**CS 630: Graduate Algorithms** Released on 10/13

# Homework 3

*Prof. Charalampos Tsourakakis* *Due to: Friday 5pm, 10/27*

## Instructions

- The homework is due on **Friday 10/27 at 5pm ET**.
- No extension will be provided, unless for serious documented reasons.
- **Despite having two weeks, better start early than late!**
- Unless specified differently, the points are distributed equally among the sub-questions.
- Study the material taught in class, and feel free to do so in small groups, but the solutions should be a product of your own work.
- This is not a multiple choice homework; reasoning, and mathematical proofs are required before giving your final answer.
- If you work with others or utilize any external tools and resources, please make sure to annotate your answers.
- Please submit your work through Gradescope. You can find the access code on Piazza.

## Exercise 1 [10 points]

### (A) Recurrences [8 pts]

Solve the following recurrences, giving your answer in $\Theta$ notation. For each of them, assume the base case $T(n) = 1$ for $n \leq 10$. Show your work.

1. (2pts) $T(n) = T(n-2) + n^4$.
2. (2pts) $T(n) = 3T(n/4) + n$.
3. (2pts) $T(n) = 5T(n-2)$.
4. (2pts) $T(n) = \sqrt{n}\, T(\sqrt{n}) + n$.

### (B) Computing an expectation [2 pts]

Consider a county in which 100,000 people vote in an election. There are only two candidates on the ballot: a Democratic candidate (denoted by $D$) and a Republican candidate (denoted by $R$). As it turns out, this county is heavily Democratic, so 80,000 people go to the polls with the intention of voting for $D$, and 20,000 go to the polls with the intention of voting for $R$. However, the layout of the ballot is somewhat confusing. As a result, each voter, independently and with a probability of $\frac{1}{100}$, votes for the wrong candidate - that is, the one they didn't intend to vote for. (It's worth noting that in this election, there are only two candidates on the ballot.) Let $X$ denote the random variable equal to the number of votes received by the Democratic candidate $D$, when the voting is conducted with this error process.

1. Determine the expected value of $X$, and provide an explanation for your derivation of this value.

## Exercise 2 [10 points]

Consider the following recurrence:

$$T(n) \quad = \quad 2T(n/2) + n \lg n.$$

Let's use a base-case of $T(2) = 2$ and let's assume $n$ is a power of 2. We would like you to solve this recurrence using the "guess and prove by induction" method.

1. (2pts) Try to prove by induction that $T(n) \leq cn \lg n$. Point out where this proof fails.
2. (4 pts) Use the way the above proof failed to suggest a better guess $g(n)$. Explain why you chose this guess and prove by induction that $T(n) \leq g(n)$ as desired.
3. (4 pts) Now give a proof by induction to show that $T(n) \geq c'g(n)$ where $c' > 0$ is some constant and $g(n)$ is your guess from (b). Combining this with (b), this implies that $T(n) = \Theta(g(n))$.

## Exercise 3 [15 points] Collaborators: Nihal Poosa and Anuedha Saha

Consider modifying the min-cut algorithm from class to determine an $s$-$t$ min-cut in an undirected graph. For this task, we're provided with an undirected graph, G, along with two specific vertices, s and t. An s-t cut represents a set of edges that, when removed, disconnects vertex s from vertex t. The goal is to identify the s-t cut with the fewest edges. As the algorithm progresses, vertex s might be merged into a new vertex due to an edge contraction, which we'll refer to as the s-vertex (with s being the initial s-vertex). Likewise, there's a t-vertex. During the contraction procedure, it's essential to avoid contracting any edge that connects the s-vertex and t-vertex.

1. Demonstrate that for certain graphs, the likelihood of this algorithm identifying an s-t min-cut can be extremely low.

## Exercise 4 [10 pts]

You are observing a flow of customers one after another, and you aim to select a random sample of $k$ distinct customers from this flow. However, you encounter several challenges:

- You can only accommodate $k$ customers at any given moment.
- You are unaware of the total number of customers in the flow.
- If you decide not to retain a customer as they pass, you lose the opportunity forever.

Design a strategy so that, once the customer flow ends, you possess a subset of $k$ customers picked uniformly at random from the entire flow. If the total count of customers in the flow is less than $k$, you should retain all of them.

