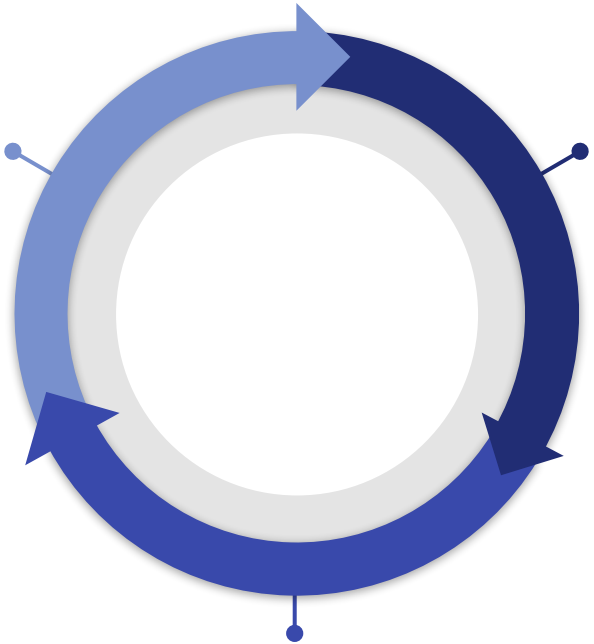
Whiteboard Interview Jam

Chung-thuy Pham

LEARN

Common CS fundamentals asked during interviews



CODE

Apply what we learned by practicing coding problems

SHARE

Our knowledge with one another to improve

Big O

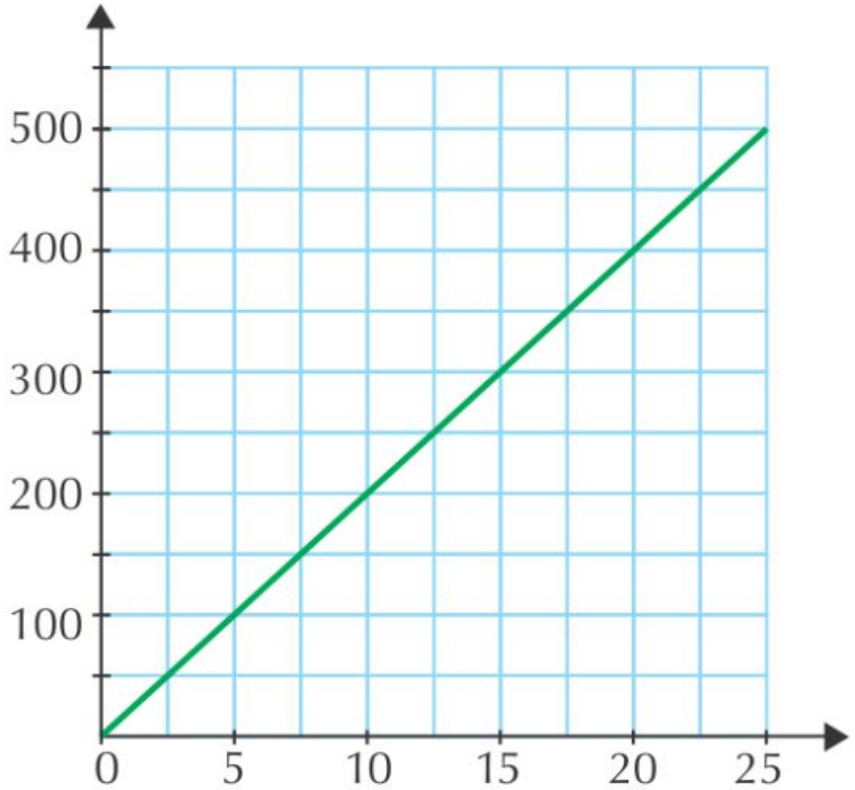
Describes the time efficiency of algorithms allowing us to express how runtime scales based on the size of the data.

