CS 310 'Algorithms' **Assignment-3**
Due date: Tuesday, November 19, 2019 at 11:55 PM. [Total Marks = 56]

IMPORTANT: Read the instructions at the end of this document and follow the naming conventions.

Stable Matching

Q1. **[10 marks]**

The archeological department is organizing an excavation and there are n teams participating in it. This excavation is spread over 'n' different locations and the teams move from one location to another after finishing work at the previous location. The organizers want that no two teams should visit the same location at the same time. They have made different visiting schedules for these teams. Due to lack of funds, they want to stop the movement of the teams and assign each of them to one location for a year. Remember that the teams will be moving according to their schedules and we want to find where each of the teams should finally stop, while fulfilling the following conditions:

1. No two teams can be at the same location at the same time. For example, if a team T_1 has chosen location L_3 as its final stopping destination then no other team can visit L_3 after that.
2. Each team should be assigned a different final destination.

You have to do the following:

1. Design an **efficient** C/C++ algorithm that takes as input the location schedules for each of the teams and outputs a final destination for each of the teams, while fulfilling the conditions mentioned above. [8]
2. In the comments at the beginning of your code, describe how your algorithm works and terminates. **What is the running time of your algorithm?** Your should include clear comments that explain your algorithm's time. ***Hint:*** *Use your knowledge of the stable Gale-Shapley matching algorithm to solve this problem. You can think of these schedules as location preference lists for the teams.* [2]

EXAMPLE: Suppose following is the time schedule for the teams. This schedule is given to you as input.

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6
Team 1 (T_1)	L_1	__	L_2	__	L_3	__
Team 2 (T_2)	__	L_1	__	L_2	__	L_3
Team 3 (T_3)	L_2	__	L_3	__	L_1	__

The correct final destinations of each of the teams is (L_1,T_3), (L_2,T_2), (L_3,T_1).
As you can see in the example, the teams are moving from one location to another according to the schedule prepared by the organizers. The dashes ' _' indicate that the teams are

travelling (i.e. in transit) on that day. No two teams are allowed to be at the same location on the same day (this condition is ensured in the schedule).

Each of the 'n' teams have to select exactly one of the 'n' locations as their final destination. Again, no two teams can select the same final destination. In the above example, T1 has chosen L3 as its final destination on Day 5. Now, no other team can go to location L3 on or after Day 5. You can see T2 stops at L2, because it cannot go past L2 towards L3 because T1 is already stopping there.

Your code will read input from a text file. The format of the file is as follows. The first line contains the value of ‘n’, followed by the schedule of each of the teams. The input of the above example is shown below.

Input :
n 3
T1 : L1 - L2 - L3 -
T2 : - L1 - L2 - L3
T3 : L2 - L3 - L1 -

Output:
Final Destinations: L1 T3, L2 T2, L3 T1

Divide and Conquer

Q2. **[10 marks]**

A group of scientists are investigating the effect of a new drug on boosting immune cells in an organism. They have set up a controlled experiment where they count the number of immune cells on each day. The experiment takes ‘n’ days to complete.
Suppose following is the result of their measurements for an experiment that ran for n=8 days.

Days	Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
No of immune cells	34	56	20	23	16	32	9	30

If on any day ‘x’ the number of immune cells fall below half the count on any of the previous days then that indicates a failed trial. We need to count the number of failed trials. For example, in the above table, on Day-2, the count has fallen below half that of Day-1. This is counted as 1 failed trial. On Day-4 the count has fallen below half of the counts of Day-0 and Day-1. This is counted as 2 failed trials. On Day-6 the count has fallen below half of those of Day-0, Day-1, Day-2, Day-3 and Day-5. This means 5 failed trials. The total failed trials in the above experiment are 1+2+5=8.

