COMS 311: Homework 5
Due: Oct 28, 11:59pm
Total Points: 100

Late submission policy. If you submit by Oct 29, 11:59PM, there will be 20% penalty. That is, if your score is x points, then your final score for this homework after the penalty will be $0.8 \times x$. Submission after Oct 29, 11:59PM will not be graded without explicit permission from the instructors.

Submission format. Your submission should be in pdf format. Name your submission file:
<Your-net-id>-311-hw5.pdf. For instance, if your netid is asterix, then your submission file will be named asterix-311-hw5.pdf.

If you are planning to write your solution and scan+upload, please make sure the submission is readable (sometimes scanned versions are very difficult to read - poor scan quality to blame).

If you plan to snap pictures of your work and upload, please make sure the generated pdf is readable - in most cases, such submissions are difficult to read and more importantly, you may end up submitting parts of your work.

If you would like to type your work, then one of the options you can explore is latex (Overleaf).

Rules for Algorithm Design Problems.

1. Unless otherwise mentioned, for all algorithm design problems, part of the grade depends on runtime. Better runtime will lead to higher grade.
2. Unless otherwise mentioned, for all algorithm design problems, you are required to write some justification of why your algorithm correctly addresses the given problem.
3. You will write pseudo-code for your algorithm. It should not be some code in a specific programming language (e.g., Java, C/C++, Python).
4. You can use any operation already covered in class-lectures in your solution without explicitly writing the algorithm. However, you need to use them appropriately (i.e., indicate the inputs to and outputs from these algorithms). For instance, you can use heapifyUp with input as the index of a heap element and you do not need to write the code for heapifyUp.

Learning outcomes.

1. Recurrence relations
2. Divide and Conquer
3. Greedy Algorithm

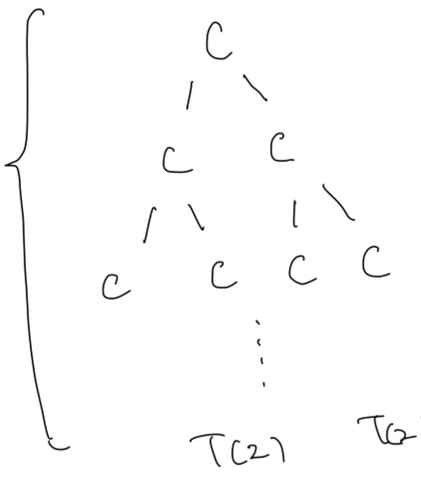
Problems 1–4 are 25pts each.

1. Solve the following recurrences using recurrence tree method (application of any other method will not be graded). For both problems assume $T(2)$ = constant.

   (a) $T(n) = 2T(n/2) + c$

$\therefore T(2) = c$

$h = ((\log_2 n) - 1$



level 1 : $2^0 c$

level 2 : $2^1 c$

level 3 : $2^2 c$

level $h = 2^{h-1} c$

$$\therefore T(n) = c\,\frac{2^{((\log_2 n)-1)} - 1}{2-1}$$

$$= c\, 2^{(\log_2(\frac{n}{2}))} - 1$$

$$= c\,\frac{n}{2} - 1$$

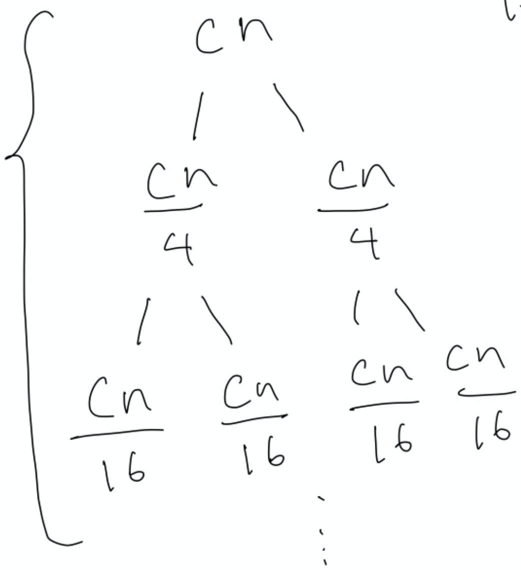
$\therefore O(n)$

n

(b) $T(n) = 2T(n/4) + cn$

$\because T(2) = c$

$h = (\log_2 n) - 1$


cn
$\frac{cn}{4}$ $\frac{cn}{4}$
$\frac{cn}{16}$ $\frac{cn}{16}$ $\frac{cn}{16}$ $\frac{cn}{16}$
$\vdots$

