Homework 3

(Due: Dec 1)

GROUP NUMBER: 54

Group Members		
Name	SBU ID	% Contribution
Sharad Sridhar	111492675	33
Nihal Harish	111498545	33
Raunak Shah	111511721	33

Collaborating Groups

Group Number	Specifics (e.g., specific group member? specific task/subtask?)
41	Some questions' sub parts were done in collaboration with members of another group
76	Question 2 was done in collaboration
35	Question 2 was done in collaboration
27	Question 3 was done in collaboration

External Resources Used

	Specifics (e.g., cite papers, webpages, etc. and mention why each was used)
1.	Introduction to Algorithms (general study)
2.	Exam papers from previous years
3.	
4.	
5.	

Task 1

Solution for part a

- For starters, we assume that the heap being used is a min-heap. Now, each root is the lowest element in the heap. To get the k-th smallest element, we have to perform k extract-min operations on the binary heap.

- Now each extract min operation is done as follows: the root is swapped with the lower-most element in the heap in order to "remove/extract" it. Then, we have to balance the tree again so that the smallest element is the root again.

- That operation would require at max a traversal of the tree from root to a leaf. This is clearly an O(log n) operation.

- Then, we have to "insert" the extracted elements back into the heap, since we are not asked to delete the element from the tree. The insertion causes another imbalance in the heap. This can be again balanced as mentioned before in O(log(n)).

- For k extract-min operations, we see that the total running time is $k * log(n)$

- This statement is executed when k's value is quite close to $\frac{n}{log(n)}$

- Substituting the value of k to the running time we get : $O(\frac{n*log(n)}{log(n))} = O(n)$ running time.

Which is the required result

Solution for part b

- Let the random variable X_i denote the probability of an element i of the array getting selected.

- We say that $X_i = 1$ when the element is selected And $X_i = 0$ when the element is not selected

- We now calculate the expected value of X, $E[X]$ as follows

$E[X_i] = 1 * Probability[X_i = 1] + 0 * Probability[X_i = 0]$

Any element from the array is equally likely to get selected. This gives us (for an array of size n) a probability $\frac{1}{n}$ that an element is selected.

- We see that $E[X_i] = \frac{1}{n}$

- Also, $E[X] = \sum_{i=1}^{i=\lceil log^2(n) \rceil} X_i = \frac{\lceil log^2(n) \rceil}{n}$

- We are asked to show that no element in the array is chosen randomly more than once. We can view this as follows:

- $P(X < 2) = 1 - P(X \geq 2)$

- Now, from Markov's Inequality: $P(X \geq \delta) \leq \frac{E[X]}{\delta}$

Using $\delta = 2$, we have:

$P(X \geq 2) \leq \frac{E[x]}{2}$

$\longrightarrow P(X \geq 2) \leq \frac{\frac{\lceil log^2(n) \rceil}{n}}{2}$

$\longrightarrow P(X \geq 2) \leq \frac{\lceil log^2(n) \rceil}{2n}$

$\longrightarrow P(X < 2) \leq 1 - \frac{\lceil log^2(n) \rceil}{2n}$

This value approaches closer to 1 as n increases. The numerator in the fraction is a logarithmic function of n, whereas the denominator is linear function. As n increases exponentially, we see that the denominator increases in value faster than the numerator. For extremely large values of n, the fraction becomes nearly 0. This shows us that the relation shown above will hold true with high probability.

Solution for part c

- Let X_i be a uniform random variable which is defined as follows:

$log^2 n$ elements is the size from which we select as is done in the algorithm.

$X_i = 1$ when the element i is selected from $log^2 n$ elements

$X_i = 0$ when the element i is not selected

$P(X_i = 1) = \frac{1}{(log)^2(n)}$

Now, expected value : $E[X_i] = 1 * Probability[X_i = 1] + 0 * Probability[X_i = 0] = \frac{1}{(log)^2(n)}$

Also, $E[X] = \sum_{i=1}^{i=n} \frac{1}{(*log)^2(n)} = \frac{n}{(log)^2(n)}$

Now $E[X] = \mu = \frac{n}{(log)^2(n)}$

Using Chernoff's bound : $P(X \geq [1+\delta]\mu) \leq \frac{e^{\delta\mu}}{(1+\delta)^{(1+\delta)\mu}}$