1. Code an **efficient** Divide and Conquer algorithm in C/C++ that reads the experiment data and prints the number of failed trials. See I/O format below. [8+2]

2. The time complexity of your algorithm should not be more than $O(n\log_2 n)$.

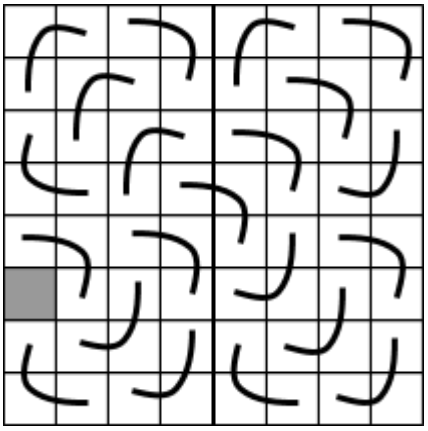
Input format:
n 8
34 56 20 23 16 32 9 30

Output format:
Failed Trials 8
(2,1)
(4,0) (4,1)
(6,0) (6,1) (6,2) (6,3) (6,5)

Note: (4,1) means that count on Day-4 was less than half the count on Day-1.

Q3. **[11 marks]**

In this question you will implement a board game using the Divide and Conquer approach. You are given an **nxn** square board, where n is a power of 2. **One** of the squares on the board is blank and you have to fill the remaining squares with boomerangs. Each boomerang occupies **three** squares that form a **right angle**. See the illustration below. The greyed square is blank and all the remaining squares should be completely filled with boomerangs. No boomerangs will share any squares.



1. Code an **efficient** algorithm in C/C++ that reads the value of 'n' and the row and column indices of the blank square. Your program should then fill all the remaining squares with boomerangs. You will use integers to represent boomerangs as shown in the nxn matrix below, where each boomerang is assigned a **different** number. The colors are for illustration only. [8]

1	1	2	2	4	4	5	5
1	3	3	2	4	6	6	5
11	3	10	10	8	8	6	7
11	11	10	9	9	8	7	7
12	12	13	13	9	14	15	15
	12	20	13	14	14	18	15
21	20	20	19	17	18	18	16
21	21	19	19	17	17	16	16

2. Give a clear description of your algorithm and data structures used to implement it in the comments at the beginning of your code. **What is the recurrence relation of your algorithm? What is the running time of your algorithm?** Your should include clear comments that explain your algorithm's time complexity. [3]

Input format:
n 8
(5,0)

Note: (5,0) means the blank is in fifth row and 0th column. The row and column indices start from zero.

Output format:
1 (0,0) (0,1) (1,0)
2 (0,2) (0,3) (1,3)
3 (1,1) (1,2) (2,1)
…

Note: 2 (0,2) (0,3) (1,3) means the boomerang represented by number 2 is placed at indices (0,2) (0,3) (1,3) in the matrix.

Q4. **[13 marks]**

You are given a **complete binary tree** of height h and $n=2^h$ leaves. Each vertex in the tree has an integer value associated with it. **We have numbered the leaves from x_1 to x_n starting from left to right.** An ancestor of a vertex is its parent, grandparent etc., up to the root of the tree. We define the following:
Ancestry(x_i) of a leaf nodes as the **set of vertices** including x_i and all its ancestors.
Ancestry(x_i, x_j) of a pair of leaves is defined as **union** of their respective ancestries, i.e.,
Ancestry(x_i) U Ancestry(x_j)

The *value* of an Ancestry is defined as the sum of integer values associated with the nodes in that set. The following examples illustrate this for the binary tree shown in Figure-1 below.

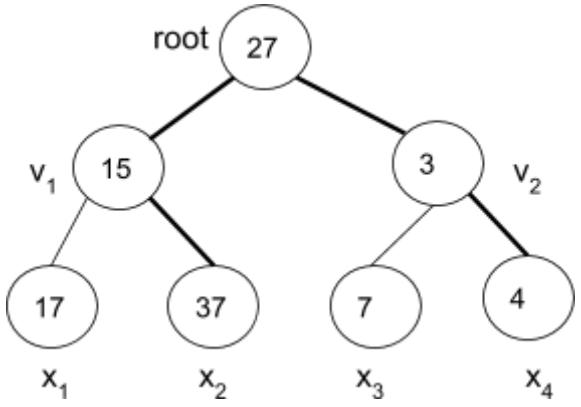


Figure 1

Example 1:
Ancestry(x_2) = {x_2, v_1, root}
Ancestry(x_4) = {x_4, v_2, root}
Ancestry(x_2,x_4) = Ancestry(x_2) U Ancestry(x_4) = {x_2, v_1, root, v_2, x_4}

