# Homework #December 1

**( Due: December 1 )**

GROUP NUMBER: 19

| GROUP MEMBERS | | |
|---|---|---|
| NAME | SBU ID | % CONTRIBUTION |
| CHAITANYA KALANTRI | 111446728 | 33.33 |
| NEEL PARATKAR | 111483570 | 33.33 |
| ROHAN KARHADKAR | 111406429 | 33.33 |

## COLLABORATING GROUPS

| Group Number | Specifics (e.g., specific group member? specific task/subtask?) |
|---|---|
| 19 | Rahul Bhansali, Kiran (Question 2) |
| 38 | Aishwarya (Question 1) |
| 74 | Varun (Question 3) |

## EXTERNAL RESOURCES USED

| | Specifics (e.g., cite papers, webpages, etc. and mention why each was used) |
|---|---|
| 1. | |
| 2. | |
| 3. | |
| 4. | |
| 5. | |

**Task 3. [ 45 Points ] Flipping Coins in NCS 220**

**Question 3(a)** Prove that the expected number of equipment activated by RAND-NCS is $\frac{61}{64n}$ (i.e., 95.3125 % of n).

**Ans.** In order to prove that the expected number of equipment activated id $\frac{61n}{64}$.
We will follow the given steps:
1. Compute the Indicator variable (Xi)
2. Compute the Expected value (E[X])
3. Compute the Mu
Let us define $X_i$ as the percentage of equipment controlled by the given number of buttons,

$$Xi = \begin{cases} 10\%, & Equipment \quad controlled \quad by \quad 3 \quad buttons \\ 20\%, & Equipment \quad controlled \quad by \quad 4 \quad buttons \\ 70\%, & Equipment \quad controlled \quad by \quad 5 \quad buttons \\ 0\%, & Otherwise \end{cases} \tag{5.0.1}$$

Consider that if 'n' be the number of buttons controlling the equipment, total number of possible combinations of the buttons will be $2^n$ and there will be one possible combination for which the equipment wont be working. So, initially we will derive the probability that the equipment is not activated.

We know, E[X] = $\sum$ I x Pr[$X_i$ = I ] = $(10\% \times \frac{1}{2^3}) \times (20\% \times \frac{1}{2^4}) \times (70\% \times \frac{1}{2^5})$ = $\frac{3}{64}$
As we have found out that the Expected value of the EQUIPMENT is deactivated. However, we need to find the probability that the equipment is activated. Which will be 1 - $\frac{3}{64}$ = $\frac{61}{64}$

Expected value of n equipments:-
Mu = E[$X_i$] = $\sum_{i=1}^{n} X_i \Rightarrow \frac{61n}{64}$

**Question 3(b)** Prove that the probability that RAND-NCS at least $\frac{61n}{64}$ equipment is at least $\frac{1}{64n}$

**Ans.** As per the question, we need to prove $Pr[X \geq \frac{61n}{64}] \geq \frac{1}{64n}$
E[X] = $\sum_{j=0}^{n} E[X_j]$
And from part(a), weknow that: E[X] = $\frac{61n}{64}$

E[X] = $\sum_{j=0}^{n}$ j × Pr[$X_i$ = j]

We define the limits over lower bound and upper bound:-

$\Rightarrow$ E[X] = $\sum_{j=0}^{\frac{61n}{64}-\frac{1}{64}} j \times Pr[X_i = j]$ + $\sum_{j=\frac{61n}{64}-\frac{1}{64}}^{n} j \times Pr[X_i = j]$

$\Rightarrow \frac{61n}{64}$ = $\sum_{j=0}^{\frac{61n}{64}-\frac{1}{64}} j \times Pr[X_i = j]$ + $\sum_{j=\frac{61n}{64}-\frac{1}{64}}^{n} j \times Pr[X_i = j]$

Here we have consider the limits in the above mentioned way, because of the requirement of the question. As if we don't split the limit in this way the answer won't be exactly as per the questions requirement.

We can eliminate the lower bound, as the probability is ¡= . Hence, considering the upper bound on j, max value of j can be $\frac{61n}{64} - \frac{1}{64}$ in the first half of the equation and n in the second half of the equation.

