119188651
kshitij

Homework 1

ENPM 809X
February 9, 2023

Problem 1

1.
   - Loop invariant: All the elements on the right side of $A[j]$ are always greater than $A[j]$.
   - Verify Initialization: When $A[j] = A.length$ there are no elements on the right of $A[j]$.
   - Verify Iterations: If we assume all the values to the right of $A[j]$ are greater than $A[j]$ at the beginning of an iteration, the for loop is executed and $A[j]$ is compared with $A[j-1]$. If $A[j-1]$ is larger than $A[j]$ the elements swap positions. The for loop continues this till $A[i+1]$. In the next iteration all the values on the right side of $A[j]$ are greater than $A[j]$.
   - Termination: When $j = i+1$ all the elements to the right of $A[j]$ are greater than $A[j]$.

2.
   - Loop invariant: All the elements on the left side of $A[i]$ are always smaller than $A[i]$ and the subarray $A[1]$ to $A[i]$ is sorted.
   - Verify Initialization: When $A[i] = A[1]$ there are no elements on the right of $A[1]$ therefore the subarray from $A[1]$ to $A[1]$ is sorted.
   - Verify Iterations: If we assume all the values to the left of $A[i]$ are smaller than $A[i]$ and sorted at the beginning of an iteration, the for loop is executed and by the end of the inner for loop the smallest number is in the i^{th} position. Therefore all the numbers to the left of $A[i]$ are smaller than $A[i]$ and the subarray $A[1]$ to $A[i]$ is sorted.
   - Termination: When $i = A.length + 1$ all the elements int the array are sorted.

3.

Algorithm 1: Bubble Sort

```
for i=1 to A.length-1                #(n-1) times
  for j=A.length downto i+1          #(n-1) times
    if A[j] < A[j-1]                 #(n-1)(n-i+1) times
      exchange A[j] with A[j-1]      #(n-1)(n-i+1) times
    end
  end
end
```

Worst Case Time Complexity = $O(n^2)$

Problem 2

1. Algorithm 2: Insertion Sort (Recursive)

```
insertionSortRecursive(arr,n)
if n<=1:                          #Base Case
    return
end
insertionSortRecursive(arr,n-1) #Recursive Call
last = arr[n-1]
j = n-2

while (j>=0 and arr[j]>last):
    arr[j+1] = arr[j]
    j = j-1
end
arr[j+1]=last
```

2. The recurrence for the running time of insertion sort is

$$T(n) = \left\{ \begin{array}{ll} \Theta(1) & \text{if } n = 1, \\ \\ T(n-1) + \Theta(n) & \text{if } n > 1 \end{array} \right\}$$

3. Let $\Theta(n) = cn + k$

   $T(n) = T(n-1) + cn + k$

   $T(n-1) = T(n-2) + c(n-1) + k$

   $T(n-2) = T(n-3) + c(n-2) + k$.

   .

   .

   .

   $T(2) = T(1) + 2c + k$

   Adding all the above equations we get

   $T(n) = T(1) + c(2 + 3 + 4 + 5 + ... + n) + (n-1)k$

   but $T(1) = c + k$

   Therefore, $T(n) = c + k + c(2 + 3 + 4 + 5 + ... + n) + (n-1)k = \frac{n(n+1)}{2}c + nk$

   The time complexity of will be $\Theta(n^2)$

Problem 3

Algorithm 3: Linear time maximum subarray

```
find_maximum_subarray(arr):
    max_sum = INT_MIN
    max_left, max_right = NULL
    sum = 0
    last_left = 0
    for i from 1 to arr.length:
        sum += arr[i]
        if sum > max_sum then
            max_sum = sum
            max_left = last_left
            max_right = i
        end
        if sum < 0:
            sum = 0
            last_left = i + 1
        end
    end
    return (max_left, max_right, max_sum)
```

Problem 4

For all question $a = 2$ and $b = 4$. We know $f(n) = n^{\log_b a}$

1. $T(n) = 2T(n/4) + 1 => f(n) = 1 = n^0 = n^{\log_4 1} = n^{\log_4 2-1}$
   Since it is in the form $n^{\log_b a-\epsilon}$ it is bounded by $\Theta(\sqrt{n})$
2. $T(n) = 2T(n/4) + \sqrt{n} => f(n) = \sqrt{n} = n^{\log_4 2}$
   Since it is in the form $n^{\log_b a}$ it is bounded by $\Theta(\sqrt{n}logn)$
3. $T(n) = 2T(n/4) + n => f(n) = n = n^{\log_4 4} = n^{\log_4 2+2}$
   Since it is in the form $n^{\log_b a+\epsilon}$ it is bounded by$\Theta(n)$
4. $T(n) = 2T(n/4) + n^2 => f(n) = n^2 = n^{\log_4 1} = n^{\log_4 16-14}$
   Since it is in the form $n^{\log_b a-\epsilon}$ it is bounded by $\Theta(n^2)$