **Unsorted**



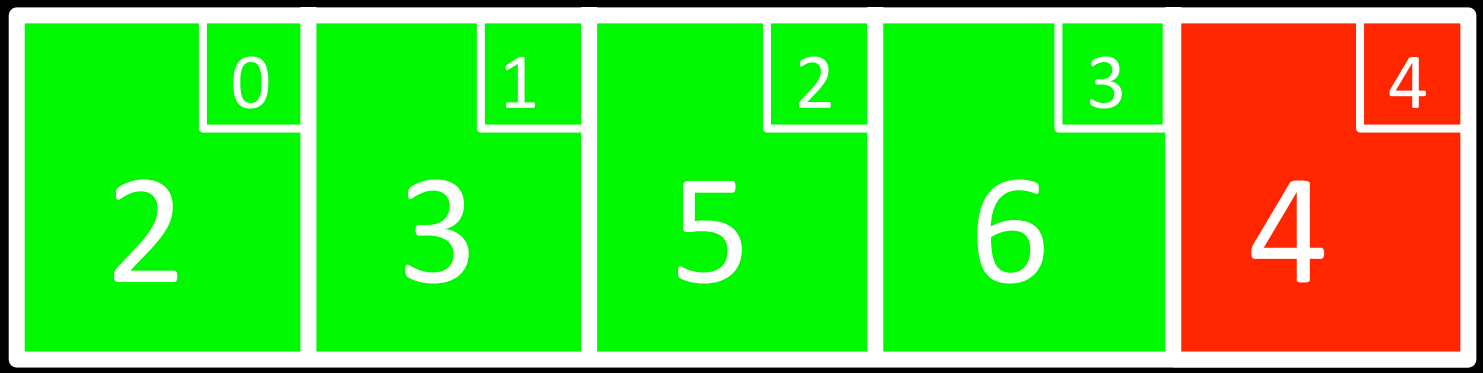
2 < 5 and 2 < 3
shift 3 and 5
insert 2 to left of 3