Therefore, the equation can be written as

$\Rightarrow \frac{61n}{64} \leq \ (\frac{61n}{64} - \frac{1}{64}) \times 1 \ + \ n \times Pr[X_i \geq \ \frac{61n}{64}]$

Solving the above equation, we get: $\Rightarrow \frac{61n}{64} - (\frac{61n}{64} - \frac{1}{64}) \ \leq \ n \times Pr[X_i \geq \ \frac{61n}{64}]$

$\Rightarrow \frac{1}{64n} \ \leq \ Pr[X_i \geq \ \frac{61n}{64}]$

Hence Proved, the probability that RAND-NCS at least $\frac{61n}{64}$ equipment is at least $\frac{1}{64n}$

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

**Ans.** Designing an algorithm that activates at least 61n/64 equipment w.h.p. in n. From part 3(b), we got the probability RAND-NCS at least $\frac{61n}{64}$ is atleast $\frac{1}{64}$.
Pr[X > $\frac{61n}{64}$] $\geq \frac{1}{64n}$

$\therefore$ We can deduce that 1 - Pr[X < $\frac{61n}{64}$] $\geq \frac{1}{64n}$

Rearranging the equation we get: Pr[X < $\frac{61n}{64}$] $\leq$ 1 - $\frac{1}{64n}$

Therefore, the probability that equipment is deactivated is at most $\frac{1}{64n}$ if we run the algorithm once. Run the algorithm $64n$ times

We know that,
(1-$\frac{1}{x}$)$^x$ = 1-$(x \times \frac{1}{x})$+$(\frac{x(x-1)}{2!} \times \frac{1}{x^2})$-$(\frac{x(x-1)(x-2)}{3!} \times \frac{1}{x^3})$ ...
= 1-1+$\frac{1}{2!} - \frac{1}{3!} + \frac{1}{4!} - \frac{1}{5!}$...
= $e^{-1}$ = $\frac{1}{e}$
Therefore, the probability of success if we run the algorithm $64n$ times will be at least 1-$(1 - \frac{1}{64n})^{64n}$ = 1-$\frac{1}{e}$
Then, Pr[C' $\neq$ C] ¡ $(1 - \frac{1}{64n})^{\frac{1}{64n}}$ ¡ $\frac{1}{e}$ $\Rightarrow \ Pr[C' = C] > 1 - \frac{1}{e}$

Hence, the answer will be: As we have the probability directly in terms of constant. Hence, in order to find the high probability, we will multiple the constant (logn) times.
Hence, this will be equal to (1 - 1/e)$^{logn}$
Running time of the algorithm is $\mathcal{O}(64n + logn)$ = $\mathcal{O}(n)$

**Question 3(d)** Given any $\epsilon \in (0, \frac{61}{64})$, prove that RAND-NCS activates at least $(\frac{61}{64} - \epsilon) \times n$ equipment with probability at least .

**Ans.** We have to prove that $Pr[X \geq (\frac{61}{64} - \epsilon)n] \ \geq \ \epsilon$
E[X] = $\sum_{j=0}^{n} E[X_j]$
from previous question E[X] = $\frac{61n}{64}$

E[X] = $\sum_{j=0}^{n}$ j $\times$ Pr[$X_i$ = j]

We define the limits over lower bound and upper bound:-

$\Rightarrow$ E[X] = $\sum_{j=0}^{(\frac{61}{64}-\epsilon)n}$ $j \times Pr[X_i = j]$ + $\sum_{j=(\frac{61}{64}-\epsilon)n}^{n}$ $j \times Pr[X_i = j]$

$\Rightarrow \frac{61n}{64}$ = $\sum_{j=0}^{(\frac{61}{64}-\epsilon)n}$ $j \times Pr[X_i = j]$ + $\sum_{j=(\frac{61}{64}-\epsilon)n}^{n}$ $j \times Pr[X_i = j]$

Taking an upper bound on j, max value of j can be $(\frac{61}{64} - \epsilon) \times n$ in the first half of the equation and n in the second half of the equation.
$\because$ since the summation in the second half of the equation is an upper bound on E[X] and upper bound on j is n as explained above, we rewrite the equation as

$\Rightarrow \frac{61n}{64} \leq$ $(\frac{61}{64} - \epsilon) \times n \times 1$ + $n \times Pr[X_i \geq (\frac{61}{64} - \epsilon) \times n]$

Solving the equation : $\Rightarrow \frac{61n}{64} - (\frac{61n}{64} - \epsilon n) \leq n \times Pr[X_i \geq (\frac{61}{64} - \epsilon) \times n]$

$\Rightarrow \epsilon \leq Pr[X_i \geq (\frac{61}{64} - \epsilon) \times n]$

Hence, Proved. The probability that RAND-NCS at least $(\frac{61}{64} - \epsilon) \times n$ equipment is at least $\epsilon$

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

Here, as from part(d), we have the probability directly in terms of constant. Hence, in order to find the high probability, we will multiple the constant (logn) times.
Hence, this will be equal to $(\epsilon)^{logn}$

Running time of the algorithm is $\mathcal{O}(logn)$

Solution 1.1:
We can convert the array into min heap or max heap in O(n) time.

