CS 374 HW 4 Problem 3

quddus2, Aldo Sanjoto, Hieu Huynh

TOTAL POINTS

90 / 100

QUESTION 1

1 3A **20 / 20**

✓ - **0 pts** **Correct**

- **20 pts** Using more than O(log n) calls to isGood.
- **10 pts** Insufficient detail
- **15 pts** IDK

QUESTION 2

2 3B **40 / 40**

✓ - **0 pts** **Correct**

- **10 pts** Using quickselect instead of select
- **40 pts** Runtime > O(n)
- **20 pts** Using more than log(n) calls to isGood.
- **30 pts** IDK
- **10 pts** No/wrong running time analysis
- **5 pts** Minor mistake

QUESTION 3

3 3C **30 / 40**

- **0 pts** Correct
- **40 pts** Using more than O(log n) to isGood than log n
- **5 pts** Runtime > O(n log k)
- **40 pts** Incorrect algorithm
- **30 pts** IDK

✓ - **10 pts** **Number of calls to isGood isn't minimal, which is O(log k n)**

- **10 pts** Missing or incorrect time analysis

It is possible to achieve O(log k n) instead of O(log 2 n/k) number of calls to isGood. See solution. (log k n = (log k 2)(log 2 n)   log 2 n/k = log 2 n - log 2 k   log k n is a better running time if you consider k a variable >> 2)

Question 3a.

```
int[] compute_all_Good_number (A[0…n-1]){
	if(A.empty())
		return int[];
	else if(A.size() == 1){
		if(isGood(A[0])
			return A[0];
		else
			return int[];
	}
	Mergesort(A[0…n-1]);
	left = 0;
	right = n-1;
	while(left < right){
		mid = floor((left + right)/2);
		if(isGood(A[mid])
			left = mid+1;
		else
			right = mid;
	}
	if(left >0)
		return A[0,.. left-1];
	else
		return int[];
}
```

The number of call to isGood() is O(logn).
The running time is O(nlog(n))

1 3A 20 / 20

✓ - 0 pts Correct

- 20 pts Using more than O(log n) calls to isGood.

- 10 pts Insufficient detail

- 15 pts IDK

Q3B

```
//Find the greatest number that is good in array A. If there is no good number,
return −∞ int find_largest_Good_val(A[0,...n-1]){
        if(A.size() == 0)
                return −∞;
        if(A.size() < 5):                    //n-1 <5
                sort(A)                      //return the largest "good" value
                for i ← n-1 to 0:
                        if(isGood(A[i]):
                                return A[i]
                return −∞
        Form lists L_1, L_2,... L_⌈n/5⌉ where L_i = {A[5i-4], ... A[5i]}
        Find median b_i of L_i using brute-force
        B = [b_1, b_2, ... b_⌈n/5⌉]
        b = find_largest_Good_val(B[b_1, b_2, ... b_⌈n/5⌉])
        if(b == −∞):
                b_min = min (b_1, b_2, ... b_⌈n/5⌉)
                for i ← 0 to n-1:            //Find all elements in A that is smaller than b_min
                        if(A[i] < b_min)
                                A_less.add(A[i]);
                return find_largest_Good_vals(A_less);
        else:
                if (b == max (b1, b2, .. b_⌈n/5⌉):
                        for i ← 0 to n-1:            //Find all elements in A that is greater than b
                                if(A[i] > b)
                                        A_greater.add(A[i]);
                        return max(b, find_largest_Good_vals(A_greater));
                else :
                        b_next = the smallest number in B that is greater than b
                        for i ← 0 to n-1:
                                if(A[i] > b && A[i] < b_next )
                                        A_greater.add(A[i]);
                        return max(b, find_largest_Good_vals(A_greater));
}
//Find all good number in array A
int[] compute_all_Good_number(A[0...n-1]){
        b = find_largest_Good_val(A[0,..n-1])
        int result[];
        for i ← 0 to n-1:
                if(A[i] <= b)
                        result.add(A[i];
        return result;
}
```

Running time analysis:

Let $T_1(n)$ be the running time of function *find_largest_Good_val*.