## Exercise 5 [30 points]

Al Gorithm wants to optimize some code for sorting arrays of size 3, since it is in the inner loop of a climate modeling system. Using quicksort, given an array $A = [a, b, c]$, the code will choose one element as the pivot and compare it to the other two. If it was lucky and the pivot was between the other two (e.g., it chose $a$ and we had $b < a < c$) then it is done. Otherwise, it needs to make one more comparison (between the non-pivots) to finish sorting.

In order to precisely describe what is going on, Al writes down a matrix with the rows corresponding to different pivot choices, columns corresponding to different possible locations for the middle (median) element, and filling in the matrix with the total number of comparisons used by that choice of pivot in that situation. Specifically, the matrix is as follows:

| | | Median element is | | |
|---|---|---|---|---|
| | | $a$ | $b$ | $c$ |
| Pivot choice | $a$ | 2 | 3 | 3 |
| | $b$ | 3 | 2 | 3 |
| | $c$ | 3 | 3 | 2 |

$\leftarrow$ total cost for sorting (# comparisons)

1. (6pts) Suppose that based on how these arrays are generated, Al believes there is a 25% chance element $a$ will be the median, a 30% chance element $b$ will be the median, and a 45% chance element $c$ will be the median. In that case, what is the best pivot to use and what is the expected cost for sorting using that pivot?
2. Suppose that Al suspects that these arrays are being generated by his nemesis Doc Chiney. Because of this, Al decides to randomize.
   (a) (4pts) With what probabilities should Al choose pivots $a$, $b$, and $c$ so that no matter what ordering Doc selects, Al's expected sorting cost is as small as possible? (I.e., Al wants

$$\max_{\text{Doc's choices}} \mathbb{E}_{\text{Al's choices}}[\text{sorting cost}]$$

to be as small as possible).

(b) (4pts) What is the expected cost under this distribution?

3. Suppose that Al's processor has special features that result in the following improved matrix (the only change is the upper-left-corner):

Median element is

| Pivot choice | $a$ | $b$ | $c$ | |
|---|---|---|---|---|
| $a$ | 1 | 3 | 3 | $\leftarrow$ total cost with Al's special processor |
| $b$ | 3 | 2 | 3 | |
| $c$ | 3 | 3 | 2 | |

(a) (4pts) *Now* what probabilities should Al use for pivots $a$, $b$, and $c$ so that the worst-case expected cost ($\max_{\text{Doc's choices}} \mathbb{E}_{\text{Al's choices}}[\text{sorting cost}]$) is as small as possible?
Hint: by symmetry, you can assume Al's probabilities on $b$ and $c$ are equal. So, for some value $p$, Al has probability $p$ on $b$, $p$ on $c$, and $1 - 2p$ on $a$.

(b) (4pts) And what *is* Al's expected cost when using those probabilities?

4. Considering the matrix in part (c) above, suppose now *Doc* is going to probabilistically decide which element to make the median.

(a) (4pts) What probabilities should Doc use so that no matter which pivot Al picks, Al's expected cost is as *high* as possible? (Formally,

$$\min_{\text{Al's choices}} \mathbb{E}_{\text{Doc's choices}}[\text{sorting cost}]$$

is as high as possible?)

(b) (4pts) And what is Al's expected cost in this case?

The above is an example of something called a "matrix game" or a "2-player zero-sum game" between players Al and Doc. If you did this correctly, the values you computed in parts c(ii) and d(ii) should be identical. The fact that they are identical is a case of von Neumann's minimax theorem (which we will discuss later in the course), and this value is called the *value* of the game. In class, when we talk about giving "upper bounds" and "lower bounds" for a problem like sorting, we are really talking about upper and lower bounds on the value of the associated game. An upper bound of $O(n \log n)$ means we have an algorithm that guarantees expected cost $O(n \log n)$ no matter what input it is given. A lower bound of $\Omega(n \log n)$ means that no algorithm can do better, which we typically will show by giving an adversarial probability distribution on inputs (an "algorithm for Doc") that is bad for all algorithms.

Also note that while in the case of $n = 3$ we have been examining, we are talking very small differences in performance between the various options, as we saw in class the gap becomes quite substantial for large values of $n$. E.g., using the first element as pivot has a worst-case of $\Omega(n^2)$, whereas randomizing over pivots guarantees any input has expected cost $O(n \log n)$.