1. When k <= (n/log n) we can convert the array into min heap and use extract-min() operation k times to get the kth smallest element. Extract-min() takes O(log n) time. So performing it k times will take O(klog n) time. But k is O(n/logn).
   Hence O(klog n) = O(n/logn x logn) = O(n).
   So in this case the complexity will be O(n)
2. When k >= n – n/log n, getting kth smallest element is same as getting n-k th largest element. Say n-k = j. We need jth largest element. J = n – k = n – (n – n/log n) = n/logn.
   We can find the jth max element, if we convert the array into a max-heap and run extract-max() j times. Extract-min() takes O(log n) time. So running it will take O(j log n) time. But j is O(n log n).
   Hence O(jlog n) = O(n/logn x logn) = O(n).
   So in this case the complexity will be O(n).

Solution 1.2:
We know that the probability of choosing 1 number from n is 1/n.
Let when X = $\begin{cases} 1 & when\ a\ number\ in\ A\ is\ selected \\ 0 & when\ the\ number\ in\ A\ is\ not\ selected \end{cases}$
E[X] over $\log^2 n$ selections = $\log^2 n$ x probability that number is selected.
E[X] = $\log^2 n$ x 1/n = $\log^2 n/n$
Now for a sample space of $\log^2 n$ elements, we can see this problem of getting number of times Heads turns up after tossing a fair coin $\log^2 n$ times. Probability that Heads turns up more than once, is similar to a number getting chosen more than 1 once.
Pr[E[x] >= 2] is the required probability.
We know Pr[E[x] >= d] <= E[x]/d
Here d is 2. So putting the value in above equation we get:
Pr[E[x] >= 2] <= ($\log^2 n$ / n) / 2 = ($\log^2 n$ / 2n)
But we can see as n->∞, $\log^2 n/n$ = 0
Which means that the Pr[E[x] >= 2] = ($\log^2 n$ / 2n) is low for very high n.
This in turn means that the probability Pr[E[x] < 2] is very high, which implies that the probability of selecting a number at most once is very high.
This means that step 5 will select the $\log^2 n$ unique elements with a high probability.

Solution 1.3:
We know that we are choosing $\log^2 n$ numbers. So there will be $\log^2 n$ buckets.
If we say X = $\begin{cases} 1 & when\ a\ number\ in\ A\ \ is\ in\ the\ bin \\ 0 & when\ a\ number\ in\ A\ is\ not\ in\ the\ bin \end{cases}$
Probability that a number is in a bin is $1/\log^2 n$.
$E[X_i] = 1/\log^2 n$. For n numbers $E[X] = n/\log^2 n$.
Let u = E[X] = expected length of a bin
We want to show |T| = O(n/logn) w.h.p
We want to show Pr[|T| <= n/log n) w.h.p
Lets find probability that |T| > n/logn
Pr[|T| >= n/logn]
Here let (1+d)u = n/log n.
But $u = n/\log^2 n$

So d = log n – 1
We know that Pr[E[x] >= (1+d)u) = $e^{-ud^2/3}$
Substituting values we get: Pr[E[x] >= (1+d)u) = $e^{-(n/(logn)^2)(logn-1)^2/3}$
If we approximate (log n – 1)^2 nearly equal to (log n)^2. So LHS becomes $e^{-n/3} = 1/e^{n/3}$
This is a very low probability. Which means that |T| will be more than n/logn with very low probability. Which in turn means that |T| will be <= n/logn with high probability.
Hence we can say that |T| = O(n/logn) with high probability.