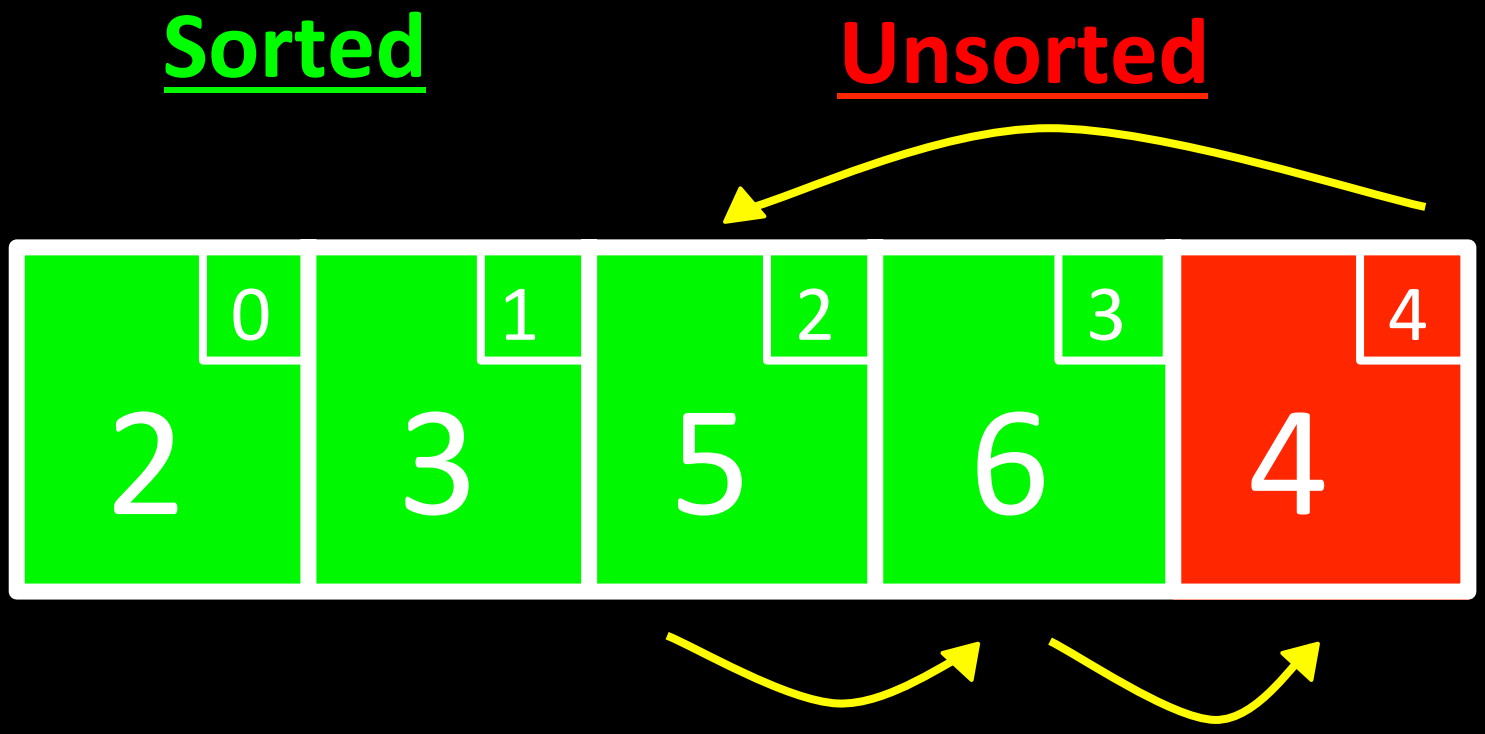
6 > 5

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



For each unsorted element n:

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

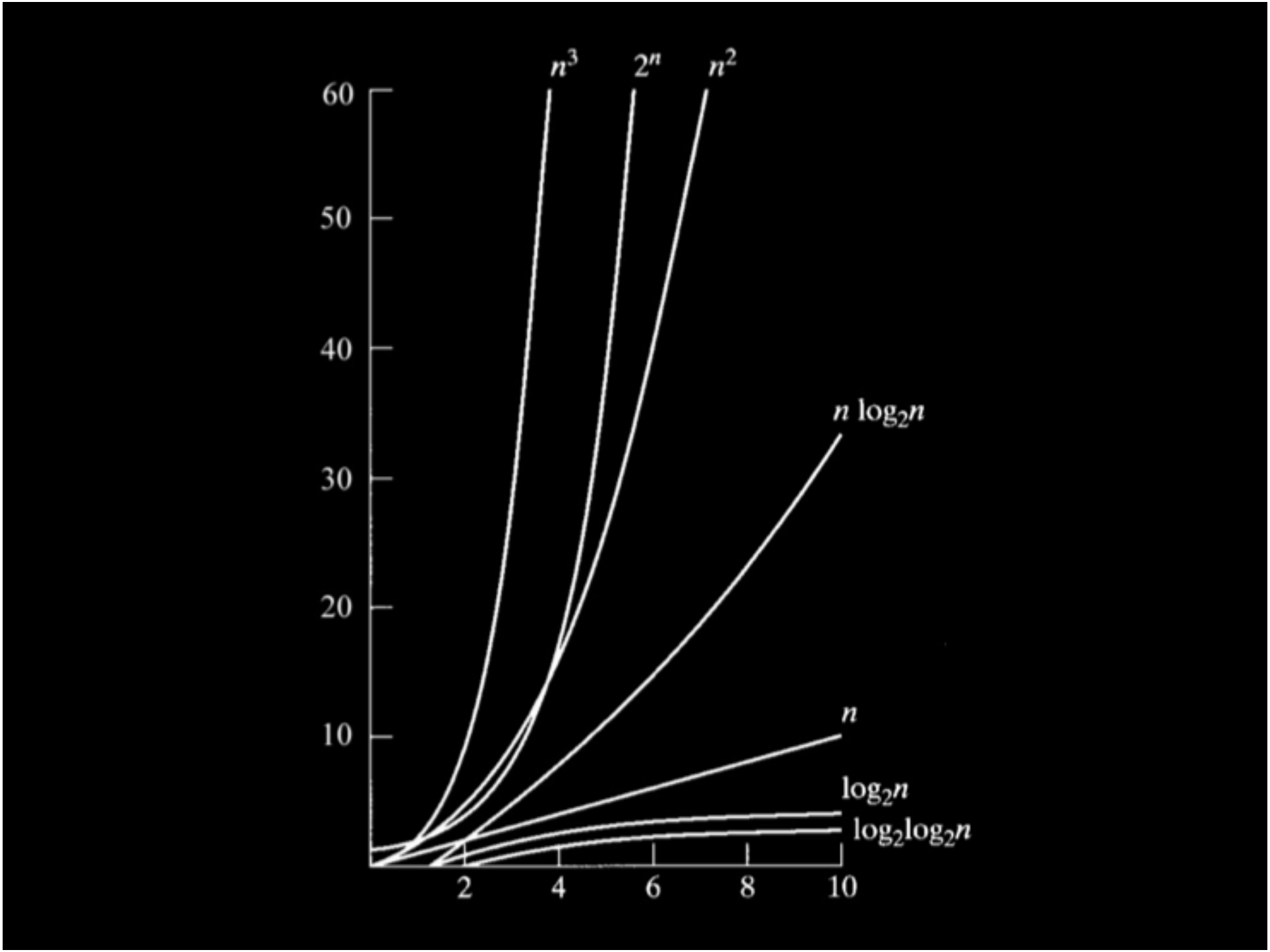
2. Shift sorted elements rightwards as necessary to make room for n