In function *find_largest_Good_val* with array of size n, we recurse twice, one with array B with size $\lceil n/5 \rceil$, and one with array size either $|A_{greater}|\ or\ |A_{less}|$. Also the running time to form the lists L_i, to find the medians b_i, and to find all elements that is greater or smaller than b is $O(n)$. Therefore, we have running time $T_1(n)$:

$$T_1(n) \leq T_1\left(\left\lceil \frac{n}{5} \right\rceil\right) + \max\{T(|A_{greater}|), T(|A_{less}|)\} + O(n)$$

Base case: $T_1(n) = O(1)$ with n < 6 because we only sort a constant size array, and apply isGood function constant time

Claim: $\max\{T(|A_{greater}|), T(|A_{less}|)\} \leq T(\frac{2*n}{5} + 2)$

Prove:

Case 1: $b == -\infty$: (No good number in array B):

Because b_i is the median of L_i and L_i has 5 elements→ there are 3 numbers in L_i that are greater or equal than b_i.

b_{min} is the minimum in array B → $b_{min} \geq b_i\ for\ all\ b_i\ in\ B$

→In any L_i, there are at least 3 numbers greater or equal b_{min}

We also have that the array A is partition into $\lceil n/5 \rceil$ lists.

→ In array A, there are at least $3 * \lceil n/5 \rceil$ numbers that are greater or equal b_{min}

A_{less} is the array of numbers in A that are less than b_{min}.

→ $|A_{less}| \leq n - \left(3 * \left\lceil \frac{n}{5} \right\rceil\right) \leq n - 3 * \left(\frac{n}{5} + 1\right) = \frac{2*n}{5} - 3 \leq \frac{2*n}{5}$

Therefore $|A_{less}| \leq \frac{2*n}{5}$

Case 2: b > $-\infty$: (There is at least 1 good number in array B)

Case a: b == b_{max}: b is the maximum number in array B

Because b_i is the median of L_i and L_i has 5 elements→ there are at most 2 numbers in L_i that are greater than b_i.

→In any L_i, there are at most 2 numbers greater than b_{max}

We also have that the array A is partition into $\lceil n/5 \rceil$ lists.

→ In array A, there are at most $2 * \lceil n/5 \rceil$ numbers that are greater than b_{max}

$A_{greater}$ is the array of numbers in A that are greater than b_{max}.

→ $|A_{greater}| \leq \left(2 * \left\lceil \frac{n}{5} \right\rceil\right) \leq \frac{2*n}{5} + 2$

Therefore, for this case ,we have $|A_{greater}| \leq \frac{2*n}{5} + 2$

Case b: b < b_{max}: b is not the maximum number in array B.

Let b_{next} be the smallest number in B that is greater than b.

For each $b_i$in B, there're two possibility, either $b_i \leq b$ or $b_i > b \Leftrightarrow b_i \geq b_{next}$

Because b_i is the median of L_i and L_i has 5 elements→ there are at most 2 numbers in L_i that are greater than b and smaller than b_{next}.

We also have that the array A is partition into $\lceil n/5 \rceil$ lists.

→ In array A, there are at most $2 * \lceil n/5 \rceil$ numbers that are smaller than b_{next} and greater than b

$A_{greater}$ is the array of numbers in A that are smaller than b_{next} and greater than b

→ $|A_{greater}| \leq \left(2 * \left\lceil\frac{n}{5}\right\rceil\right) \leq \frac{2*n}{5} + 2$

Therefore, for this case ,we have $|A_{greater}| \leq \frac{2*n}{5} + 2$

Therefore, for any cases, we have

$$\max\{|\ A_{greater}\ |, |A_{less}|\} \leq \frac{2*n}{5} + 2$$

$$\rightarrow \max\{T(|A_{greater}|), T(|A_{less}|)\} \leq T(\frac{2*n}{5} + 2)$$

Therefore,

$$T_1(n) \leq T_1\left(\left\lceil\frac{n}{5}\right\rceil\right) + T_1(\frac{2*n}{5} + 2) + O(n)$$

Ignore all constant, we have:

$$T_1(n) = T_1\left(\frac{n}{5}\right) + T_1\left(\frac{2*n}{5}\right) + O(n)$$

Solve $T_1(n)$:

Assume that $T_1(n) < C * n$. We prove that there exist constant C.

$T_1(n) = T_1\left(\frac{n}{5}\right) + T_1\left(\frac{2*n}{5}\right) + a * n$ with a be a constant

$C * n \geq T_1\left(\frac{n}{5}\right) + T_1\left(\frac{2*n}{5}\right) + a * n$

$C * n \geq C * \frac{n}{5} + C * \frac{2*n}{5} + a * n = n * \left(\frac{3}{5} * C + a\right)$

$C \geq \left(\frac{3}{5} * C + a\right)$

$C \geq \frac{5*a}{2}$

There is such constant exists. So, $T_1(n)$ = O(n)

Therefore, the running time of *find_largest_Good_val()* is O(n).

