

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
- $$E[X_i] = 1 * \text{Probability}[X_i = 1] + 0 * \text{Probability}[X_i = 0]$$
- 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:
- $$P(X \geq 2) \leq \frac{E[x]}{2}$$

$$\begin{aligned} \longrightarrow P(X \geq 2) &\leq \frac{\frac{[\log^2(n)]}{2}}{n} \\ \longrightarrow P(X \geq 2) &\leq \frac{[\log^2(n)]}{2n} \\ \longrightarrow P(X < 2) &\leq 1 - \frac{[\log^2(n)]}{2n} \end{aligned}$$

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

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

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

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

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

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

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

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

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

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

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 $|T| < O(\frac{n}{\log n})$

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 $O(n \log(n))$ 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 : $O(n \log(\log^2(n)))$
 - We have seen that the bound in lines 10 -12 hold with high 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.
-

Task 2

2.1 Question a

Prove that after the loops in steps 11-12 terminates in $|A'| = \theta\left(\frac{n}{\log \log n}\right)$ holds w.h.p. in n .

Solution:

Let $X_i = \begin{cases} 1 & \text{if the } i\text{th element is added to } A_i \\ 0 & \text{otherwise} \end{cases}$

$$E[X_i] = \Pr[X_i = 1] = 1/\log \log n$$

$$\therefore E[X] = \sum_{i=0}^n E[X_i] = n/\log \log n$$

$$\mu = E[X] = n/\log \log n$$

Take $\gamma = c * \frac{n}{\log \log n}$, where c is any constant $0 < c < 1$
 $\therefore 0 < \gamma < \mu$

WKT

$$\Pr[X \leq \gamma + \mu] = e^{-\gamma^2/3\mu}$$

$$\Pr[X \leq (c + 1)n/\log \log n] = e^{-(cn/\log \log n)^2/3(c/\log \log n)}$$

$$\Pr[X \leq (c + 1)n/\log \log n] = e^{-(cn/\log \log n)/3}$$

$$\Pr[X \leq (c + 1)n/\log \log n] = 1/e^{(cn/\log \log n)/3}$$

$$\therefore \Pr[X \geq (c + 1)n/\log \log n] = 1 - 1/e^{(cn/\log \log n)/3}$$

thus w.h.p in n , $|A'| = \theta(n/\log \log n)$

2.2 Question b

Let $\Delta = k_{right} - k_{left}$. Show that w.h.p in n , $2\sqrt{m \log n} \leq \Delta \leq 2\sqrt{2m \log n}$

Solution:

Given:

$$\begin{aligned}k_{left} &= k' - 2\sqrt{(k' \log n)} \\k_{right} &= k' + 2\sqrt{((m - k') \log n)} \\ \Delta = k_{right} - k_{left} &= 2\sqrt{((m - k') \log n)} + 2\sqrt{(k' \log n)} \\ \Delta &= 2\sqrt{\log n}(\sqrt{(m - k')} + \sqrt{(k')})\end{aligned}$$

$$\begin{aligned}\Delta &= 2\sqrt{\log n}(\sqrt{(m - k')} + \sqrt{(k')}) \\ \text{WKT } k' &= m * \frac{k}{n}\end{aligned}$$

$$\begin{aligned}\Delta &= 2\sqrt{\log n}(\sqrt{(m - (m * \frac{k}{n}))} + \sqrt{(m * \frac{k}{n})}) \\ \Delta &= 2\sqrt{m \log n}(\sqrt{(1 - (\frac{k}{n}))} + \sqrt{(\frac{k}{n})})\end{aligned}$$

Also,

$$m = n / \log \log n$$

We have to prove

$$2\sqrt{m \log n} \leq \Delta \leq 2\sqrt{2m \log n}$$

Differentiating Δ , we get

$$\frac{d \Delta}{dx} = \frac{1}{2}(\sqrt{\frac{n}{k}} - \sqrt{\frac{n}{n-k}})$$

Equating $\frac{d \Delta}{dx}$ to 0,

$$\begin{aligned}\frac{1}{2} \left(\sqrt{\frac{n}{k}} - \sqrt{\frac{n}{n-k}} \right) &= 0 \\ \therefore k &= \frac{n}{2}\end{aligned}$$

At most,

$$\Delta = 2\sqrt{2m \log n}$$

Also, the value of Δ decreases either side of $n/2$. We can find the smallest value of Δ at $k = n$.

