Gebze Technical University
Computer Engineering

CSE 222
2017 Spring

HOMEWORK 2 REPORT

DENİZ CAN ERDEM YILMAZ
151044001

Course Assistant: Nur Banu Albayrak

Q1.

1) Three simple statements takes constant time to process which evaluates as Θ(1). Inner for loop iterates (n-i) times for each outer loop which iterates (n-1) times. For best case which is (n=2) Both loops iterate 1 times. With three simple statements T(n) would be 3. So best case will be Ω(1) which is constant time. In worst case total loop would take (n-1) x (n-i) x 3 times which equals $T(n) = 3n^2 - 3in + 3n + 3i$. This can be represented as $O(n^2)$. In overall this alogithm will be between Ω(1) (lowe bound) and $O(n^2)$ (upper bound)
2) As a recursive function there are 2 returns which evaluate in constant time. Best case would be null or empty string which means Ω(1). If we take n as the length of string in worst case T(n) = 1 + T(n-1). On next step T(n-1) = 1 + T(n-2), T(n-2) = 1 + T(n-3) … T(2) = 1 + T(1), T1(1) = 1. If we eliminate same expressions on different sides this recursion evaluates as T(n) = 1+1+1+..+1 (n times) so it would be T(n) = n. This algorithm takes Ω(1) time for worst case, O(n) times for the worst case

Q2.

1) This function takes an array and sorts it from smallest to biggest but does not change first elements position due to outer loops starting value as 1 instead of 0.
2) Best case is when array has minimum elements, which can be 3 to start outer loop. Both loops iterates one time only, evaluates swap in constant time (3 expressions). T(n) for best case is constant which is Θ(1). In worst case, each for loop iterates n times which equals T(n) to n^2. So worst case running time is $Θ(n^2)$ for this algorithm.

Q3.

This relation can be explained by using sandwich thorem (also known as squeeze theorem). This theorem used in calculus to find limit of a mathematical expression by comparing it with 2 other function or expressions named as upper and lower bounds. If an expression is squeezing in between same functions then the expression is also equals to that function. If we apply this rule to algorithm compare notation, if an algorithm is bounded from bottom with a function g(n) (Ω(g(n))) and from upper with function g(n) (O(g(n))) then this algorithm’s work time notation is Θ(g(n))

Q4.

1)

```
function insertionSortR(array A, int n)

      if n>1

            insertionSortR(A,n-1)

            x = A[n]

            j = n-1

            while j >= 0 and A[j] > x

                  A[j+1] ← A[j]

                  j ← j-1
            end while

            A[j+1] = x

        end if

    end function
```

2) As a recursive one, this algorithms worst case working time function if we take (n) the length of the array would evaluates as $T(n) = T(n-1) + 2n + 3$, first a recursive call with (n-1) elements, then 2 simple statements, a loop of 2 simple statments (n) times and another simple statment. If we write down each next step in order and eliminate the same elements on different sides just like on the answer of first part of the first question, we get $T(n) = n * (2n+3)$ and it evaluates to $2n^2 + 3$. So worst case running time of this algorithm is $O(n^2)$.

3) Here is the explanation of the algorithm. Function calls itself until (n) (length of array) equals to 1, which causes function to ignore if statement and end without doing anything. Then upper function continues with (n=2). Stores the element on second index to a temp variable. Sets loop control variable to 1 lesser than (n) and shift all elements bigger than stored element to right until the loop control variable reaches first index if loop did not interrupted by the foundation of an element lesser than the stored one. When this step is done the most left (first index side) side would be sorted while other side wont. When execution jumps to upper function it takes first element of unsorted region and inserts it in the proper position of array. Continues until the end of the array to sort all of it.

Q5.

1) While limit goes infinity $f(n)/g(n) = 0$ So $f(n) = O(g(n))$

2) While limit goes infinity $f(n)/g(n) =$ infinity. So $f(n) = \Omega(g(n))$

3) While limit goes infinity $f(n)/g(n) = 0$ So $f(n) = O(g(n))$

References

- Q3) Information about squeeze theorem written with informations learned from:

  https://en.wikipedia.org/wiki/Squeeze_theorem

- Q4) Recursive insertion sort pseudeu code taken from link below:

  https://en.wikipedia.org/wiki/Insertion_sort