Name: Harshit Jain
Access ID: hmj5262
Recitation: 8

Problem 0

Points:

Acknowledgements

(a) I did not work in a group.

(b) I did not consult without anyone my group members.

(c) I used USTC algorithms book, chapter 9: Sorting in Linear Time as a reference.

Problem 1 | **Points:**

Solving recurrences

$$T(n) = aT(n/b) + \theta(n^d)$$

(a) Here, $d = 1.3$ and $\log_b a = \log_5 11 = 1.48$

So, $\log_b a > d$, we will use case-3 of Master's Theorem:

$T(n) = \underline{\theta(n^{\log_5 11}) = \theta(n^{1.48})}$

(b) Here, $d = 2.8$ and $\log_b a = \log_2 6 = 2.58$

So, $d > \log_b a$, we will use case-1 of Master's Theorem:

$T(n) = \underline{\theta(n^{2.8})}$

(c) Here, $d = 0$ and $\log_b a = \log_3 5 = 1.46$

So, $\log_b a > d$, we will use case-3 of Master's Theorem:

$T(n) = \underline{\theta(n^{\log_3 5}) = \theta(n^{1.46})}$

(d) $T(n) = T(n-2) + \log(n)$

$= [T(n-4) + \log(n-2)] + \log(n)$

$= [T(n-6) + \log(n-4)] + \log(n-2) + \log(n)$

$\vdots$

$= [T(n-2k) + \log(n-(2k-2))] + \cdots + \log(n-2) + \log(n)$

Let $2k = n$,

$= T(0) + \log(2) + \log(4) + \log(6) + \cdots + \log(n)$

$= 1 + \log(2 \cdot 4 \cdot 6 \cdot \cdots \cdot n)$

$= 1 + \log(2^{n/2} \cdot (1 \cdot 2 \cdot 3 \cdot \cdots \cdot n/2))$

$= 1 + (n/2)(\log(2)) + \log((n/2)!)$

$\Rightarrow \log(1) + \cdots + \log(1) < \log(1) + \log(2) + \cdots + \log(n/2) < \log(n) + \cdots + \log(n)$

$\Rightarrow 0 < \log((n/2)!) < \log((n/2)^{n/2})$

Also, $\log((n/2)^{n/2}) = (n/2) \cdot \log(n/2) = \frac{n \cdot \log_c(n)}{2 \cdot \log_c(2)} = O(n\log(n))$

$= \underline{O(n\log(n))}$

Problem 2

Points:

Sorted Array

```
def Search(low, high, A):
if(low == high):
    if(A[low] == low):
        return low
    else:
        return False
else:
    mid = (low+high)//2
    if(A[mid] == mid):
        return mid
    elif(A[mid] > mid):
        return Search(low, mid-1, A)
    else:
        return Search(mid+1, high, A)
```

Run-time Analysis: $\underline{O(\log(n))}$

Problem 3

Points:

Linear Time Sorting

```python
def linearTimeSorting(unsortedList, k):
    sortedList = [0 for _ in range(len(unsortedList))]
    tempList = [0 for _ in range(k+1)]

    for j in range(0, len(unsortedList)):
        tempList[unsortedList[j]] += 1

    for i in range(1, k+1):
        tempList[i] = tempList[i] + tempList[i-1]

    for j in range(len(unsortedList)-1, -1, -1):
        sortedList[tempList[unsortedList[j]]-1] = unsortedList[j]
        tempList[unsortedList[j]] -= 1

    return sortedList
```

Initially, the *tempList* is initialized and it takes $O(k)$ time where k is $\underset{i}{max}(x_i)$.

The first 'for loop' is making $tempList[i]$ to hold the number of input elements equal to i for each integer $i = 1, 2, \cdots, k \Rightarrow$ frequency of elements. It takes $O(n)$ time.

The second 'for loop' is keeping running sum of array *tempList* to get how many input elements $\leq i$. Tt takes $O(k)$ time.

The third 'for loop' placing each element $unsortedList[j]$ in its correct sorted position in the $sortedList$; also we decrement $tempList[unsortedList[j]]$ each time we place a value $unsortedList[j]$ into the $sortedList$. It takes $O(n)$ time.

Therefore, the time is $O(k+n)$. Note that, there is a significance of $\underset{i}{min}(x_i)$ that the *tempList* starts changing from the index $= \underset{i}{min}(x_i)$ and not from index 0 everytime. This is because the given list may or may not have the minimum integer as 0. Depending on what minimum number the list contains, the *tempList* is starting to change from that index instead of 0.

Therefore, the **Overall Time Complexity is:** $O(n + \underset{i}{max}(x_i) - \underset{i}{min}(x_i))$

$$\Rightarrow \underline{O(n+M)} \text{ (where } M = \underset{i}{max}(x_i) - \underset{i}{min}(x_i))$$

The $\Omega(n \cdot \log(n))$ does not apply in this case because it is not a comparison sort. There is no comparison among input elements in this algorithm.