At $k = n$,

$$\begin{aligned}\Delta &= 2\sqrt{m \log n} \\ \therefore 2\sqrt{m \log n} &\leq \Delta \leq 2\sqrt{2m \log n}\end{aligned}$$

To prove w.h.p, we use Chernoffs bounds.

$$\begin{aligned}E[\Delta] &= 2\sqrt{m \log n} \sqrt{1 - \left(\frac{k}{n}\right)} + \sqrt{\frac{k}{n}} = \mu \\ \text{Let } x &= \sqrt{1 - \left(\frac{k}{n}\right)} + \sqrt{\frac{k}{n}}\end{aligned}$$

$$E[\Delta] = 2x\sqrt{m \log n} = \mu$$

Taking the chernoff bound

$$\Pr[X \geq (1 + \partial)\mu] \leq e^{-\mu\partial^2/3}$$

We have

$$\begin{aligned}(1 + \partial)\mu &= 2\sqrt{(2m \log n)} \\ \partial &= \frac{\sqrt{2}}{x} - 1\end{aligned}$$

$$\Pr[X \geq 2\sqrt{(2m \log n)}] \leq e^{-2x\sqrt{m \log n} * (\frac{\sqrt{2}}{x} - 1)^2/3}$$

Solving,

$$\Pr[X \geq 2\sqrt{(2m \log n)}] \leq e^{-\left(\frac{4}{x} - 2x - 4\sqrt{2}\right) * \sqrt{m \log n}/3}$$

$$\Pr[X \geq 2\sqrt{(2m \log n)}] \leq e^{-c * \sqrt{m \log n}}$$

where $c = \left(\frac{4}{x} - 2x - 4\sqrt{2}\right)/3$

$$\Pr[X \geq 2\sqrt{(2m \log n)}] \leq 1/e^{c * \sqrt{m \log n}}$$

$$\Pr[X \leq 2\sqrt{(2m \log n)}] \geq 1 - 1/e^{c * \sqrt{m \log n}}$$

$$\Pr[X \leq 2\sqrt{(2m \log n)}] \geq 1 - 1/e^{c * \sqrt{(n/\log \log n) * \log n}}$$

therefore w.h.p in n , $\Delta \leq 2\sqrt{2m \log n}$

For the lower bound,

$$\begin{aligned}\Pr[X \leq (1 - \partial)\mu] &\leq e^{-\mu\delta^2/2} \\ \Pr[\Delta \leq 2\sqrt{m \log n}] &\leq e^{-\mu\delta^2/2}\end{aligned}$$

Again,

$$\begin{aligned}\mu &= 2x\sqrt{m \log n} \\ (1 - \partial)\mu &= 2\sqrt{m \log n} \\ \therefore \partial &= 1 - \frac{1}{x}\end{aligned}$$

$$\Pr[\Delta \leq 2\sqrt{m \log n}] \leq e^{-2x\sqrt{m \log n}(1 - \frac{1}{x})^2/2}$$

$$\Pr[\Delta \leq 2\sqrt{m \log n}] \leq e^{-c\sqrt{m \log n}/2}$$

where $c = 2x(1 - \frac{1}{x})^2$

$$\begin{aligned}\Pr[\Delta \leq 2\sqrt{m \log n}] &\leq 1/e^{c\sqrt{m \log n}/2} \\ \Pr[\Delta \leq 2\sqrt{m \log n}] &\leq 1/e^{c\sqrt{(n/\log \log n) * \log n}/2} \\ \Pr[\Delta \geq 2\sqrt{m \log n}] &\geq 1 - 1/e^{c\sqrt{(n/\log \log n) * \log n}/2}\end{aligned}$$

Therefore w.h.p in n , $\Delta \leq 2\sqrt{2m \log n}$

Thus, we have proven both bounds.

2.3 Question d

Prove that in step 22 of RandSelect2, $|T| = O(n/\log n)$ holds w.h.p in n .