3. Insert n into sorted portion of the list

```
for i = 0 to n - 1
        element = array[i]
        j = i
        while (j > 0 and array[j - 1] > element)
                array[j] = array[j - 1]
                j = j - 1
        array[j] = element
```

What's the worst case runtime of insertion sort?

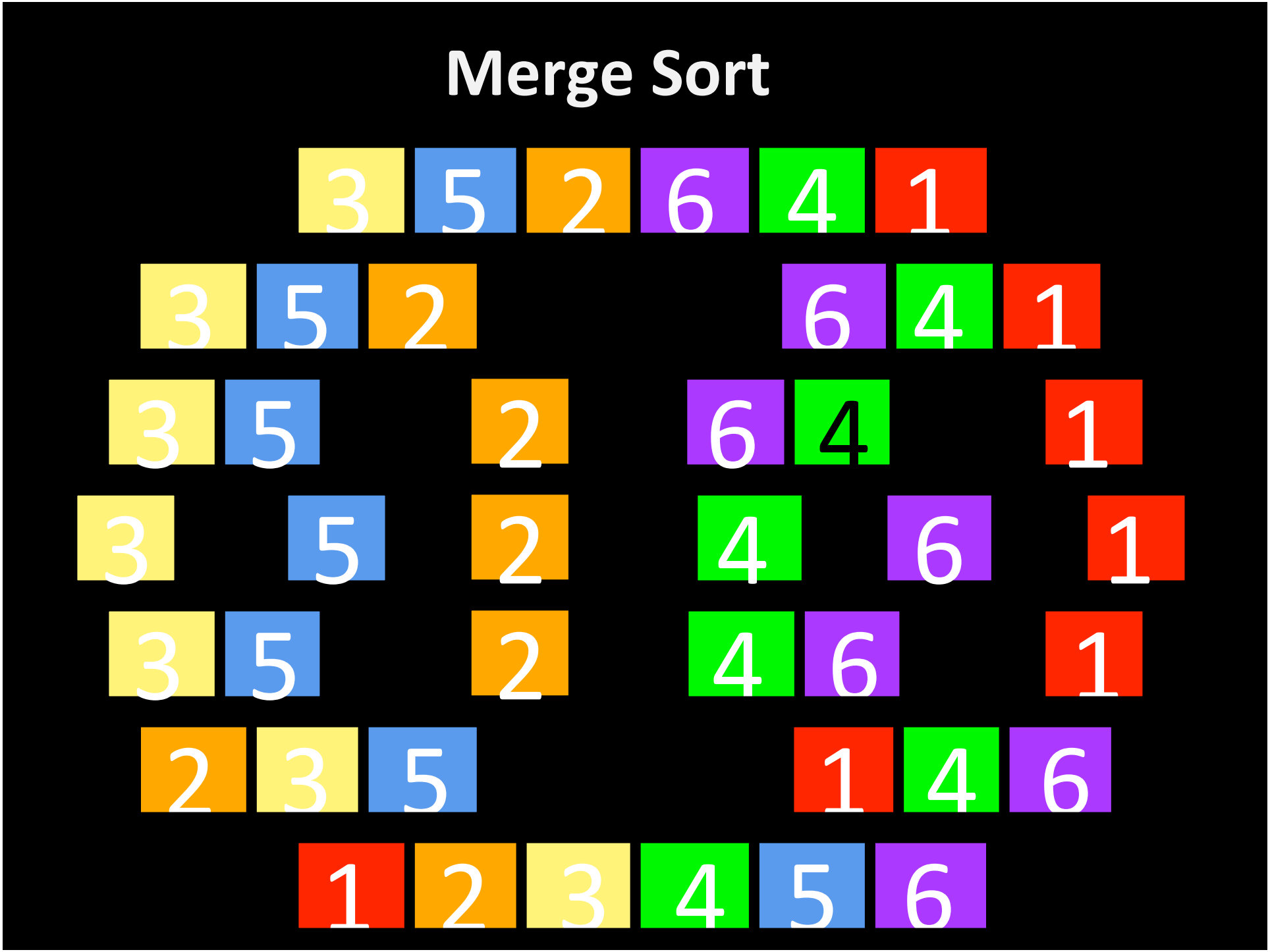
What's the best case runtime of insertion sort?