Simple for loop

```
for (int a : arrA){
    print(a);
}

for (int b : arrB) {
    print(b);
}
```

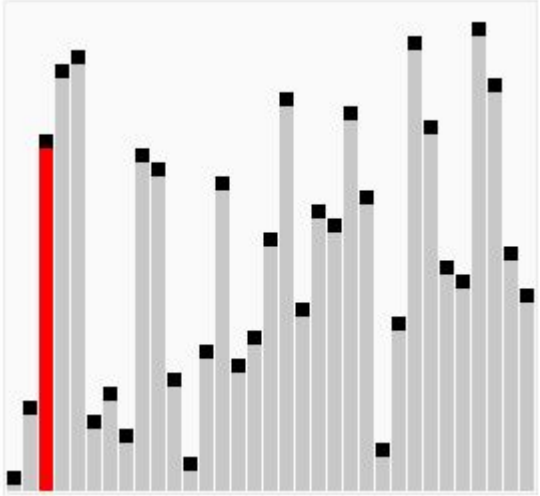
O(n)



Bubble sort

6 5 3 1 8 7 2 4

Compares two adjacent values, higher number gets swapped and "bubbles up" to the top.



```
void bubbleSort(int arr[])
{
    int n = arr.length;
    for (int i = 0; i < n-1; i++)
        for (int j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
            {
                // swap arr[j+1] and arr[i]
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
}
```

https://www.geeksforgeeks.org/bubble-sort/

Runtime scaling

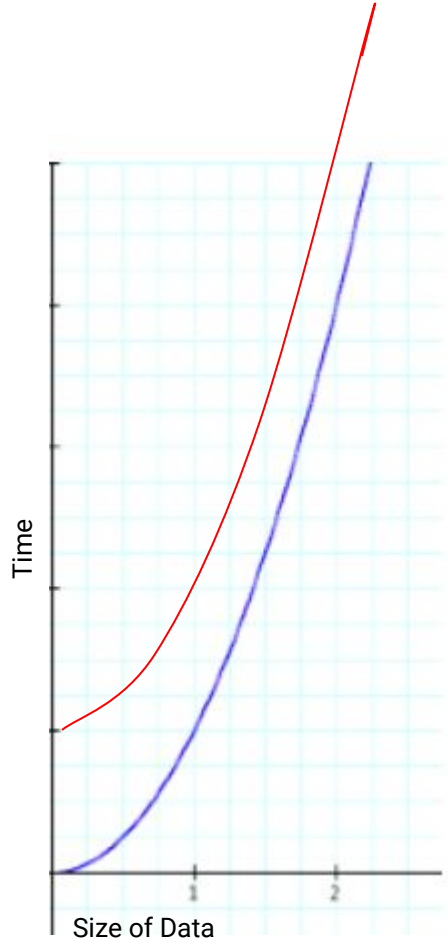
The amount of time it takes to bubble sort a data set as the size of the data grows

As the data increases, the time it takes to sort that data increases exponentially.

This is the graph of n^2, therefore the Big O of a bubble sort algorithm is $O(n^2)$

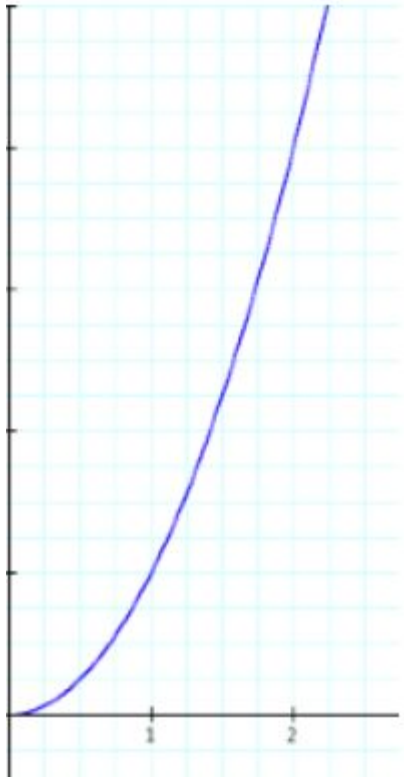
What happens if I run this on a MUCH slower or faster machine?

$O(n^2)$



Nested for loop

```
for (int a : arrA){
    for (int b: arrB){
            print(a + "," + b);
      }
}
```



BEST CASE

Algorithm's behavior under optimal conditions

'least amount of loops'

WORST CASE

Longest path through the algorithm

'maximum number of loops'

EXPECTED CASE

Usually goes hand-in-hand with worst case.