The running time of function compute_all_isGood equals the running time of find_largest_Good_val() plus the running time of extracting all elements in A that is smaller or equal the return value of find_largest_Good_val(), which is O(n). Therefore, the total running time of the algorithm is O(n)

Analysis the total number of calls to isGood:

Let $T_2(n)$ be the total number of calls to isGood in function *find_largest_isGood_val()*.

$$T_2(n) = T_2\left(\frac{n}{5}\right) + T_2\left(2 * \frac{n}{5}\right) + O(1) = T_2\left(3 * \frac{n}{5}\right) + O(1)$$

Using the recursion tree, at Level i we have:

Number of sub-problem: 1

Work done by all sub-problems: O(1).

And the tree has $O\big(log(n)\big)$ levels. Therefore, $T_2(n) = \mathrm{O}(\log(\mathrm{n}))$.

Total number of calls to *isGood* is the number of calls to *isGood* that is used in *find_largest_isGood_val()* because the function *compute_all_isGood()* does not call *isGood*. Therefore, the total number of calls to isGood is O(log(n)).

2 3B 40 / 40

- ✓ - **0 pts** Correct
- - **10 pts** Using quickselect instead of select
- - **40 pts** Runtime > O(n)
- - **20 pts** Using more than log(n) calls to isGood.
- - **30 pts** IDK
- - **10 pts** No/wrong running time analysis
- - **5 pts** Minor mistake

Question 3 C

```
int[] Compute_all_good_val(A[0,1,2,…n-1]){
        int result[];                //Array to store all good values
        Break A into k arrays A_0, A_2,.., A_{k-1} each of size ⌈n/k⌉
        for i ← 0 to k-1:
                Mergesort(A_i)
        int left[k];
        int right[k];
        int mid[k];
        bool directions[k];
        while(left[i] < right[i] for some i in range [0,k-1]){
                for i← 0 to k-1:
                        if left[i] < right[i]:
                                mid[i] = ⌊(left[i]+right[i])/2⌋
                directions = isGood([ A_0[mid[0], A_1[mid[1], A_2[mid[2], …., A_{k-1}[mid[k-1] ]);
                for i ← 0 to k-1:
                        if(left[i] < right[i]):
                                if(direction[i] == true):          //A[mid] is good
                                        left = mid +1;
                                else:                              //A[mid] is not good
                                        right = mid;
        }
        for i ← 0 to k-1:
                for j ← 0 to left[i]-1:
                        result.add(A_i[j])
        return result;
}
```

Running time analysis:

To sort one array of size n/k, it takes O($\frac{n}{k} * \log\left(\frac{n}{k}\right)$). Therefore, to sort k arrays each has size n/k, it takes O($k * \frac{n}{k} log(\frac{n}{k})) = O(n \log(\frac{n}{k}))$.

Running time of the while() loop:

For each round, each array A_i, we either choose to consider the left half or the right half of that array based on the output of isGood function. Therefore, the size of each array A_i is decrease by half after each round. Therefore, after each round, the total number of elements in k arrays decrease by half. Let T(m), with m be the size of array each arrays A_i , be the number of rounds of the while loop.

$$T(m) = T\left(\frac{m}{2}\right) + O(1)$$

$$T(1) = O(1)$$

Using the recursion tree, we have at level i, there is 1 sub-problem, and it takes O(1) time. And the tree has height log(m). Therefore the total number of rounds is:

$$T(n) = O(\log(m))$$

In each round, we have k arrays, and each of them takes O(1) time. Therefore, each round take O(k) running time.

Therefore, the total running time of the while() loop is $O(k * \log(n/k))$

Therefore, the total running time for the whole algorithm is

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

Number of calls to isGood analysis:

We only call to function isGood once every round of the while() loop. As proven above, the number of round until the while() loop is finish is $O(\log(m))$, with m be the size of each sub-array A_i. And because we divide the array of size n into k arrays of size $\lceil n/k \rceil$. Therefore, m = $\lceil n/k \rceil$ → The number of rounds is $O(log\left(\left\lceil\frac{n}{k}\right\rceil\right))$

Therefore, the number of calls to isGood is $O(log\left(\left\lceil\frac{n}{k}\right\rceil\right))$

3 3C 30 / 40

- **- 0 pts** Correct
- **- 40 pts** Using more than O(log n) to isGood than log n
- **- 5 pts** Runtime > O(n log k)
- **- 40 pts** Incorrect algorithm
- **- 30 pts** IDK
- ✓ **- 10 pts Number of calls to isGood isn't minimal, which is O(log k n)**
- **- 10 pts** Missing or incorrect time analysis

It is possible to achieve O(log k n) instead of O(log 2 n/k) number of calls to isGood. See solution. (log k n = (log k 2)(log 2 n) log 2 n/k = log 2 n - log 2 k log k n is a better running time if you consider k a variable >> 2)