Let $(1+\delta)\mu = \frac{n}{logn}$

$\longrightarrow (1+\delta)\frac{n}{log^2 n} = \frac{n}{logn}$

$\longrightarrow (1+\delta)\frac{1}{logn} = 1$

$\longrightarrow \delta = log(n) - 1$

Now, using the bound and substituting the values of δ and μ

$P(X \geq [1+\delta]\mu) \leq \frac{e^{(log(n)-1)\frac{n}{log^2 n}}}{(log(n))^{(log(n))(\frac{n}{log^2 n})}}$

We can observe that this value converges to 0 for higher values of n.

Now this probability is the probability of the number being greater than the given rank. The required probability is 1-given probability (say, P(x)).

This shows us that the required probability converges to 1 as the value of n increases. Thus it is proved with high probability that $|T| < O(\frac{n}{logn})$

Solution for part d

- We can find the upper bound of the running time by analyzing individual sections of the code.

- **Line 2** - The running time is given as O(n)

- **Line 5** - Assuming that each element can be accessed in constant time, the complexity is $O(\lceil log^2(n) \rceil)$

- **Line 6** - The removal of duplicate elements will take max $O(\lceil log^2(n) \rceil)$ time

- **Line 8** - A sorting operation can be done in $O(nlog(n)$ time. Here n can be replaced by $O(\lceil log^2(n) \rceil)$. This is the maximum elements that will be present in the list S (this is the case when there are no duplicate elements). Now we get the complexity as : $O(\lceil log^2(n) \rceil) log((\lceil log^2(n) \rceil))$

- **Line 10 - 12** - The for loop runs for n iterations. Within each loop, each of the $q = O(\lceil log^2(n) \rceil)$ bins will be visited to determine the location of $A[j]$. Also, the ideal search would be a binary search that runs in log(n) time (The input for search is sorted). This gives us a complexity of $O(n * log(\lceil log^2(n) \rceil))$

- **Line 13** - This will take O(n) time

- **Line 15-16** - The for loop runs in O(n) time

and finally,

- **Line 17** - The time complexity here is used from the results of the previous question. This is $O(\frac{n}{log(n)})$

- Combining all these complexities from the algorithm, we see that the bound for complexity is $O(n * log((\lceil log^2(n) \rceil))$. This can also be written as :$O(nlog((log^2(n)))$

- We have seen that the bound in lines 10 -12 hold with hight probability in n. We can extend it to say that, in a similar way, the bound also holds for the worst case running time of the algorithm with high probability.

Solution for part e

- Condition 1 is where the first IF condition is executed. We see that there is absolutely no randomization occurring. The steps are deterministic and should cover all cases pertaining to the given conditions. Thus the algorithm does not fail for the first condition.

- Condition 2 covers all other conditions. Here we have to take into account that there is a portion of code that relies on randomization. This could cause a situation where the actual result might not be in the selected subset. When the search algorithm is run on this subset, we could end up with receiving nothing, even though the initial input contains the correct result. The randomization might fail. We can take a closer look at the 2nd portion of the code.

- Line 5 selects a subset of the given array with some probability. This results in a case where some elements are left out. Now the k^{th} element could be absent from the chosen subset. This could result in a problem if not fixed.

- Line 7 is important because it sets $s_0 = -\infty$ and $s_{q+1} = \infty$.

- In the next lines, we see the creation of bins according to the given limits. Now, with the chosen subset of elements, we can end up leaving out some limits, and subsequently, some elements. But line 7 fixes the problem. We see that the elements that weren't selected are included in the first and last bins when the bins get searched according to the limits of the bin(s). The bins are checked and all elements get sorted into their respective bins. The corresponding c_i is updated to reflect the relation of the element to the bin B_i.

- The first bin and last bin correspond to the numbers lower and higher than the ones in the selected subset respectively.

- The next few lines of the algorithm are completely deterministic. There is no random component to consider. We can see that all possible cases are covered, without any possibility of leaving any number out.

- Hence, we can conclude that the algorithm not only returns a result every time, but also that the algorithm produces the correct result since all numbers correspond to some bin.