## Exercise 6 [15 pts] Collaborator : Aneesha Saha

Suppose that we wanted a data structure to maintain a set of elements and quickly query for the median of the set (in case of even sized set of elements, call either of the two middle elements a median). We want to support the following operations with the specified worst-case costs:

- **Build(n):** Build a data structure consisting of $n$ elements in time $O(n)$.
- **makeDS(x):** Build a data structure consisting of 1 element in time $O(1)$.
- **findMedian(D):** Find the median of $D$ in time $O(1)$.
- **insert(x,D):** Insert element $x$ into a structure $D$ consisting $n$ elements in time $O(\log n)$.

Build a data structure that supports all the above operations.

## Exercise 7 Coding DOULION [10 pts]

For the description of this problem, see the class' git repo.

# Exercise 1 - Part A

1. Solve $T(n) = T(n-2) + n^4$

$T(n) = T(n-2) + n^4$

Let's expand this:

$$T(n) = T(n-4) + (n-2)^4 + n^4$$

$$= T(n-6) + (n-4)^4 + (n-2)^4 + n^4$$

$$= T(n-8) + (n-6)^4 + (n-2)^4 + n^4$$

$$= T(n-10) + (n-8)^4 + (n-6)^4 + (n-2)^4 + n^4$$

$$\vdots$$

$$= T(n-2k) + (n-2(k-1))^4 + (n-2(k-2))^4 \dots n^4 \quad \text{—— (1)}$$

We know that:

$T(n) = 1$ for $n \leq 10$

Therefore:

$$n - 2k = 10$$
$$\Rightarrow k = \frac{n-10}{2}$$

Substituting k in equation (1)

$$T(n) = T\left[n - 2\left(\frac{n-10}{2}\right)\right] + \left[n - \left[2\left(\frac{n-10}{2}\right) - 1\right]\right]^4 + \left[n - \left[2\left(\frac{n-10}{2}\right) - 2\right]\right]^4 \dots n^4$$

$$= T(10) + [11]^4 + [12]^4 \dots n^4$$

$$= 1 + [11]^4 + [12]^4 \dots n^4$$

$$\leq 1 + O(n^5) \quad \left\{ \because \text{there are } (n-10) \text{ terms with max value } n^4 \therefore \text{sum} = (n-10)n^4 = O(n^5) \right\}$$

$$\leq O(n^5)$$

Now that we have proved the upper bound. Let's try to find $\Omega(T(n))$

Let's consider the same inequality:

$$T(n) \geq 1 + [11^4 + 12^4 + \dots + n^4] \quad \text{—— (2)}$$

Now in this series, let's consider a part of it

$$\left(\frac{n}{2}\right)^4 + \left(\frac{n}{2}+1\right)^4 + \ldots n^4$$

using the similar reason as above, we get the

$$\text{sum} \geq \left(\frac{n}{2}\right)^5$$

$$\therefore T(n) \geq 1 + \left(\frac{n}{2}\right)^5$$

$$\Rightarrow T(n) \geq \Omega\left[\left(n/2\right)^5\right]$$

$$\therefore T(n) = \Theta(n^5)$$

---

2. Solve $T(n) = 3T(n/4) + n$

We can solve this recurrence relation using master's theorem:

So, $a = 3$
$b = 4$
$f(n) = n$

We know that:

$$\log_4 3 = \log 3/2 = 0.79 < 1 \quad \{\because f(n) = n^1\}$$

$\therefore$ we can see that

$$f(n) > n^{\log_b a}$$

This resembles the third case of master's theorem, where:

$$f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$$

here, $\varepsilon = 0.21$

and,

$$a f(n/b) \leq c f(n) \quad \forall\, c < 1$$

$$3 f(n/4) \leq c f(n)$$

$$\frac{3n}{4} \leq cn$$

We know that this inequality holds for $c<1$. {Eg: when $c = 4/5$}

$\therefore T(n) = \Theta(f(n)) = \Theta(n)$

---

3. Solve $T(n) = 5 \cdot T(n-2)$

Let's expand this:

$$T(n) = 5\left[5 \cdot T(n-2-2)\right] = 5^2 T(n-4)$$
$$= 5^2\left[5\,T(n-4-2)\right] = 5^3 T(n-6)$$
$$\vdots$$
$$= 5^k\, T(n-2k)$$