Solution:

Let X be a random variable.

$$X_i = \begin{cases} 1 & \text{if } A[i] \text{ is included in } T \\ 0 & \text{otherwise} \end{cases}$$

$$E[X_i] = \Pr[X_i = 1] = 1/\log^2 n$$

$$\therefore E[X] = \sum_{i=1}^{n \log n} \frac{1}{\log^2 n} = n / \log n$$

To prove w.h.p, we use the Chernoff bound

$$\Pr[X \leq (1 - \delta)\mu] \leq e^{-\mu\delta^2/2}$$

$$\mu = n / \log n$$

Taking

$$\delta = \frac{1}{2}$$

$$(1 - \delta)\mu = n / 2 \log n$$

$$\Pr[X \leq n / 2 \log n] \leq e^{-(n / \log n)(\frac{1}{2})^2/2}$$

$$\Pr[X \leq n / 2 \log n] \leq e^{-(n / \log n)/8}$$

$$\Pr[X \leq n / 2 \log n] \leq 1 / e^{(n / \log n)/8}$$

$$\Pr[X \leq n / 2 \log n] \geq 1 - 1 / e^{(n / \log n)/8}$$

Thus, w.h.p in n , $\Pr[X = O(n / \log n)]$

2.5 Question e

Give an upper bound (the best you can come up with) on the running time of RandSelect2 that holds w.h.p in n

Solution:

We consider the worst case running times of all lines in the algorithm to find the upper bound of RandSelect2.

- Line 2 takes $O(n)$ time, using a standard binary heap.
- Line 5 takes $O(\log^2 n)$ time, to select $\log^2 n$ elements from A
- Line 8 sorts the elements of S in $O(q \log q)$ time, where $q = |S|$

In the worst case, $q = \log^2 n$, hence this line takes $O(\log^2 n \log \log^2 n)$

- Line 11 takes $O(n)$ time, to loop over n times
- The for loop in line 13 takes at most $O(n / \log \log n) * \log \log^2 n$ time.

As $|A'| = n/\log \log n$ in the worst case and the number of bins $= \log^2 n$. A binary search on these bins requires $\log \log^2 n$ time. This equation is dominated by the $O(n)$ time.

- Lines 17-18 take at most $O(\log^2 n)$ time to calculate l and r
- Lines 20-21 take $O(n)$ time to complete the for loop
- Line 22 takes at most $\log^2 n \log \log n$ time to perform the sort.

Therefore, the cost $O(n)$ dominates the cost of all other operations and is an upper bound on the running time of RandSelect2

2.6 Question f

Does RandSelect2 ever fail to produce an answer? Why or why not? Is the answer it produces always guaranteed to be correct? Why or why not?

Solution:

Rand-Select2 can fail to produce an answer. The algorithm involves selecting elements from A with a probability of $1/\log \log n$ and copying it into A'.

3 cases are possible, $|A| \leq |A'|$, $|A| \geq |A'|$ or $|A| == |A'|$

it is possible that the A' does not contain the k^{th} smallest element of A

In the worst case, A' contains no elements and thus no element is present is k_{left} or k_{right} and the array $T = \emptyset$. In this case, the algorithm will not produce a result.

In another case, $|A'|$ is less than the k^{th} smallest element and A' does not contain enough elements for the algorithm to produce a result.

Results produced by the algorithm are not guaranteed to be correct. Consider the following cases:

- a) A trivial case is if the k^{th} smallest element is not chosen in line 5 and hence A' does not contain the k^{th} smallest element. In this case the result produced will be incorrect.
- b) Another case is when the k^{th} smallest element is chosen in A' but the partitions k_{left} and k_{right} are not partitioned correctly. If both k_{left} and k_{right} in A' are either greater than or less than the k^{th} smallest element. In this case, the range of s_l and s_{r+1} calculated in line 14 will be incorrect, and the k^{th} smallest element will not be present in T, producing an incorrect result.

1 Task 3

1.1 Question a

Prove that the expected number of equipment activated by RAN D-N CS is $\frac{61}{64}n$ (i.e., 95.3125% of n).

