CSCI 305 Assignment 1

(Solo) Isaac Boaz

October 24, 2023

Problem 1

```
1  for i = 1 to n
2      j = 1
3      while j<= i do
4          j = 2.5 * j
```

cost	time
C_1	$n+1$
C_2	n
C_3	$(n+1)\log_{2.5}$
C_4	$n\log_{2.5} n$

Detailed Runtime:

$$C_1 \cdot (n+1) + C_2 \cdot n + C_3 \cdot \lceil (n+1)\log_{2.5} \rceil + C_4 \cdot \lceil n \log_{2.5} n \rceil$$

Asymptotic Runtime:

$$\Theta(n \log n)$$

Problem 2

```
0  def factorial(n):
1      x = 1
2      while n > 1:
3          x = x * n
4          n = n - 1
5      return x
```

cost	time
C_1	1
C_2	n
C_3	$n-1$
C_4	$n-1$
C_5	n

Detailed Runtime:

$$C_1 \cdot 1 + C_2 \cdot n + C_3 \cdot (n-1) + \cdot C_4 \cdot (n-1) + C_5 \cdot n$$

Asymptotic Runtime:

$$O(n)$$

Problem 3

```
0  def Binary_InsertionSort(A):
1      for j = 2 to n:
2          key = A[j]
3          // insert A[j] into the
               sorted sequence A[1..j-1]
4          left = 1
5          right = j - 1
6          while right > left:
7              mid = floor((left + right
                   ) / 2)
8              if key > A[mid]:
9                  left = mid + 1
10             else right = mid
11         if key > A[left]:
12             left = left + 1
13         for i = j downto (left + 1):
14             A[i] = A[i - 1]
15         A[left] = key
```

best	worst
n	n
$n-1$	$n-1$
$n-1$	$n-1$
$n-1$	$n-1$
$n-1$	$n-1$
$(n-1)(2)$	$\sum_{j=2}^{n+1} \log(j)$
$(n-1)$	$\sum_{j=2}^{n} \log(j)$
$(n-1)$	$\sum_{j=2}^{n} \log(j)$
$(n-1)/2$	$\sum_{j=2}^{n} \log(j)$
$(n-1)/2$	$\sum_{j=2}^{n} \log(j)$
$n-1$	$n-1$
$(n-1)/2$	$(n-1)/2$
$n-1$	$\sum_{j=2}^{n+1} j$
$n-1$	$\sum_{j=2}^{n} j$
$n-1$	$n-1$

1. We can tell that the best case scenario will occur when the array is already sorted, as the binary search will exit after one check, and the inner for loop will only run once to push items. The worst case scenario occurs when the array is reverse-sorted, as the inner for block will have to run n times to push items.

2. For best case, the while block is run once for each iteration of the loop, resulting in

$$\sum_{j=2}^{n} 1 \text{ runtime, which at max is } O(n)$$

   For worst case, the while block is run $\log j$ times for each iteration of the loop, resulting in

$$\sum_{j=2}^{n} \log j \text{ runtime, which at max is } O(n \log n)$$

3. The if block is not in any loop aside from the outer-most for loop. Since the for loop's best and worst case runtime is the same, the if's runtime will also be the same at

$$O(n)$$

4. For best case, the for loop is only run once, resulting in

$$\sum_{j=2}^{n} 1 \text{ runtime, which at max is } O(n)$$

   For worst case, the for loop is run j times, resulting in

$$\sum_{j=2}^{n} j \text{ runtime, which at max is } O(n^2)$$

5. Note that regardless of input, the for loop will always run n times, with the contents of the loop running $n-1$ times.

   Worst Case occurs when the array is reverse sorted. Within the top level for loop, the *while* loop will run $\log j$ times, the if block will run once, and the inner *for* loop will run j times.

$$\sum_{j=2}^{n} O(j + \log j + 1)$$

   This boils down to $O(n^2)$

   Best Case occurs when the array is already sorted, as the left and right bounds will only need to be updated once. Within the top-level for loop, the *while* loop will run once, the if

block will run once, and the inner *for* loop will run once.

$$\sum_{j=2}^{n} O(1+1+1)$$

This boils down to $O(n)$

6. Given that the primary improvement to binary insertion sort is that binary search is used to find the proper index for each key, I do not see any situation where this version would be *worse* than standard insertion sort. For example, if the array is already sorted, the binary search will only run once, and the inner for loop will only run once, resulting in a runtime of $O(n)$, which is the same as standard insertion sort.