Solution 1.4:
Lets take into consideration the running time complexities of parts of the algorithm to analyze the best upper bound.
For lines1-3: as mentioned in solution 1.3 the running time is O(n) using binary heap
For lines 5-6: Selecting can be done in $O(\log^2 n)$ time and duplicated can be removed in $O(\log^2 n)$ time and O(1) space.
For line 8: sorting can be done in (for $M = \log^2 n$) is O(M log M)
For lines 10-12 : Searching can be done using binary search in (for $M = \log^2 n$) in O(log M) time. The loop is run n times hence time will be O(n Log M) = $O(n \log(\log^2 n))$
For line 13: Finding the smallest number can again be done using Binary heap in O(M) time for $M = \log^2 n$
For lines 15-16 : The loop is run over all elements so time is O(N)
For lines 17-18: We have proved that |T| = O(n/log n) with high probability.
Sorting can thus be done for P = n/log n elements in O(Plog P) = O( (n/log n) x (log (n/logn) )

From above complexities we can see that $O(n \log(\log^2 n))$ dominates over all other time complexities. Hence the time overall complexity will be $O(n \log(\log^2 n))$.

Solution 1.5:

1. In the algorithm, we are selecting random numbers from a set. And working on them. These numbers are used to form bins which will be of non-uniform lengths as a result. During the loop in line 10-12 we are considering all the elements of the loop and increase the counter for each bin. So by the definition of the algorithm, we will choose atleast 2 bins with high probability (though we have proved that with high probability that the number of elements chosen at random will be more than $\log^2 n$,which in turn means the number of bins will be $\log^2 n$ with high probability) and subsequently divide the array into more than 2 partitions. As we are not omitting any number in the array during any selection, there will always be non-zero number of elements in each bin. Hence it will always return a number. Hence the algorithm will never fail to return an answer.
2. By the definition of the algorithm, we are first creating bins, selecting a bin with number of elements in it and before it is less than k. Then we return the Tth element such that t = k – number of elements in bins before the pth bucket. Tth element in pth bucket is thus kth smallest element in the entire array. Hence the algorithm always returns correct answer.

**Task 2. (a):**

We have to bound the probability of adding element from A to A'.

Let

$Xi = \{ \quad 1 \quad if\ the\ ith\ element\ of\ A\ is\ added\ to\ A'$

$\qquad\quad o \quad otherwise$

Then the number of elements added in n iterations is

$X = \Sigma i = 1\ to\ n\ Xi$

We know, E[Xi] = Pr[Xi = 1] = $1 / log\ log\ n$

Hence, E[X] = $\Sigma i = 1\ to\ n\ E[Xi] = n / log\ log\ n$

Proving |A'| = $\theta(n/log\ log\ n)$ holds w.h.p in n is equivalent to proving

$c1\ .\ n / log\ log\ n <= |A'| <= c2\ .\ n / log\ log\ n$

Splitting it, we get

$|A'| >= c1\ .\ n / log\ log\ n$

$|A'| <= c2\ .\ n / log\ log\ n$

To prove that they occur w.h.p in n, we can prove the following occurs with low probability in n:

$|A'| <= c1\ .\ n / log\ log\ n + constant$

$|A'| >= c2\ .\ n / log\ log\ n - constant$

I.e. proving Pr($X >= (1 + \delta)\mu$) and Pr($X <= (1 - \delta)\mu$) occurs with low probability in n proves A' = $\theta(n / log\ log\ n)\ w.h.p\ in\ n$

Let $\delta = 0.5$. As $0 < \delta < 1$, we can apply Chernoff Bounds (Case 2), for any $\delta$ in the said range.

Pr ($X <= (1 - 0.5)\ .\ n / log\ log\ n$) <= $e^{-\mu\delta 2/2}$

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

So, $Pr\ (X <= 0.5\ X\ n / log\ log\ n) <= e^{-n/\ 8log\ log\ n}$

As n's value becomes larger and larger, the value will become close to 0. So we can say that Pr ($X <= 0.5\ X\ n / log\ log\ n$) occurs with low probability in n as n becomes larger and larger.

We can prove in the same way that

Pr ($X >= (1 + 0.5)\ .\ n / log\ log\ n$) <= $e^{-\mu\delta 2/3}$

So, $Pr\ (X >= 1.5\ X\ n / log\ log\ n) <= e^{-\mu\delta 2/3}$

Substituting the values, we get

$Pr\ (X >= 1.5\ X\ n / log\ log\ n) <= e^{-n/\ 12log\ log\ n}$

As n's value becomes larger and larger, the value will become close to 0. So we can say that Pr ($X >= 1.5\ X\ n / log\ log\ n$) occurs with low probability in n as n becomes larger and larger.