We know that:

$$T(n) = 1 \quad \forall\ n \leq 10$$

$\therefore\ n - 2k = 10$

$\Rightarrow k = n-10/2$

Substituting this in the recurrence relation, we get:

$$T(n) = 5^{(n-10)/2}\, T(10)$$
$$= 5^{(n-10)/2} = \Theta(5^{n/2})$$

---

4. Solve $T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n$

$\hookrightarrow\ n^{1/2} \cdot T(n^{1/2}) + n$

Let's expand this:

$$T(n) = n^{1/2} \cdot \left[n^{1/2^2}\, T(n^{1/2^2}) + n^{1/2}\right] + n = n^{3/4}\ T(n^{1/2^2}) + 2n$$
$$= n^{1/2+1/2^2}\left[n^{1/2^3}\, T(n^{1/2^3}) + n^{1/2^2}\right] + 2n = n^{7/8}\ T(n^{1/2^3}) + 3n$$
$$\vdots$$
$$= n^{2^k-1/2^k}\, T(n^{1/2^k}) + kn \qquad \text{— (1)}$$

We know that

$$T(n) = 1 \quad \forall\ n \leq 10$$

$\therefore\ n^{1/2^k} = 2$

$\frac{1}{2^k} \log n = 1$

$\log n = 2^k$

$\Rightarrow k = \log(\log n)$

Let's substitute this back to equation (1):

$$T(n) = n^{\log n - 1 / \log n}\, T(2) + \log(\log n) \cdot n$$

$$= n \cdot n^{-1/\log n} + \log(\log n)\, n$$

$$= n \left[ n^{-1/\log n} + \log(\log(n)) \right]$$

Since $\log(\log(n))$ is the dominating term, we get:

$$T(n) = \Theta\left(n \log(\log(n))\right)$$

---

## Exercise 1: Part B

People who intended to vote for D = 80,000
People who intended to vote for R = 20,000

Pr(voting the wrong person) = 1/100

Pr(voting the right person) = 1 - Pr(voting the wrong person)

$= 1 - 1/100$

$= 99/100$

Expected number of people who voted for D = People who intended to vote for D and voted for D + People who intended to vote for R but voted for D

$= 80{,}000 \times \frac{99}{100} + 20{,}000 \times \frac{1}{100}$

$= 79400$

# Exercise 2

1. Given

$T(n) = 2T(n/2) + n\lg n$

$T(2) = 2$

Let's try to prove using induction that $T(n) \leq cn\lg n$

Base case: $T(2) = 2$ {given}

$T(2) \leq c \cdot 2\lg 2$

$2 \leq c \cdot 2 \Rightarrow c \geq 1$

∴ for all positive integeral values of c, the base case holds.

Induction hypothesis: Let's assume that $T(k) \leq ck\lg k \;\forall\; k < n$

Inductive step: $T(n) \leq c \cdot n\lg(n)$ } to prove

LHS: $T(n) = 2T(n/2) + n\lg n$

$\leq 2\left[c \cdot \frac{n}{2} \cdot \lg(n/2)\right] + n\lg(n)$ {using inductive hypothesis}

$\leq c \cdot n[\lg n - \lg(2)] + n\lg n$

$\leq cn\lg n - cn + n\lg n$

Let's compare LHS and RHS

$cn\lg n - cn + n\lg n \leq cn\lg n$

Simplifying this inequality, we get

$n\lg n \leq cn$

We can see that for small values of n this inequality will hold but for as n increases the term $n\lg n$ will grow faster than n. This means the inequality will not hold for greater values of n

Hence, we proved that $T(n) \leq cn\lg n$

---

2. Guess: $g(n) = cn\lg^2 n$

Because we saw that the term $n\lg n$ in recurrence relation can grow faster than $cn\lg n$ for some values of n.