level 1 : $\frac{cn}{2^0}$

level2 : $\frac{cn}{2^1}$

level3 : $\frac{cn}{2^2}$

level h : $\frac{cn}{2^{h-1}}$

$$\therefore T(n) = cn \frac{1 - \left(\frac{1}{2}\right)^{(\log_2 n) - 1}}{1 - \frac{1}{2}}$$

$$= cn\, 2 \left(1 - \left(\frac{1}{2}\right)^{\log_2\left(\frac{n}{2}\right)}\right)$$

$$= cn\, 2 \left(1 + \frac{n}{2}\right)$$

$\therefore O(n^2)$ //

2. Suppose you're consulting for a bank that's concerned about fraud detection, and they come to you with the following problem.

   They have a collection of *n* bank cards that they've confiscated, suspecting them of being used in fraud. Each bank card is a small plastic object, containing a magnetic stripe with some encrypted data, and it corresponds to a unique accountin the bank. Each account can have many bank cards corresponding to it, and we'll say that two bank cards are equivalent if they correspond to the same account.

   It's very difficult to read the account number off a bank card directly, but the bank has a high-tech "equivalence tester" that takes two bank cards and, after performing some computations, determines whether they are equivalent. Their question is the following: among the collection of *n* cards, is there a set of more than *n/*2 of them that are all equivalent to one another?

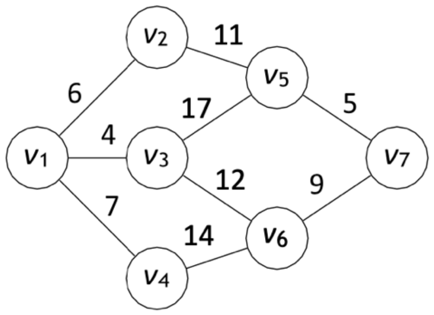
   Assume that the only feasible operations you can do with the cards are to pick two of them and plug them in to the equivalence tester. Write an algorithm to determine the answer to their question. Justify the correctness of your algorithm and derive its runtime.

```
*Assuming cards are in an array, 'cards'.
*The function of equivalence tester is called 'isEquivalent()' in the following code
*Assuming there are more than 1 cards.

Pick one card from the set and hold as 'cur'
counter = 1;
for(i=1;i<n;i++){
        //check if cur is equivalent to the card picked up from the stack of cards
        if(cur.isEquivalent(card[i]){
                counter++;
        }
        else{
                counter--;
                if(counter==-1){
                        cur=card[i];
                        counter=1;
                }
        }
}

Compare all the cards with the card currently held and count how many cards are equivalent to
one another.

Return if the number is bigger than n/2 or not;
```

Runtime:
In the first for-loop, it finds the card appears most by going through each card only once
->O(n)
Compare all the cards with the card currently held and count how many cards are equivalent to one another.
->O(n)
∴O(n)+O(n)=O(n)

Correctness:
This algorithm works because it is going to check if more than half of the cards are the same. It means the rest of the cards other than the card is less than n/2. In the first path, it finds the card can be majority if there is a majority. After that, it confirms if it is a majority in the set.

3. Consider the following graph



(a) Apply the Dijkstra algorithm with the source vertex v_1 and show the computation of *d*-value for each of the vertices in the tabular format as illustrated below.

Iteration	v_1	v_2	v_3	v_4	v_5	v_6	v_7
0	0	∞	∞	∞	∞	∞	∞
1	**0**	6_{v1}	4_{v1}	7_{v1}	∞	∞	∞
2		6_{v1}	**4_{v1}**	7_{v1}	21_{v3}	16_{v3}	∞
3		**6_{v1}**		7_{v1}	17_{v2}	16_{v3}	∞
4				**7_{v1}**	17_{v2}	16_{v3}	∞
5					17_{v2}	**16_{v3}**	25_{v6}
6					**17_{v2}**		22_{v5}
7							**22_{v5}**

*The number shaded indicates the vertex is currently extracted int the iteration

(b) Apply the Prim's algorithm (start the algorithm at vertex v_1) and show the computation of *d*-value for each of the vertices in the tabular format as illustrated above.