n^3
2^n
n^2
60
50
40
30
20
10
$n \log_2 n$
n
$\log_2 n$
$\log_2 \log_2 n$
2
4
6
8
10

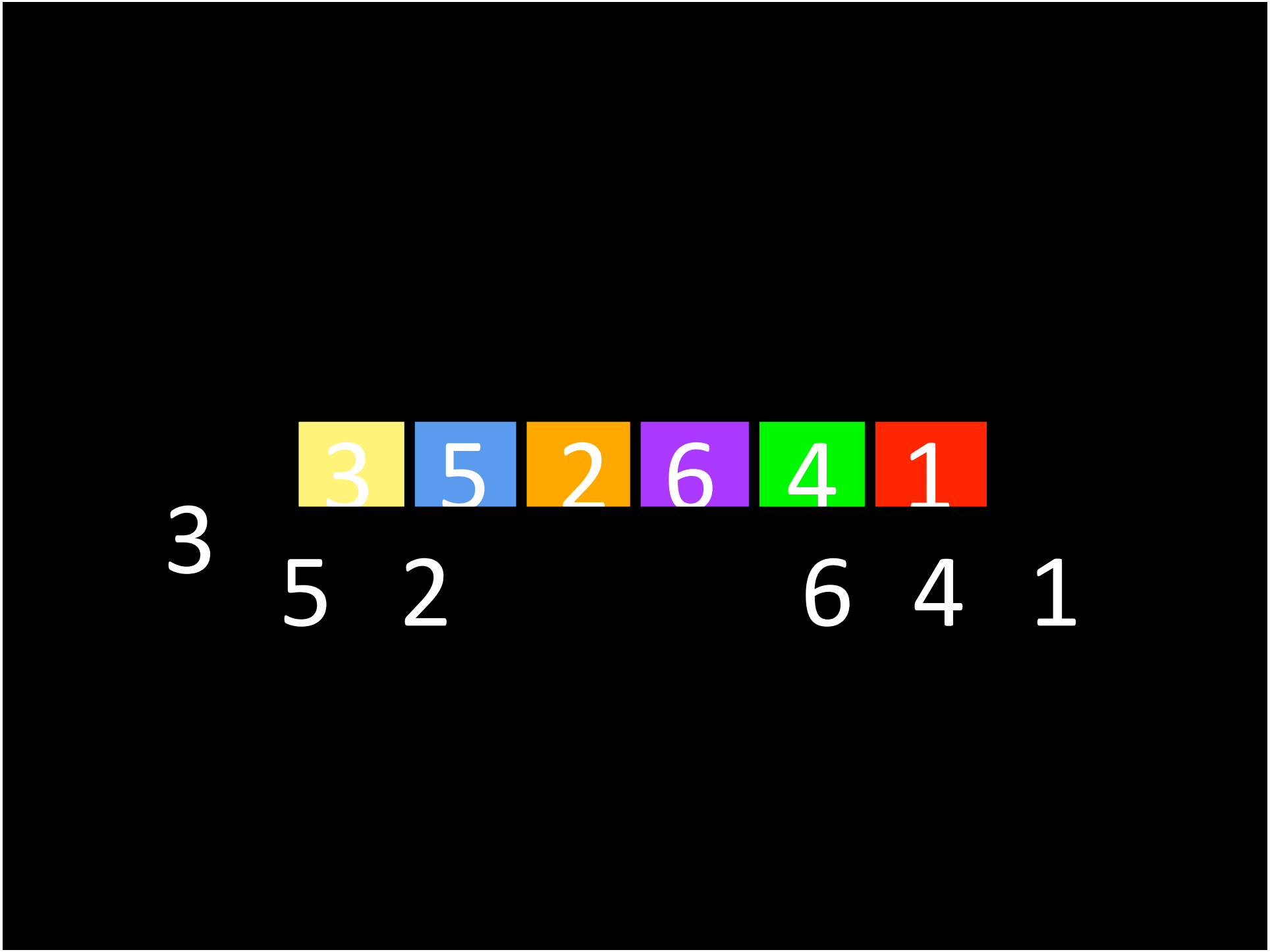
	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort
O	n^2	n^2	n^2	$nlogn$
Ω	n	n^2	n	$nlogn$
Θ		n^2		$nlogn$