Proving this by induction:

Base case: $n=2$

$$T(2)=2 \leq c(2 \cdot \lg^2 \cdot 2) \leq 2c$$

$\therefore$ Base case holds for $c \geq 1$

Induction hypothesis: Assume $T(k) \leq ck\lg^2 k \quad \forall k<n$

Inductive step:

$$T(n) = 2T(n/2) + n\lg n$$

$$\leq 2\left[c\frac{n}{2}\lg^2\frac{n}{2}\right] + n\lg n \quad \{\text{using inductive hypothesis}\}$$

$$\leq cn\lg^2(n/2) + n\lg n$$

$$\leq cn\left[\lg^2 n - 2\lg n \lg 2 + \lg^2 2\right] + n\lg n$$

$$\leq cn\left[\lg^2 n - 2\lg n + 1\right] + n\lg n$$

$$\leq cn\lg^2 n - 2cn\lg n + cn + n\lg n$$

$$\leq cn\lg^2 n - cn\lg n + cn + n\lg n - cn\lg n$$

$$\leq cn\lg^2 n + cn\left[1 - \lg n\right] + n\lg n\left[1 - c\right]$$

$cn[1-\lg n]$: decreasing term

$$\therefore T(n) \leq cn\lg^2 n + [1-c]n\lg n$$

Now we can see that this inequality is always less than equal to $cn^2\lg n$ for $c \geq 1$.

---

3. To prove: $T(n) \geq c' g(n)$

$$\Rightarrow T(n) \geq c' n\lg^2 n \quad \forall c>0$$

Base case: ~~$\forall$~~ $n=2$

$$T(2) \geq c' 2\lg^2 2$$

$$2 \geq c' \cdot 2$$

$\Rightarrow$ Base case holds for $c' \leq 1$

Induction hypothesis: Assume $T(k) \geq c' k \lg^2 k \quad \forall k < n$

Inductive step:

$$T(n) = 2T(n/2) + n\lg n$$

$$\geq 2\left[c' \frac{n}{2} \lg^2 \frac{n}{2}\right] + n\lg n \quad \{\text{using induction hypothesis}\}$$

$$\geq c'n\left[\lg^2 n - 2\lg n + 1\right] + n\lg n$$

$$\geq c'n\lg^2 n - 2c'n\lg n + n\lg n + c'n$$

$$\geq c'n\lg^2 n + n\lg n\left[1 - 2c'\right] + c'n$$

↳ for $T(n) \geq c'n\lg^2 n$, these two terms should be $\geq 0$

$\therefore n\lg n\left[1-2c'\right] + c'n \geq 0$

↳ we can see that if this term is negative it will dominate $c'n$ & $T(n) \leq c'n\lg^2 n$ We don't want that

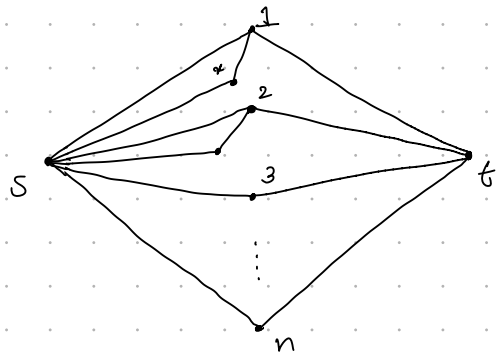
$\therefore n\lg n[1-2c'] \geq 0$

$\Rightarrow c' \leq \frac{1}{2}$

$\therefore T(n) \geq c' \cdot n\lg n \quad \forall \; 0 \leq c \leq 1/2$

# Exercise 3

Let's consider a family of graphs, which would look like this:




We can see that there are total of 4n edges in this graph.

→ n edges connecting the vertices in the middle with t

→ n edges connecting the vertices in the middle with s

→ n edges connecting s with the vertices like x

→ n edges connecting vertics in the middle with vertices like x

Now let's consider the possible cuts which can lead to s-t min cut:

→ We can cut all the edges that are coming out / going into {not sure ∴ it is an undired graph} t. This leads to total n edges in the cut.

→ Now consider the scenario where we cut all the edges coming out / going into s. This will lead to 2n edges in the cut.

→ Now, we are cutting the middle edges ie the edges connecting vertices like x with the middle vertices (1...n). In this case, we would need another set of edges to separate s and t. This could be either the edges directly connmiddle vertices (1...n) with s or with t. This leads to 2n edges in the cut.

→ We can also consider the scenario, where we are cutting only one set of edges coming out / going into s. Either the ones that are connected to middle verticds (1...n) or the ones connecting to vertices like x. In both the cases it leads to 2n edges in the cut.

As we want a cut with minimum number of edges Let's go ahead with the one where we cut all the edges coming out / going into t.

