CS301 Algorithms - Homework 1

Ege Demirci - 28287

Sabanci University, Spring 2022-2023, Algorithms

1. Question - 1

1.1. Part - A

It is observed worst case for insertion sort when the input array is in reverse sorted order. In this array form, the largest element is at the beginning of the array and each subsequent element is smaller than the previous element, naturally, to sort such an array, it is necessary to move the last element to the starting index and the first element to the ending index.

1.2. Part - B

The worst-case time complexity of insertion sort for an input array of size n in reverse sorted order is $\Theta(n^2)$.

1.3. Part - C

If we analyze basically, in worst case each element of the array's must be compared to every other element and shifted to its correct position in insertion, shifting in this case will identify running time of the algorithm. So for every number from $k = 2$ to $k = n$ (size of the array) there will be k-1 shifts. To find the total number of shifts, we can use sum formula. Since we want to find the summation of the numbers from 1 to n, this resulting in $n(n-1)/2 = \Theta(n^2)$. The worst case time complexity of insertion sort is $\Theta(n^2)$

If we analyze it referring to the insertion sort algorithm in the lecture slides, the inside of the outer loop will iterate n-1 times, the reason is each element must be compared to the preceding elements and shifted to its correct position. The inner while loop is always executed and perform swaps, because the condition A[i] $>$ num is always true. So the inside of the inner loop will also have to iterate j-1 times for each i iteration, where j is the index of the element being inserted and i is the index of the element to its left.

So the key point here is inner loop. Because worst-case input causes the inner loop to execute the maximum number of times, and this lead to n^2 time complexity. For these reasons this form of array results in maximum number of comparisons and swaps and this leads to $\Theta(n^2)$ complexity.

2. Question - 2

2.1. Part - A

It is observed best case for insertion sort when the input array is in already sorted order. In this array form, the smallest element is at the beginning of the array and each subsequent element is larger than the previous element, naturally, to sort such an array, there is no need to swap anything since it's already sorted ascending so the outer loop of the algorithm will iterate but the inner loop will not execute at all.

2.2. Part - B

The best-case time complexity of insertion sort for an input array of size n in already sorted order is $\Theta(n)$.

2.3. Part - C

If we analyze basically, in best case still each element of the array must be compared to every other element but there will be zero swaps/shifts since all the elements are in already sorted position. For this reason, the number of operations performed by the algorithm just will be related to the outer loop and the time complexity is therefore Θ(n), or in other words for every insertion, it will take constant time since the array is already sorted, so from k=2 to n, we will sum the 1s, the result of this (1+1+1....) from k=2 to n will be n-1 so again the time complexity is therefore Θ(n)

If we analyze it referring to the insertion sort algorithm in the lecture slides, the inside of the outer loop will iterate n-1 times. The inner while loop is never executed, because the condition A[i] > num is always false. This means that, for each j from 2 to n (where n is the size of the array and j second index), the algorithm simply assigns A[j] to num and then finds the correct position of num by comparing it with the elements to its left, without having to perform any swaps. So it only does comparison operation.

Again, the key point here is inner loop. Because best-case input causes the inner loop to execute zero times, this lead to n time complexity. For these reasons this form of array results in maximum number of comparisons and 0 swaps and this leads to Θ(n)*complexity*.

3. Question - 3

3.1. Part - A

As it stated in the question In order to show that $(5n + 4)^2 = O(n^2)$ by using the formal definition of O–notation, we need to pick constants c and n0 such that for any n $\geq$ n0 we have.

To show that $(5n + 4)^2 = O(n^2)$ by using the formal definition of O-notation, we need to pick constants c and n_0 such that for any $n \geq n_0$ we have:

$(5n + 4)^2 \leq cn^2$

Expanding the left side of the equality:

$(5n + 4)^2 = 25n^2 + 40n + 16$

Substituting back in the inequality:

$25n^2 + 40n + 16 \leq cn^2$

Dividing each side by n^2:

$25 + \frac{40}{n} + \frac{16}{n^2} \leq c$

Since $n_0 = 2$ is the smallest value possible for n in this question (this comes from n >= n0 since n0 is 2, smallest possible value for n is also 2), we can substitute it directly into the equation:

$(25 + \frac{40}{2} + \frac{16}{2^2}) = 49 \leq c$

Therefore, the smallest possible value for c in this case is 49. Answer = 49

3.2. Part - B

Let's start by expanding the left-hand side:

$(5\text{n} + 4)^2 = 25n^2 + 40n + 16$

Now we can set $c = 36$ and try to find the smallest n_0 such that $25n^2 + 40n + 16 \leq 36n^2$ for all $n \geq n_0$.

Simplifying the inequality, we get:

$25\text{n}^2 + 40n + 16 \leq 36n^2$

$11\text{n}^2 - 40n - 16 \geq 0$

We can solve for the roots of the quadratic equation $11n^2 - 40n - 16 = 0$ to find the intervals where the inequality is true. Using the quadratic formula, we get:

$(11n + 4)(n - 4) \geq 0$

$n \leq -\frac{4}{11} \qquad n \geq 4$

Since we are looking for the smallest n_0, the first solution ($n \leq -\frac{4}{11}$) set doesn't provide anything bigger than 0, so we skip the first solution set. If we combine the information from second solution set ($n \geq 4$) and $n \geq 0$ from the question; we can acquire the solution. Since $n \geq 4$, smallest n_0 value that we can use is 4 in this case. Because for all $n \geq n0$ equation should be correct and since smallest value for n is 4, smallest value for n_0 is also 4. Answer = 4

4. Question - 4

4.1. Part - A

Answer is : $2^{2^n} > n! > n2^n > lg(n!) > (lgn)! > lg^2 n$