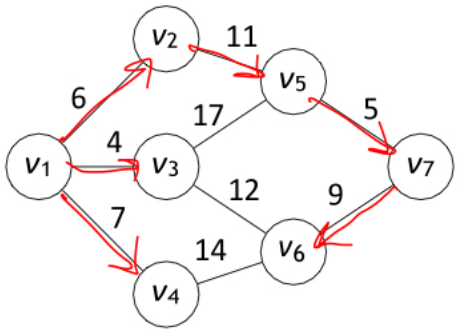
Iteration	v_1	v_2	v_3	v_4	v_5	v_6	v_7
0	0	∞	∞	∞	∞	∞	∞
1	**0**	$6v_1$	$4v_1$	$7v_1$	∞	∞	∞
2		$6v_1$	**$4v_1$**	$7v_1$	$17v_3$	$12v_3$	∞
3		**$6v_1$**		$7v_1$	$11v_2$	$12v_3$	∞
4				**$7v_1$**	$11v_2$	$14v_4$	∞
5					**$11v_2$**	$14v_4$	$5v_5$
6						$9v_7$	**$5v_5$**
7						**9_{v7}**	

*The number shaded indicates the vertex is currently extracted int the iteration

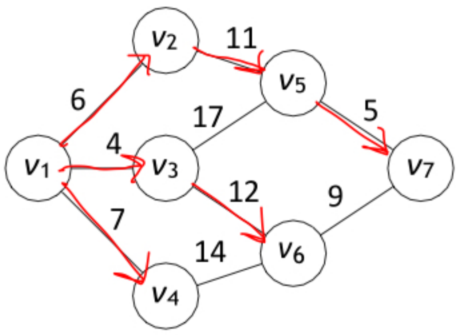
4. Prove/disprove the following claims:

   (a) Applying Prim's algorithm from a vertex *v* in a weighted undirected graph results in a MST that includes all the shortest paths from the vertex *v*.

   This is the graph generated by Prim's algorithm in Question 3…i



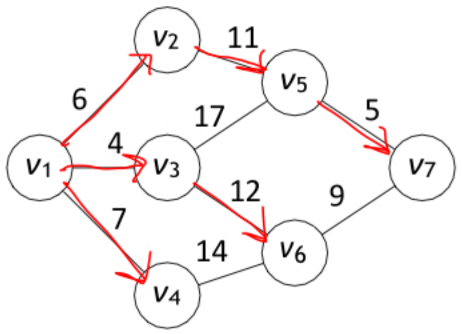
   This is the graph generated by Dijkistra's algorithm to find the shortest path to the node from v_1 in Question 3…ii



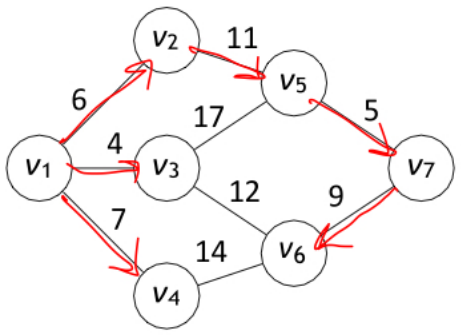
∵i and ii
The MST generated by Prim's algorithm does not include the shortest path to v6 from v1.
∴<u>The statement is false.</u>

(b) Applying Dijkstra's algorithm with source vertex *v* in a (positive) weighted undirected graph and generating a tree using the "parent"-relation results in a MST.

This is the graph generated by Dijkstra's algorithm to find the shortest path to the node from v_1 in Question 3…i



This is the graph generated by Prim's algorithm in Question 3…ii



∵i and ii
The MST generated by Dijkstra's algorithm does not result in MST.
<u>∴The statement is false.</u>