What are some differences between the sorts?

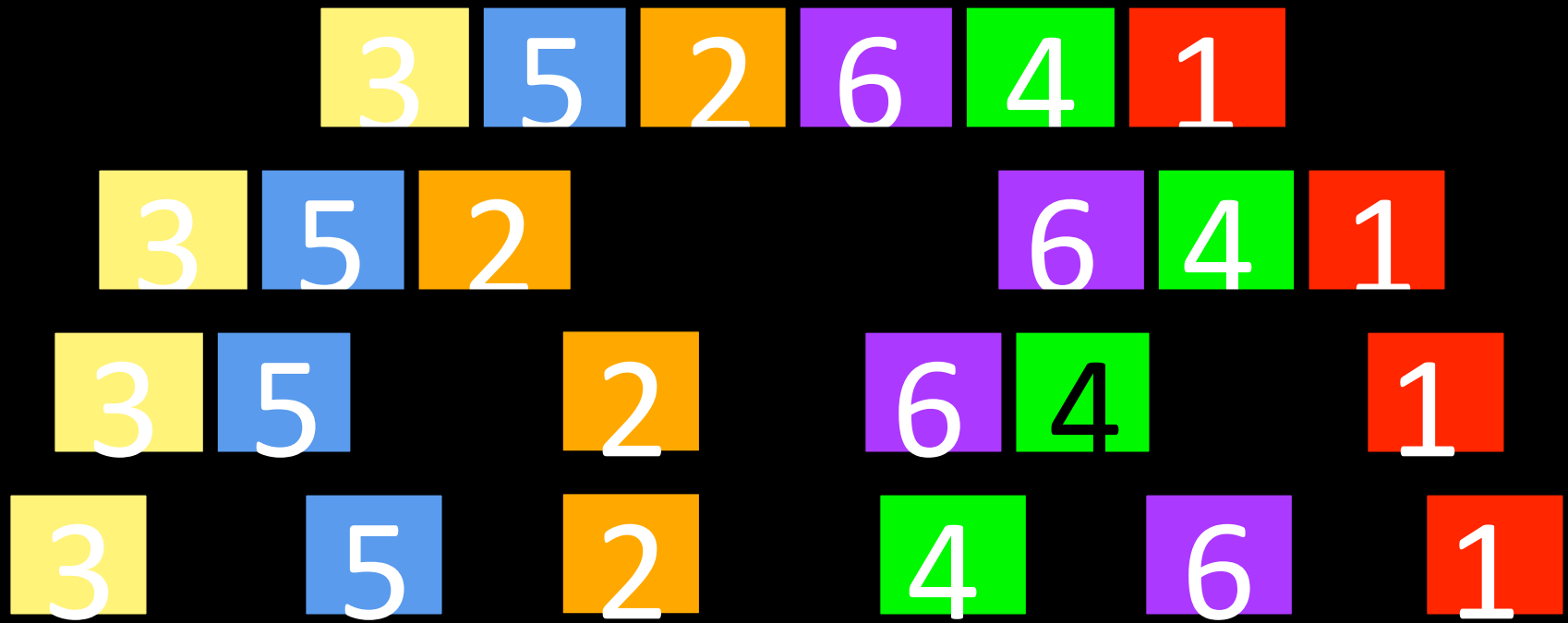
Merge Sort



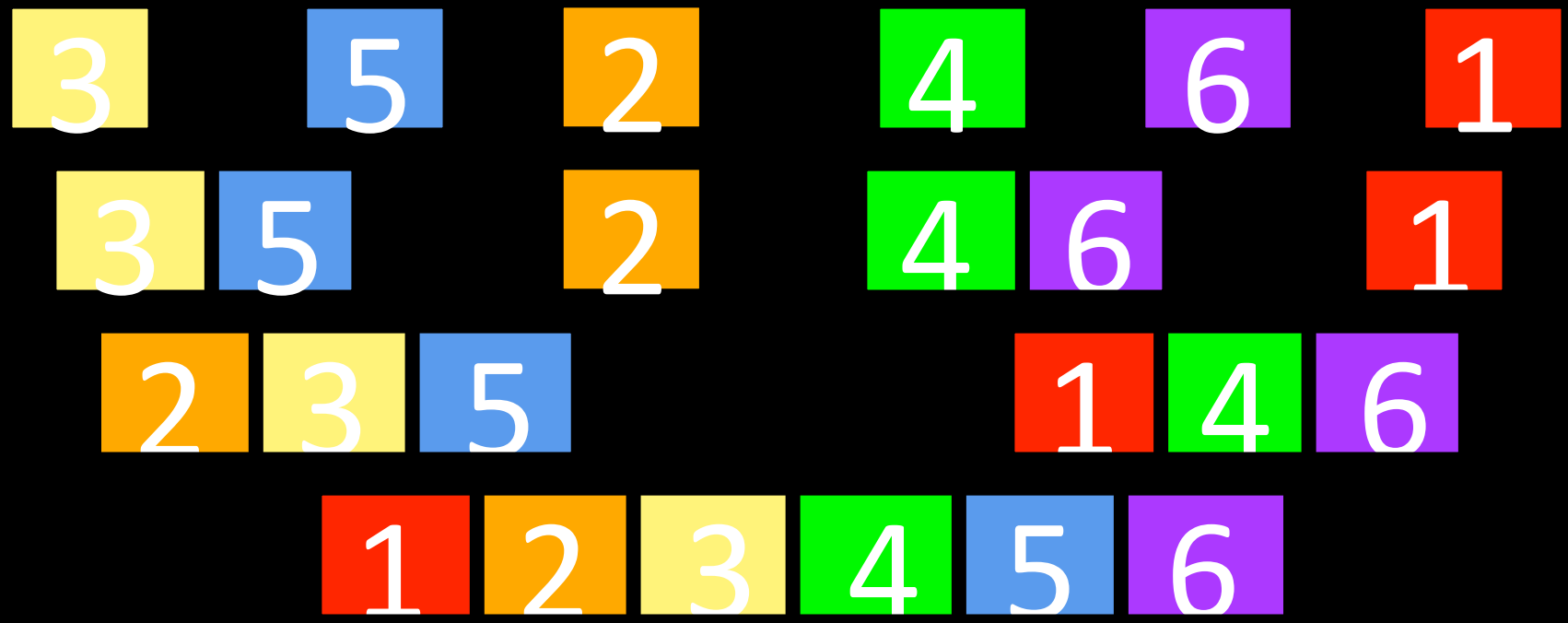
```
On input of n elements:
    If n < 2
            Return.
    Else
            Sort left half of elements.
            Sort right half of elements.
            Merge sorted halves.
```