Solution:

Let us define a random variable X_i for an $E(Equipment)$ such that:

$$X_i = \begin{cases} 1 & ; \text{if } E \text{ is active due to the combination of states of its corresponding switches} \\ 0 & ; \text{otherwise} \end{cases}$$

We then have:

$$Pr(X_i = 1) = \frac{1}{2} + \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^3 + \dots + \left(\frac{1}{2}\right)^x$$

Now, we can define $E[X]$ as:

$$E[X] = \sum_{i=0}^{n-1} E[X_i] = \left(\frac{1}{2} + \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^3 + \dots + \left(\frac{1}{2}\right)^x\right) \times N_x$$

Where N_x is the number of equipments that can be turned on by a combination of switches.

Using the values of N_x supplied by in the question:

We get N_x

$$N_x = \begin{cases} 0.1n & ; \text{for a combination of 3 switches} \\ 0.2n & ; \text{for a combination of 4 switches} \\ 0.7n & ; \text{for a combination of 5 switches} \end{cases}$$

Plugging in the values of N_x , we get:

$$\mu = E[X] = \left[\left(\frac{1}{2}\right) + \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^3\right] \times 0.1n + \left[\left(\frac{1}{2}\right) + \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^3 + \left(\frac{1}{2}\right)^4\right] \times 0.2n + \left[\left(\frac{1}{2}\right) + \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^3 + \left(\frac{1}{2}\right)^4 + \left(\frac{1}{2}\right)^5\right] \times 0.7n$$

$$\mu = E[X] = \left(\frac{7}{8}\right) \times 0.1n + \left(\frac{15}{16}\right) \times 0.2n + \left(\frac{31}{32}\right) \times 0.7n$$

$$\boxed{\mu = E[X] = \frac{61}{64}n}$$

1.2 Question b

Prove that the probability that RAN D-N CS at least 61 n equipment is at least $\frac{1}{64}n$

Solution:

Considering the following:

- Since $\lceil (\frac{61}{64}n - 1) \rceil \geq (\frac{61}{64}n - 1)$ then,
 $\lceil (\frac{61}{64}n - 1) \rceil = (\frac{61}{64}n - 1)$ if $(\frac{61}{64}n - 1)n$ is an integer else
 $\lceil (\frac{61}{64}n - 1) \rceil > (\frac{61}{64}n - 1)$
- $\lceil (\frac{61}{64}n - 1) \rceil$ is the closest integer value greater than or equal to $(\frac{61}{64}n - 1)$
- X takes only integer values
- $\lceil (\frac{61}{64}n - 1) \rceil = (\frac{61}{64}n - 1) + c$, where c , $0 \leq c \leq 1$ is added to make $(\frac{61}{64}n - 1)$ an integer.

$E[X]$ for a random variable is defined as :

$$E[X] = \sum_{k=1}^n K Pr[X = K] ; \text{ Where } X \text{ takes a value between 1 and } n.$$

To prove $\frac{1}{64n} \leq Pr[X \geq \frac{61n}{64}]$, use $E[X]$ computed in the previous question:

$$\begin{aligned} \frac{61}{64}n &= \sum_{k=1}^{\lceil \frac{61n}{64} - 1 \rceil} K Pr[X = K] + \sum_{L=\lceil \frac{61n}{64} \rceil}^n L Pr[X = L] \\ \frac{61}{64}n &\leq \sum_{k=1}^{\lceil \frac{61n}{64} - 1 \rceil} (\frac{61n}{64} - 1) Pr[X = K] + \sum_{l=\lceil \frac{61n}{64} \rceil}^n n Pr[X = L] \end{aligned}$$

Because the max value K can take is $\frac{61n}{64} - 1$ and the max value L can take is n .

$$\frac{61}{64}n \leq (\frac{61n}{64} - 1 + c) \sum_{k=1}^{\lceil \frac{61n}{64} - 1 \rceil} Pr[X = K] + n \sum_{l=\lceil \frac{61n}{64} \rceil}^n Pr[X = L]$$

$$\text{But since, } \sum_{k=1}^{\lceil \frac{61n}{64} - 1 \rceil} Pr[X = K] \leq 1$$

$$\frac{61}{64}n \leq (\frac{61n}{64} - 1 + c) \times 1 + n \sum_{l=\lceil \frac{61n}{64} \rceil}^n Pr[X = L]$$