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

**Task 2. (b):**

Here, we are given $\Delta = kright - kleft$ where $kright = k' + 2(\sqrt{(m - k')\, X\, log\ n})$ and $kleft = k' - 2(\sqrt{k'\, X\, log\ n})$

Solving for $\Delta$, we get

$$\begin{aligned}\Delta &= kright - kleft \\ &= k' + 2(\sqrt{(m - k')\, X\, log\ n}) - (k' - 2(\sqrt{k'\, X\, log\ n})) \\ &= 2(\sqrt{(m - k')\, X\, log\ n}) + 2(\sqrt{k'\, X\, log\ n}) \\ &= 2\sqrt{logn}\ X\ (\sqrt{(m - k')} + \sqrt{k'})\end{aligned}$$

Substituting k' = k . m / n in the above equation,

$$\begin{aligned}\Delta &= 2\sqrt{logn}\ X\ (\sqrt{(m - mk/n)} + \sqrt{mk/n}) \\ &= 2\sqrt{mlogn}\ X\ (\sqrt{(1 - k/n)} + \sqrt{k/n})\end{aligned}$$

Calculating expected value of $\Delta$:

Let

$$Xi = \begin{cases} 1 & \text{if the ith element is between kright and kleft} \\ o & \text{otherwise} \end{cases}$$

Then the total number of elements

$X = \Sigma i = 1\ to\ m\ Xi$

We know, E[Xi] = Pr[Xi = 1] = $(2\sqrt{mlogn}\ X\ (\sqrt{(1 - k/n)} + \sqrt{k/n}\ ))\ /\ m$

Hence, E[X] = $\Sigma i = 1\ to\ m\ E[Xi] = 2\sqrt{mlogn}\ X\ (\sqrt{(1 - k/n)} + \sqrt{k/n})$

Where E[Xi] = $\mu$

We have to show that w.h.p in n, $2\sqrt{m\ log\ n} <= \Delta <= 2\sqrt{2m\ logn}$ w.h.p in n.

This is equivalent to proving:

Pr ($2\sqrt{m\ log\ n} <= \Delta$) w.h.p. in n and Pr ($2\sqrt{2m\ log\ n} >= \Delta$) w.h.p. in n

To prove that they occur w.h.p in n, we can prove the following occurs with low probability in n:

Pr ($2\sqrt{m\ log\ n} >= \Delta$) and Pr ($2\sqrt{2m\ log\ n} <= \Delta$)

To find $\Delta$,

$$(1 + \delta)\,\mu = 2\sqrt{2m\ log\ n}$$

$$(1 + \delta)\ 2\sqrt{m\ log\ n}\ X\ (\sqrt{(1 - k/n)} + \sqrt{k/n}) = 2\sqrt{2m\ log\ n}$$

$$1 + \delta = \sqrt{2}/\ (\sqrt{1 - k/\ n} + \sqrt{k\ /\ n})$$

$$\delta = \sqrt{2}/\ (\sqrt{1 - k/\ n} + \sqrt{k\ /\ n}) - 1$$

From the above equation, we can find that k lies between 1 and n and this results in $\delta$ values lying between 0 and 1.

We can apply Chernoff Bounds (Case 2), for any $\delta$ in the said range.

Pr ($\Delta \geq 2\sqrt{2mlogn}$) $\leq e^{-\mu\delta^2/3}$

Substituting the $\delta$ and $\mu$ values, we get

Pr ($\Delta \geq 2\sqrt{2mlogn}$) $\leq e^{(-2\sqrt{mlogn})(\sqrt{1-\frac{k}{n}}+\sqrt{\frac{k}{n}})*(\frac{\sqrt{2}}{\left(\sqrt{1-\frac{k}{n}}+\sqrt{\frac{k}{n}}\right)}-1)^2/3}$

$\leq e^{-2\sqrt{mlogn}(\sqrt{2}-(\sqrt{1-\frac{k}{n}}+\sqrt{\frac{k}{n}}))^2/3}$

As k becomes closer and closer to n,
Pr ($\Delta \geq 2\sqrt{2mlogn}$) $\leq e^{-c\sqrt{mlogn}}$

As n increases,
Pr ($\Delta \geq 2\sqrt{2mlogn}$) becomes closer and closer to 0.
Therefore, we can say that Pr ( $\Delta \geq 2\sqrt{2mlogn}$) happens with a low probability in n.