∴ the number of edges in the cut = n

Now, let's calcute the likelihood of contracting any one such edge [say 1-t]

$$Pr[\text{contracting } 1\text{-}t] = \frac{n}{4n}$$

Now let's calculate the probability of contracting (2-t) after this

$$Pr[\text{contracting } 2\text{-}t \text{ after } 1\text{-}t] = \frac{n-1}{4n-1}$$

Similarly, we can understand that the probability of contracting the last edge [n-t] after contracting all the edges like [1-t, 2-t ... (n-1)-t] is $1/4n-(n-1)$

$\therefore$ Pr [successfully finding a min cut using these edges]

$$= \frac{n}{4n} \cdot \frac{n-1}{4n-1} \cdots \frac{1}{4n-(n-1)}$$

$$= \prod_{i=0}^{n-1} \frac{n-i}{4n-i}$$

$$= \prod_{i=0}^{n-1} 1 \Big/ \frac{4n-i}{n-i}$$

$$= \prod_{i=0}^{n-1} 1 \Big/ \frac{4n-4i+3i}{n-i}$$

$$= \prod_{i=0}^{n-1} 1 \Big/ 4 + \frac{3i}{n-i}$$

$$< \prod_{i=0}^{n-1} 1/4$$

$$< \frac{1}{4^n}$$

# Exercise 4

The strategy to select k customers given the constraints:

1. One by one select all the customer till either:

   → You have selected k customers

   → You have reached the end of flow of customer. In this case stop selecting after this

2. For every next customer: {say index of this customer is i}

   → Generate a random number between 1 and i. Say this number is x.

   → If $1 \leq x \leq k$, then replace the $x^{th}$ customer from the selected sample with current customer from the customer flow.

   → Else: move on to next customer.

The correctness of this algorithm lies in the fact that every customer has an equal probability of ending up in the final sample. At each & every step customer has $k/i$ probability of being in the sample.

Let's prove this using induction

Suppose $P(i)$ is the probability that customer at index $i$ is in the final sample after processing $i$ customers

Base case: $i=k$
Initially, we select the first k customers, so for $i=k$; $Pr(k) = 1$. As these k customers are definately in the sample.

Induction hypothesis:
Assume that for the $j^{th}$ customer $(k \leq j \leq n)$. $P(j)$ is same for all customers up to and including the $j^{th}$ customer.

Inductive step:
when $i = j+1$
We basically want to prove that $P(j+1)$ is same for all customers upto and including the $(j+1)^{th}$ customer.

Now in this case when you generate the random number in step two, there can be two scenarios:

1. When $x \leq k$:
   In this case $j+1$ customer replaces one of the customer in the selected sample.

Customer $(j+1)$ is basically selected with a probability $1/j+1$, which is equal to the probability of sampling all customers in the sample upto that point.

2. When $x > k$

In this case, we don't change the selected sample. And probability of that happening is $(j+1-k)/(j+1)$, this is also same for all the customers upto customer $j$.

This proves that the probability of choosing a customer is same for all customers upto that customer.

# Exercise 5

1. Let's find the expected cost for sorting using all the pivots (a, b & c):

$$E[\text{cost of sorting when pivot is a}] = Pr[\text{a being the median}] * \text{cost of sorting} + Pr[\text{a not being the median}] * \text{cost of sorting in that case}$$

$$= \frac{25}{100} \times 2 + \frac{75}{100} \times 3$$

$$= 2.75$$

$$E[\text{cost of sorting when pivot is b}] = Pr[\text{b being the median}] * \text{cost of sorting} + Pr[\text{b not being the median}] * \text{cost of sorting in that case}$$

$$= \frac{30}{100} \times 2 + \frac{70}{100} \times 3$$

$$= 2.70$$

$$E[\text{cost of sorting when pivot is c}] = Pr[\text{c being the median}] * \text{cost of sorting} + Pr[\text{c not being the median}] * \text{cost of sorting in that case}$$

$$= \frac{45}{100} \times 2 + \frac{55}{100} \times 3$$

$$= 2.55$$

Therefore, the best choice for pivot should be c as the expected cost for sorting is minimum in that case.

---

2. a. As A1 chooses to randomize, how he chooses the pivot because of the adversary Doc. He should choose the pivot ~~with~~ with equal probability ie (1/3)

$\therefore Pr[\text{a being chosen as pivot}] = Pr[\text{b being chosen as pivot}] = Pr[\text{c being chosen as pivot}]$

$= 1/3$

2. b.