Where c is the offset added to obtain the ceil value and $0 \leq c \leq 1$, so we can drop the value of c

$$1 \leq n \sum_{l=\lceil \frac{61n}{64} \rceil}^n Pr[X = L]$$

$$\text{But, } \sum_{l=\lceil \frac{61n}{64} \rceil}^n Pr[X = L] = Pr[X \geq \frac{61n}{64}] \text{ So,}$$

$$\begin{aligned} 1 &\leq n Pr[X \geq \frac{61n}{64}] \\ \frac{1}{n} &\leq Pr[X \geq \frac{61n}{64}] \end{aligned}$$

$$\boxed{\therefore \frac{1}{64n} \leq Pr[X \geq \frac{61n}{64}]}$$

1.3 Question c

Use your results from part 3(b) to design an algorithm that activates at least $61n$ equipment w.h.p. in n . What is the running time of your algorithm?

Solution:

From the result we obtained in 3(b), we know that the probability of success of $rand_ncs()$ is given by:

$$Pr[X \leq \frac{61n}{64}] \leq \frac{1}{64n}$$

And hence,

$$\text{the Probability of failure} \leq (1 - \frac{1}{64n})$$

Now, if we run $rand_ncs()$ $64n$ times, the probability of failure changes to:

$$Pr_{failure} \leq (1 - \frac{1}{64n})^{64n}$$

Which reduces to :

$$Pr_{failure} \leq (\frac{1}{e})$$

We now run the above step $\log n$ times:

So, we get:

$$Pr_{failure} \leq (\frac{1}{e})^{\log n} \leq \frac{1}{n}$$

Thus the probability of success is:

$$Pr[X \leq \frac{61n}{64}] = 1 - \frac{1}{n} \text{ whp in } n$$

Thus we can design an algorithm as follows:

Algorithm 1 Rand_NCS

```

1: procedure RAND_NCS()
2:   for  $i = 1$  to  $n_b \rightarrow$  where  $n_b$  is the number of buttons
3:      $x = coin\_toss()$ 
4:     if  $x = \text{heads}$ 
5:       activate  $S_i$ 
6:     else
7:       do nothing

```

By analyzing the pseudocode for *modified_rand_ncs*, we get the following complexity:

Algorithm 2 Reset_NCS

```
1: procedure RESET_NCS()
2:   for  $i = 1$  to  $n_b \rightarrow$  where  $n_b$  is the number of buttons
3:     reset  $S_i$ 
```

Algorithm 3 Modified Rand_NCS

```
1: procedure MODIFIED_RAND_NCS()
2:   for  $i = 1$  to  $\log n$ 
3:     j=1 to  $64n$ 
4:        $rand\_ncs()$ 
5:        $reset\_ncs()$ 
```

$O(\log n \times 64n \times n_b)$

We can calculate the upper bound on n_b as follows:

$$n_b = 0.1 \times 3n + 0.2 \times 4n + 0.7 \times 5n = 4.6n$$

Thus, O reduces to:

$O(n^2 \log n)$

1.4 Question 3d

Given any $\epsilon \in (0, \frac{61}{64})$, prove that RAND-NCS activates at least $(\frac{61}{64}\epsilon)n$ equipment with probability at least ϵ .

Solution:

$E[X]$ for a random variable is defined as :

$E[X] = \sum_{k=1}^n KPr[X = K]$

 ; Where X takes a value between 1 and n.

Considering the following:

- Since $\lceil (\frac{61}{64} - \epsilon)n \rceil \geq (\frac{61}{64} - \epsilon)n$ then,
 $\lceil (\frac{61}{64} - \epsilon)n \rceil = (\frac{61}{64} - \epsilon)n$ if $(\frac{61}{64} - \epsilon)n$ is an integer else
 $\lceil (\frac{61}{64} - \epsilon)n \rceil > (\frac{61}{64} - \epsilon)n$
- $\lceil (\frac{61}{64} - \epsilon)n \rceil$ is the closest integer value greater than or equal to $(\frac{61}{64} - \epsilon)n$
- X takes only integer values
- $\lceil (\frac{61}{64} - \epsilon)n \rceil = (\frac{61}{64} - \epsilon)n + c$, where c , $0 \leq c \leq 1$ is added to make $(\frac{61}{64} - \epsilon)n$ an integer.