Similarly, we can prove for
Pr ( $\Delta \leq 2\sqrt{mlogn}$)

Finding Chernoff Bounds:
(1 - $\delta$) $\mu$) = $2\sqrt{mlogn}$
(1 - $\delta$) $2\sqrt{mlogn}$)( $\sqrt{1-k/n}$ + $\sqrt{k/n}$ ) = $2\sqrt{mlogn}$
(1 - $\delta$) = $\frac{1}{(\sqrt{1-\frac{k}{n}}+\sqrt{k/n})}$
$\delta$ = $\frac{1}{(\sqrt{1-\frac{k}{n}}+\sqrt{k/n})}$ + 1

K lies between1 and n. If k is set to n, value of $\delta$ will lie between 0 and 1.
Applying Chernoff Bounds using rule 2, we get:
Pr ($\Delta \leq 2\sqrt{mlogn}$) $\leq e^{-\mu\delta^2/2}$

Substituting the values, we get:

Pr ( $\Delta \leq 2\sqrt{mlogn}$) $\leq e^{-2\sqrt{mlogn})(\sqrt{1-\frac{k}{n}}+\sqrt{\frac{k}{n}})*(\frac{1}{\left(\sqrt{1-\frac{k}{n}}+\sqrt{\frac{k}{n}}\right)}+1)^2/2}$

Pr ( $\Delta \leq 2\sqrt{mlogn}$) $\leq e^{-2\sqrt{mlogn}(1+(\sqrt{1-\frac{k}{n}}+\sqrt{\frac{k}{n}}))^2/2}$

If k comes closer and closer to n, it becomes a constant number.
Pr ( $\Delta \leq 2\sqrt{mlogn}$) $\leq e^{-c\sqrt{mlogn}}$

Therefore, as n increases, the values of Pr ( $\Delta \leq 2\sqrt{mlogn}$) becomes closer and closer to 0. Therefore, Pr ( $\Delta \leq 2\sqrt{mlogn}$) occurs with a low probability in n.

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

## Problem No 2

**2 d.**

According to the problem,

We have , |T| = $O(\frac{n}{logn})$ w.h.p in n.

Algorithm (Lines 20-21), says that the number of elements in T directly depends on the number of elements in a bin.

So, find w.h.p the number of elements in a bin.

Define an indicator variable $X_i$ such that

$X_i$ = { 1, if the element from A lies in a bin
0, otherwise}

Expected value of $X_i$:

E[$X_i$] = 1 * Pr($X_i = 1$) + 0 * Pr($X_i = 0$)

= Pr($X_i = 1$)

which is probability of an element lying in a particular bin:

= $\frac{1}{(log\ n)^2}$ (as there are $(\log n)^2$ bins)

Total expected value of E[$X$] will be:

E[X] = $\sum_{i=1}^{n/\log(\log n)} \frac{1}{(\log n)^2}$ = $\frac{n}{\log(logn)\ (\log n)^2}$ = $\mu$

So, we need to show:

Pr($X < \frac{n}{\log n}$) is high

⇨ Pr($X \geq \frac{n}{\log n}$) is low.

So, we'll use Chernoff bound:

Pr($X \geq (1+\delta)\mu$) $\leq (\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}})^{\mu}$

Here,

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

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

By simplifying, we get:

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

Using Chernoff Bound:

$$\Pr(X \geq (1+\delta)\mu) \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu}$$

Substituting the values, we get:

$$\Pr\left(X \geq \frac{n}{\log n}\right) \leq \left(\frac{e^{\log n \log\log n - 1}}{(\log n \log\log n)^{(\log n \log\log n)}}\right)^{\frac{n}{(\log n)^2 \log\log n}}$$

$$\Pr\left(X \geq \frac{n}{\log n}\right) \leq \left(\frac{e^{\frac{n}{\log n} - \frac{n}{(\log n)^2 \log\log n}}}{(\log n \log\log n)^{\frac{n}{\log n}}}\right)$$

By simplifying, we get:

$$\Pr\left(X \geq \frac{n}{\log n}\right) \leq \left(\frac{e^{\frac{n}{\log n}\left(1 - \frac{1}{\log n \log\log n}\right)}}{(\log n \log\log n)^{\frac{n}{\log n}}}\right)$$

$e^{\frac{n}{\log n}\left(1 - \frac{1}{\log n \log\log n}\right)}$ = constant and it is dominated by the denominator i.e. $(\log n \log\log n)^{\frac{n}{\log n}}$ as the value of n increases.

So, the whole expression:

$\frac{e^{\frac{n}{\log n}\left(1 - \frac{1}{\log n \log\log n}\right)}}{(\log n \log\log n)^{\frac{n}{\log n}}}$ will becomes low as n becomes large.