Now, let's find the expect cost of sorting in each case:

$$E[\text{cost of sorting when a is pivot}] = \frac{1}{3} * 2 + \frac{2}{3} * 3$$

$= 8/3$

$E[\text{cost of sorting when b is pivot}] = \frac{1}{3} * 2 + \frac{2}{3} * 3$

$= 8/3$

$E[\text{cost of sorting when c is pivot}] = \frac{1}{3} * 2 + \frac{2}{3} * 3$

$= 8/3$

---

3. a. Let p be the probability b or c being the median

$\Rightarrow 1-2p$ is the probability of a being the median

Let's find the expected cost in each case:

$E[\text{cost of sorting when pivot is a}] = 1*(1-2p) + 3*2p$

$= 1-2p+6p$

$= 1+4p$

$E[\text{cost of sorting when pivot is b}] = 2*p + (1-p)*3$

$= 2p+3-3p$

$= 3-p$

$E[\text{cost of sorting when pivot is c}] = 2*p+(1-p)*3$

$= 3-p$

We basically want to find p such that $\max(3-p, 1-4p)$ is minimized.

This is possible when $1+4p = 3-p$

$\Rightarrow p = 2/5$

b. $E[\text{cost of sorting when pivot is a}] = 1+4p$

$= 13/5$

$E[\text{cost of sorting when pivot is b}] = 3-p = 13/5$

E[cost of sorting when pivot is c] = $3-p = 13/5

---

4.a In this case, we basically want to maximize the min(1+4p, 3-p)

And this only happens when $1+4p = 3-p$

$\Rightarrow p = 2/5$

b. Similarly how we did for 3.b.

E[cost of sorting when pivot is a] = $1+4p$

$= 13/5$

E[cost of sorting when pivot is b] = $3-p = 13/5$

E[cost of sorting when pivot is c] = $3-p = 13/5$

# Exercise 6

We can achieve all the given operations using two heaps: one max and one min.

Let's see how we can do that:

1. Build(n)
   a. Initialize one max heap and one min heap.

   b. Insert the first element into any heap with probability 1/2 and set it as median

   c. for rest of the elements in the set:
      → if current element ≤ median:
         - Insert current element to max heap
      → else:
         - Insert current element to min heap

      → Update the median:
         - if n is even:
            median = root (max(height(minheap), height(maxheap)))
         - else:
            median = root (max heap)

   d. After this ensure that both the heaps are balanced. Else balance them. Also while inserting, make sure to heapify the heaps if required.

Time complexity of Build(n):

We know for inserting an element into a heap requires traversing the height of the heap which is $\log n$.

Now let's consider time required to insert:

$1^{st}$ element = $\log 1 = O(1)$
$2^{nd}$ element = $\log 2 = O(1)$
⋮

This means time complexity of Build(n) = $O(1) + O(1) \ldots$ n times $= O(n)$

2. makeDS(x)
   a. Initialize one max heap and one min heap.

   b. Insert x into any heap with probability 1/2 and set it as median

   Time complexity:

We know for inserting an element into a heap requires traversing the height of the heap which is $\log n$.

∴ time required to insert just 1 element is $\log(1) = O(1)$

3. findMedian(n):

a. if n is even:
   median = root (max(height(minheap), height(maxheap)))
b. else:
   median = root (max heap)
c. Return median

Time complexity

We can see that there are no loops. We just need to compute some basic conditions like: if n is odd or even, which heap has maximum height etc.

All these operations only require $O(1)$ time.

4. Insert(x, n):

a. If size(maxheap) = size(minheap) = 0:
   insert x in any of these heaps with probability 1/2.

b. else if size(maxheap) = 0 or $x \leq$ median:
   insert x to max heap.

c. else
   insert x to minheap

d. if size(maxheap) > size(minheap) + 1:
   delete the root of max heap and insert it into minheap

e. else size(minheap) > size(maxheap) + 1
   delete the root of min heap and insert it into maxheap

Time complexity

To insert an element into either of the heaps would require traversing these heaps and then inserting.
We know that height of these heaps are $\log(n)$.

∴ time complexity of inserting an element is $O(\log n)$