Also called Average case

How to calculate Big O without graph?

```
for (int a : arrA){
    print(a);
}

for (int b : arrB) {
    print(b);
}
```

```
for (int a : arrA){
    for (int b: arrB){
            print(a + "," + b);
      }
}
```

Remember to

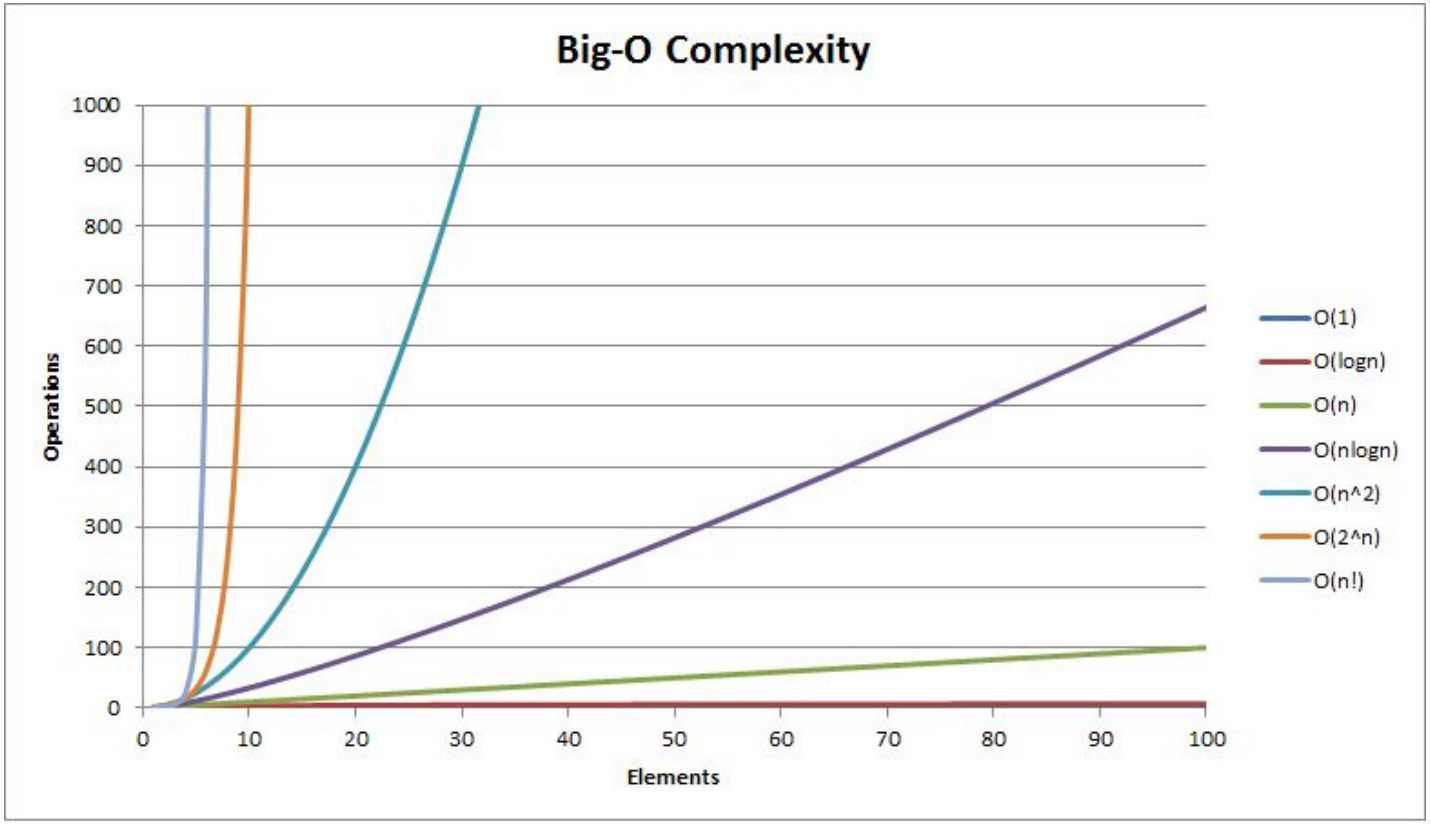
Drop the constants

Drop the non-dominant terms

Merge sort

6 5 3 1 8 7 2 4

https://www.toptal.com/developers/sorting-algorithms/merge-sort


Big-O Complexity
Operations
1000
900
800
700
600
500
400
300
200
100
0
0
10
20
30
40
50
60
70
80
90
100
Elements
O(1)
O(logn)
O(n)
O(nlogn)
O(n^2)
O(2^n)
O(n!)