$$\frac{61}{64}n = \sum_{k=1}^{\lceil (\frac{61}{64} - \epsilon)n \rceil} KPr[X = K] + \sum_{L=\lceil (\frac{61}{64} - \epsilon)n \rceil}^n LPr[X = L]$$

$$\frac{61}{64}n \leq \sum_{k=1}^{\lceil (\frac{61}{64}-\epsilon)n \rceil} ((\frac{61}{64}-\epsilon)n)Pr[X=K] + \sum_{l=\lceil (\frac{61}{64}-\epsilon)n \rceil}^n nPr[X=L]$$

Because the max value K can take is $\frac{61n}{64} - 1$ and the max value L can take is n.

$$\frac{61}{64}n \leq ((\frac{61}{64}-\epsilon)n + c) \sum_{k=1}^{\lceil (\frac{61}{64}-\epsilon)n \rceil} Pr[X=K] + n \sum_{l=\lceil (\frac{61}{64}-\epsilon)n \rceil}^n Pr[X=L]$$

$$\text{But since, } \sum_{k=1}^{\lceil (\frac{61}{64}-\epsilon)n \rceil} Pr[X=K] \leq 1$$

$$\frac{61}{64}n \leq ((\frac{61}{64}-\epsilon)n + c) \times 1 + n \sum_{l=\lceil (\frac{61}{64}-\epsilon)n \rceil}^n Pr[X=L]$$

Where c is the offset added to obtain the ceil value and $0 \leq c \leq 1$, so we can drop the value of c

$$n\epsilon \leq n \sum_{l=\lceil (\frac{61}{64}-\epsilon)n \rceil}^n Pr[X=L]$$

$$\text{But, } \sum_{l=(\frac{61}{64}-\epsilon)n}^n Pr[X=L] = Pr[X \geq (\frac{61}{64}-\epsilon)n]$$

So,

$$\boxed{\epsilon \leq Pr[X \geq (\frac{61}{64}-\epsilon)n]}$$

1.5 Question 3e

Use your results from part 3(d) to design an algorithm that activates at least $(\frac{61}{64}\epsilon)n$ equipment w.h.p. in n. What is the running time of your algorithm?

Using the results from part 3(d), we can represent the probability of failure as:

$$Pr[X \leq (\frac{61}{64}-\epsilon)n] \leq 1 - \epsilon$$

Put,

$$1 - \epsilon = \delta$$

Thus,

$$Pr[X \leq (\frac{61}{64}-\epsilon)n] \leq \delta$$

$$(Pr[X \leq (\frac{61}{64}-\epsilon)n])^x \leq \delta^x$$

Substituting $x = \log n$, we get:

$$(Pr[X \leq (\frac{61}{64}-\epsilon)n])^{\log n} \leq \delta^{\log n}$$

$$(Pr[X \leq (\frac{61}{64}-\epsilon)n])^{\log n} \leq n^{\log \delta}$$

$P[\text{overall success in } \log n \text{ trails}] \geq 1 - n^{\log \delta}$

But we know that if:

$\epsilon \in (0, \frac{61}{64})$ then,

$1 - \epsilon \in (1 - 0, 1 - \frac{61}{64})$

Hence,

$1 - \epsilon = \delta \in (\frac{3}{64}, 1)$ We can define δ as:

$\frac{3}{64} < \delta < 1 \Rightarrow \log \delta < 0$

Put $\log \delta = -\alpha$

$\therefore P[\text{overall success}] \geq 1 - n^{-\alpha}$

$P[\text{overall success}] \geq 1 - \frac{1}{n^\alpha}$

$P[\text{overall success}] \geq 1 - \frac{1}{n^{\log \frac{1}{1-\epsilon}}}$

Therefore, we can design an algorithm that activates at least $(\frac{61}{64} - \epsilon)n$ equipment w.h.p.

Algorithm 4 Modified Rand_NCS

```

1: procedure FINAL_RAND_NCS()
2:   for  $i = 1$  to  $\log n$ 
3:      $rand\_ncs()$ 

```

The complexity of the above algorithm will be:

$O(\log n \times n_b) = \boxed{O(n \log n)}$