Therefore,

$\Pr(X \geq \frac{n}{\log n})$ is low and hence,

$\Pr(X < \frac{n}{\log n})$ = 1 - $\Pr(X \geq \frac{n}{\log n})$

$$= 1 - \frac{e^{\frac{n}{\log n}\left(1 - \frac{1}{\log n \log\log n}\right)}}{(\log n \log\log n)^{\frac{n}{\log n}}}$$

= high probability in n.

So, we have shown that the number of elements in a bin will be $O(\frac{n}{logn})$ w.h.p in n.
As |T| will be some constant factor times number of elements in a bin, so we can say |T| = $O(\frac{n}{logn})$.
Hence, proved.

**Task 2. (e):**

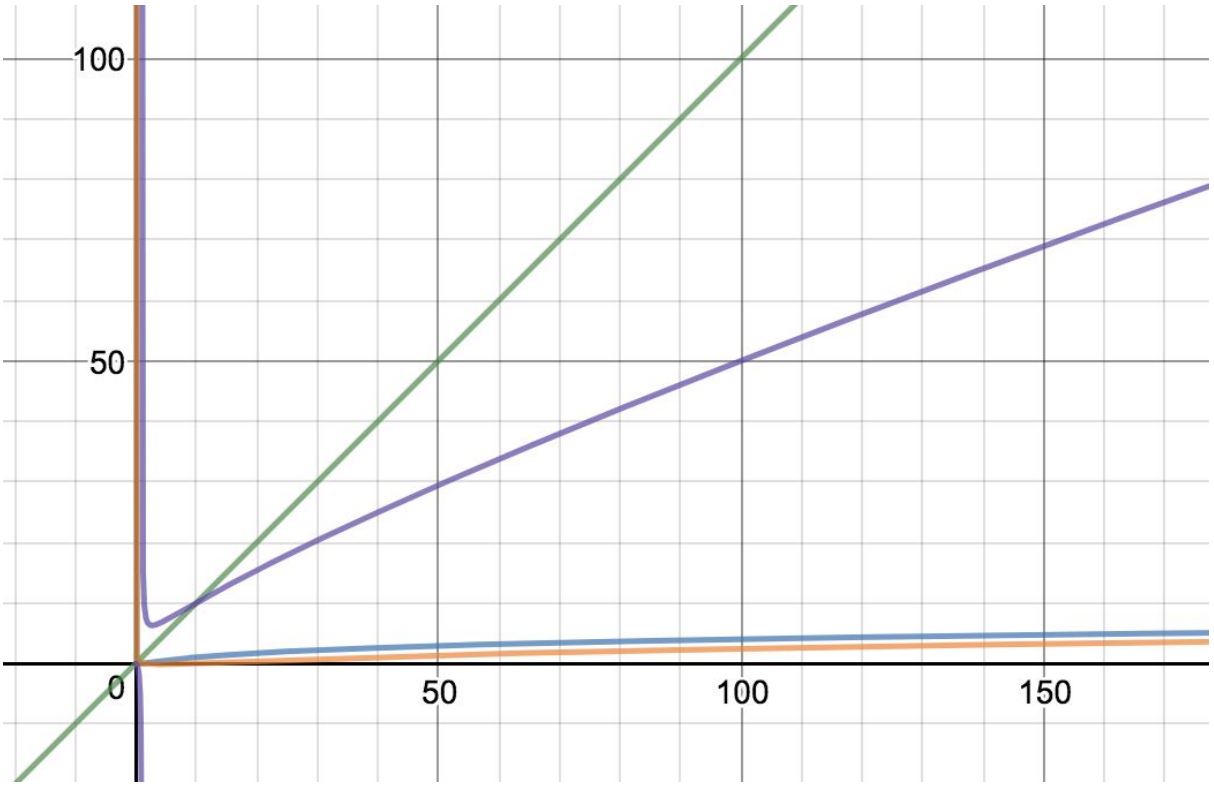
Following are the time complexities of every line of code:
RandSelect-2( A, n, k )

