#algorithms

Iftekhar Uddin

Problem 0

Fib outputted

```
Entering fib(5)... Press Enter to continue.
Entering fib(4)... Press Enter to continue.
Entering fib(3)... Press Enter to continue.
Entering fib(2)... Press Enter to continue.
Entering fib(1)... Press Enter to continue.
Entering fib(0)... Press Enter to continue.
Entering fib(1)... Press Enter to continue.
Entering fib(2)... Press Enter to continue.
Entering fib(1)... Press Enter to continue.
Entering fib(0)... Press Enter to continue.
Entering fib(3)... Press Enter to continue.
Entering fib(2)... Press Enter to continue.
Entering fib(1)... Press Enter to continue.
Entering fib(0)... Press Enter to continue.
Entering fib(1)... Press Enter to continue.
```

I'm not sure if that's the appropriate stack.

Problem 1

2. The worst case scenario is when every single item has to be swapped back one. It runs N times with N being the length of the array as it has to go through and compare. This is seen by the for loop in insertion sort.

   N

   Then it has to go through (N-1) times because after the first N times, we only have the first item in the correct place. The second N times the loop runs, the second item will be in the correct place and so on.

   (N-1)

then Because of the number of arrays being K, this has to go through an extra K times

K(N(N-1))

But K is a constant that will always run K times so it's not relevant. Then simplified to the biggest polynomial.

O(N^2)

---

1. The way it can be improved is by having a pointer at each array we're given and as we go through a new array of length K x N, insert the smallest number and increment the pointer at that array. We can do this because the arrays given are already sorted. This would make the algorithm run exactly K x N times or n for short giving us O(n).

Problem 2

1. It only runs once through the array. A summation from 1 to N. This runs O(n)
2. I genuinely don't know how to make this faster. A binary tree method to get it to be O(logn)?