3 5 2 6 4 1
3
5 2
6 4 1

Halve until each subarray is size 1



Merge Sorted Halves

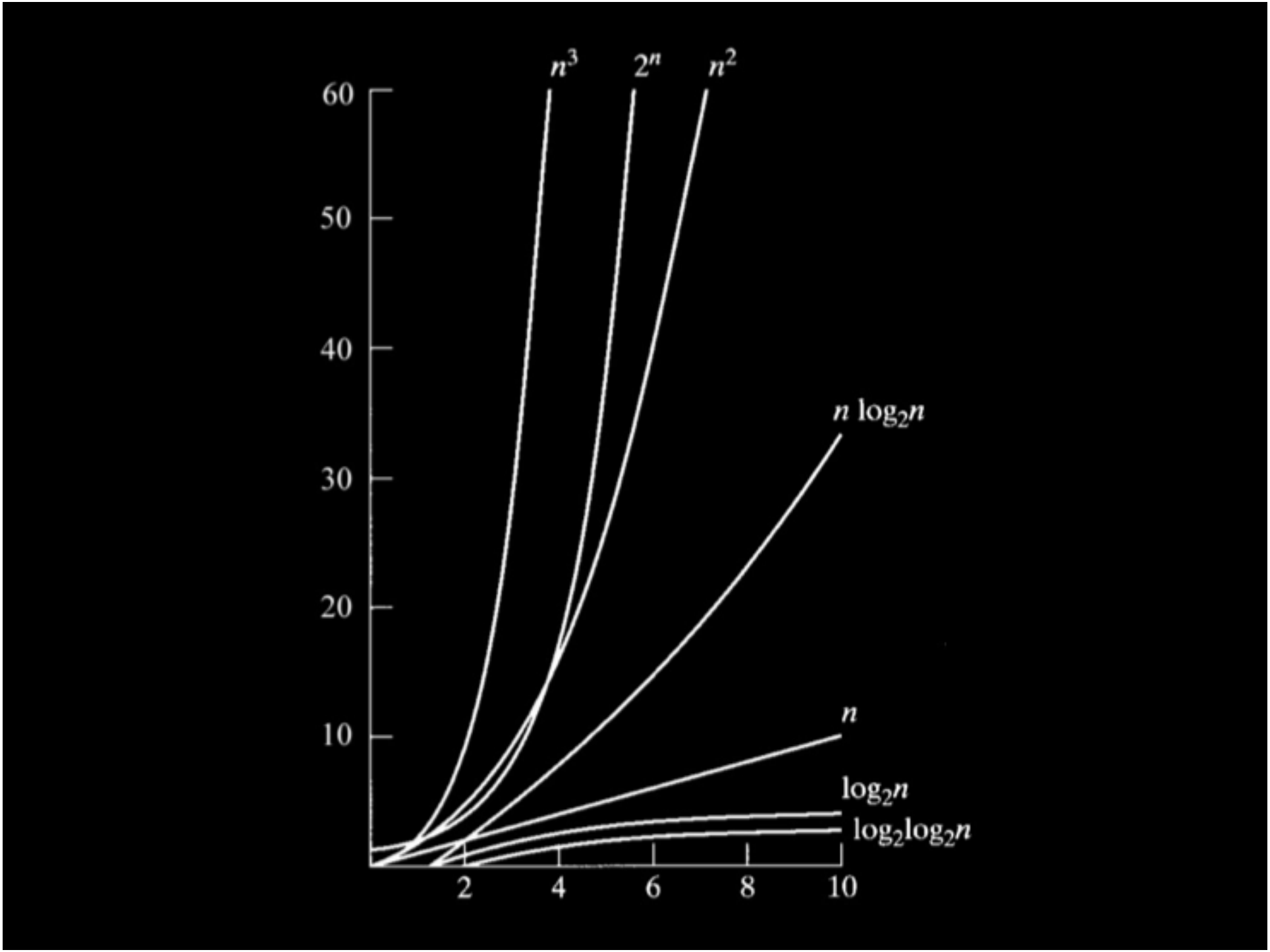


```
sort (int array[], int start, int end)
{
  if (end > start)
        {
            int middle = (start + end) / 2;

                sort(array, start, middle);
                sort(array, middle + 1, end);

                merge(array, start, middle, middle + 1, end);
        }
}
```

What's the best case runtime of merge sort?

What's the worst case runtime of merge sort?

What's the expected runtime of merge sort?


n^3
2^n
n^2
60
50
40
30
20
10
$n \log_2 n$
n
$\log_2 n$
$\log_2 \log_2 n$
2
4
6
8
10

	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort
O	n^2	n^2	n^2	$nlogn$
Ω	n	n^2	n	$nlogn$
Θ		n^2		$nlogn$