CS 374 HW 4 Problem 2

Aldo Sanjoto, Hieu Huynh, quddus2

TOTAL POINTS

15 / 100

QUESTION 1

1 2A **10 / 20**

+ **20 pts** Correct

+ **5 pts** IDK

+ **0 pts** Solution unreadable, too long or too complicated

✓ + **10 pts** **Solution slower than O(kn), but still right**

- **5 pts** Minor errors or unimportant typos

+ **0 pts** Major mistake

- **5 pts** runtime not argued

+ **10 pts** Argue with average time

This is O(n^2+k).

QUESTION 2

2 2B **0 / 60**

+ **60 pts** Correct

+ **15 pts** IDK

✓ + **0 pts** **Solution illegible, impossible to follow, or algorithmic idea not on the right track**

+ **40 pts** Correct algorithm

+ **10 pts** Apply the idea of splitting the array B of ranks into two parts by the median index

+ **10 pts** Correctly partition A into two parts with values less than and greater A[i_mid] using O(n) time

+ **20 pts** Correct recursive calls on each part of A with the appropriate part of ranks and rest of the details

- **10 pts** Minor Flaw in algorithm design

+ **20 pts** Correct running time analysis

+ **10 pts** Correct recursive function of running time

+ **10 pts** Correct solution to recursion

- **5 pts** Minor flaws in computation of recursion

+ **10 pts** recursion formula without explicit running time

+ **5 pts** Give a running time without justification

+ **10 pts** justify running time without recursive formula

QUESTION 3

3 2C **5 / 20**

+ **20 pts** Correct

✓ + **5 pts** **IDK**

+ **0 pts** Solution illegible, or impossible to follow

+ **10 pts** Correct reduction: noting that running with n=k & B=[1,2,...n] and output in increasing order of ranks simply outputs a sorted array

+ **10 pts** Noting that fastest sorting by comparison takes O(n log n) and prove by contradiction

- **5 pts** minor bug

+ **0 pts** wrong

2a)

```
int[ ] compute_k_elems(A[1,2,...,n ], B[1,2...k]){
        c[B[k]]; // arrays used to contain k smallest numbers
        for( i = 0; i < B[k]; i++){                    // loop that runs k time where k is i_k
                for(j = 1; j<n; j++){                  // loop that runs n times
                        if(A[j] < A[i]){
                                swap(A[i], A[j])
                        }
                }
                c[i] = A[i]
        }
        result[];
        for(i = 0; i < len(B); i++){
                Result[i] = C[B[i] - 1];
        Return result;
}
```

Explanation:

In our algorithm, we do a bubble sort to get the first k smallest values in order. From that point we run another loop which is O(length of array B) to print out the ranks in order from $i_0 \ldots i_k$. The following algorithm gives us the running time of O(nk)+O(length of B) => O(nk)

1 2A 10 / 20

+ 20 pts Correct

+ 5 pts IDK

+ 0 pts Solution unreadable, too long or too complicated

✓ + 10 pts **Solution slower than O(kn), but still right**

- 5 pts Minor errors or unimportant typos

+ 0 pts Major mistake

- 5 pts runtime not argued

+ 10 pts Argue with average time

This is O(n^2+k).

2b)

```
Int[] compute_k_elems(A[1,2….n], B[1,2,..k]){
        m = ceil(n/k)
        Break A into m arrays each has size k: A1,A2,...,Am
        For i ← 1 to m:
                MergeSort(Ai)
        Let idx_i be index point to first element in array Ai
        We have m arrays, so we have m indexes point to m arrays
        int B_k_smallest_vals[B[k]]
        int count = 0;
        while(count < B[k]){
                Find idx_i of array Ai such that Ai[idx_i] is the smallest value
                amongA1[idx_1],A2[idx_2], … Am[idx_m]
                Add A_i[idx_i] into B_k_smallest_vals array
                Increment idx_i
                Count++;
        }
        For i ← 0 to k:
                result[i] =  B_k_smallest_vals array[B[i]]
        Return result;
}
```

Running time Analysis:

To sort the one array of size k using MergeSort, it takes O(klog(k)). To sort n/k arrays each has size k takes O((n/k) * klog(k)) = O(nlog(k)).

The while() loop stop after k times. Each round take O(n/k) time to find the minimum among sub-arrays. Therefore, while() loop takes O(k*n/k) = O(n).

Therefore, the total running time of the algorithm is:

$$T(n) = O(n\log(k)) + O(n)$$

$$\Rightarrow \mathbf{T(n) = O(n\log(k))}$$

2 2B 0 / 60

+ 60 pts Correct

+ 15 pts IDK

✓ + 0 pts **Solution illegible, impossible to follow, or algorithmic idea not on the right track**

+ 40 pts Correct algorithm

+ 10 pts Apply the idea of splitting the array B of ranks into two parts by the median index

+ 10 pts Correctly partition A into two parts with values less than and greater A[i_mid] using O(n) time

+ 20 pts Correct recursive calls on each part of A with the appropriate part of ranks and rest of the details

- 10 pts Minor Flaw in algorithm design

+ 20 pts Correct running time analysis

+ 10 pts Correct recursive function of running time

+ 10 pts Correct solution to recursion

- 5 pts Minor flaws in computation of recursion

+ 10 pts recursion formula without explicit running time

+ 5 pts Give a running time without justification

+ 10 pts justify running time without recursive formula

Question 2C.

IDK

3 2C 5 / 20

+ 20 pts Correct

✓ + 5 pts IDK

+ 0 pts Solution illegible, or impossible to follow

+ 10 pts Correct reduction: noting that running with n=k & B=[1,2,...n] and output in increasing order of ranks simply outputs a sorted array

+ 10 pts Noting that fastest sorting by comparison takes O(n log n) and prove by contradiction

- 5 pts minor bug

+ 0 pts wrong