| Line No. | Code | Time Complexity | Explanation |
|---|---|---|---|
| 1 | if k < (n / log n) or k >= (n - n / log n) then | O(1) | Comparison |
| 2 | compute the k-th smallest element x of A in O(n) time using a standard binary heap | O(n) | $k^{th}$ smallest element using binary heap |
| 3 | return x | O(1) | returning x |
| 4 | else | O(1) | |
| 5 | choose $\log^2 n$ elements uniformly at random from A[1 : n] (with replacement) | O($\log^2 n$) | Choosing the same amount of elements |
| 6 | let S be the set of elements chosen in step 5 after removing duplicates | O($\log^2 n$) | Putting these same amount of elements in a set |
| 7 | $q \leftarrow \lvert S \rvert,\ s0 \leftarrow -\infty,\ sq + 1 \leftarrow +\infty$ | O(1) | Assigning operations |
| 8 | sort the elements of S in increasing order of value, and let s1, s2, . . . , sq be those elements in sorted order | O($\log^2 n \log(\log^2 n)$) | Sorting n elements is O(n log n), so sorting $\log^2 n$ elements takes O($\log^2 n \log(\log^2 n)$) time |
| 9 | let Bi be a bin with range (si, si+1], and count ci $\leftarrow$ 0, where $0 <= i <= q$ | O(1) | Creating bin and assigning operation to count ci |
| 10 | A' $\leftarrow \phi$ | O(1) | Assigning operation |
| 11 | for j $\leftarrow$ 1 to n do | O(n) | For loop running n times |
| 12 | $A' \leftarrow A' \cup \{A[j]\}$ with probability 1/ log log n | O(n) | For loop running n times |
| 13 | for each z $\varepsilon\ A'$ do | O(n) | $n / log\ log\ n$ $X\ log\ (log 2\ n)$ = $n/\ log\ log\ n$ $X\ log(log\ n * log\ n)$ |

| | | | |
|---|---|---|---|
| | | | = $n / log\ log\ n\ X$ $log\ log\ n\ +\ log\ log\ n$ = 2n |
| 14 | find bin Bi such that $z\ \varepsilon\ (si,\ si + 1]$ | O(n) | Same as 13 |
| 15 | $ci \leftarrow ci\ +\ 1$ | O(n) | Same as 13 |
| 16 | $m \leftarrow \|A'\|,\ k' \leftarrow m.k/n,\ kleft \leftarrow k' - 2\sqrt{k'\ log\ n}$, kright $\leftarrow k'\ +\ 2\sqrt{(m\ -\ k')log\ n}$ | O(1) | Assigning operations |
| 17 | find the smallest l $\varepsilon$ [0, q] such that kleft <= $\Sigma i\ =\ 0\ to\ l\ ci$ | O($\log^2$ n) | Traversing elements of $\log^2$ n size |
| 18 | find the smallest r $\varepsilon$ [0, q] such that kright <= $\Sigma i\ =\ 0\ to\ r\ ci$ | O($\log^2$ n) | Traversing elements of $\log^2$ n size |
| 19 | T $\leftarrow \phi$ | O(1) | Assigning operation |
| 20 | For j 1 to n do | O(n) | For loop running n times |
| 21 | $if\ A[j]\ 2\ (sl,\ sr + 1]\ then\ T \leftarrow T\ \cup\ \{A[j]\}$ | O(n) | For loop running n times |
| 22 | sort the elements of T in increasing order of value | O(n / log n) | From Task 2. (d) |
| 23 | find the t-th smallest element x in the sorted version of T, where $t\ =\ k\ -\ \Sigma i\ =\ 0\ to\ l\ -\ 1\ ci$ | O($\log^2$ n) | Worst case to find $t^{th}$ element in T |
| 24 | return x | O(1) | Returning x |

So, the total worst case complexity will be O(n).

Here is a picture depicting the complexities where the worst complexity is given by n:




**Task 2. (f):**

Yes, RANDSELECT-2 can fail to produce an answer.

No, the answer it produces always guaranteed to be correct. That means RANDSELECT-2 can produce incorrect output.

For instance, let's consider that their is no from A which gets chosen for A'. Which means A' is empty, as we add elements from A only when they get chosen.

Moreover, there is a very low, but greater than zero probability that the element is picked.

Which can be written as,
Probability of an element from A not getting selected for A' = 1- 1/ (logn*logn)

Hence, the probability that no element of A is picked for A' =
$(1 - 1/(logn * logn))^2 > 0$

Proof:
Consider A' is empty .

| Line No. | Code Implication |
|---|---|
| 1 | l =0 , r = 0 |
| 9 | $s_0 = infinite$ , $s_1 = Smallest\ pivot$ |
| 13-15 | $c_i$ = 0 |
| 16 | m = 0, k' = 0, $k_{right} = 0$ , $k_{left} = 0$ |

In this case, the kth smallest element between -infinity to p+1 is not possible ass the number won't be in the range of the bin size. In this case, the bin size is $(s_0, s_1]$.

Hence, the claim that the above algorithm doesn't always return the correct kth smallest element in A is proved.