Value of Ancestry(x_2) = 37+15+27
Value of Ancestry(x_4) = 4+3+27
Value of Ancestry(x_2,x_4) = 37+15+27+3+4 = 86

Example 2:
Ancestry (x_1) = {17, 15, 27}
Ancestry (x_2) = {37, 15, 27}
Value of Ancestry (x_1, x_2) = 96 /* **NOTE:** 17+37+15+27 = 96. Common ancestors v_1 and root are only counted ***once*** since it is a ***union*** of sets. */

You have to describe a **Divide and Conquer (D&C)** algorithm that finds **two** leaves x_i and x_j such that the **Value** of Ancestry (x_i,x_j) is **maximum**.

1. Code an **efficient** algorithm in C/C++ that solves the above problem. Your code will read the value of the height of the complete binary tree 'h' from the input file. This is followed by 2n-1 values associated with the nodes of the tree (note that the number of leaves = n = 2^h and the number of internal nodes is n-1). The node values are listed in breadth-first order from left to right, starting from the root (see input format below).

   Your code will then run the D&C algorithm on the tree to find x_i and x_j that maximize the value of Ancestry(x_i, x_j). Your algorithm should print the following as output: the nodes in Ancestry (x_i), nodes in Ancestry (x_j) and Value of Max Ancestry (x_i, x_j). See output format below. [10]

2. Give a clear description of your algorithm and data structures used to implement it in the comments at the beginning of your code. **What is the recurrence relation of**

your algorithm? What is the running time of your algorithm? Your should include clear comments that explain your algorithm's time complexity. [1]

3. You algorithm should run for large values of 'n'. The above example is only for illustration purposes.
4. You will get partial credit if you design a $O(n\log_2 n)$ D&C algorithm. It is possible to design a D&C algorithm for this problem that runs in $O(n)$ time. [2]

Input format (for Figure 1):

h 2

27 15 3 17 37 7 4 /* **NOTE:** node values are listed in breadth-first order from left to right, starting from the root. */

Output format:

(xi,xj) = (x1,x2)
Ancestry x1 = {17, 15, 27}
Ancestry x2 = {37, 15, 27}
Value of Max Ancestry (x1,x2) = 96

Q5. **[12 marks]**

You are given 'n' encrypted photos. The images in the photos belong to different biological species. You have to determine if there are more than $\frac{n}{2}$ photos that belong to the same species or not. Since the photos are encrypted, you can't examine them directly and have to **call a function** that takes as input **two** photos and tells you if they belong to the same species or not. Assume there can be at most 'm' different species, where $m<n$. The 'm' species are identified using integer values from 0 to m-1.

3. Code an **efficient** Divide & Conquer (D&C) algorithm in C/C++ that solves the above problem. Your code reads positive integer values of 'n' and 'm' from the input file. This is followed by the species of each of the 'n' photos (see input format below). You will also write a helper function decode()**,** that takes two photos as parameters and returns "Y" if they belong to the same species, otherwise returns "N" **(decode() is only allowed to compare two photos at a time and there is no way to inspect or compare them except through this function)**. If more than $\frac{n}{2}$ photos belong to the same species, then your D&C algorithm will report "success" and lists the indices of those photos, otherwise reports "failure" (see output format below). [10]
4. Note that the species values in the input can ***only*** be used by the decode() function. **You will get a zero if your algorithm reads it directly and counts the species.**
5. Give a clear description of your algorithm and data structures used to implement it in the comments at the beginning of your code. **What is the recurrence relation of your algorithm? What is the running time of your algorithm?** Your should include clear comments that explain your algorithm's time. [2]
6. The time complexity of your algorithm should not be more than $O(n\log_2 n)$.
7. You algorithm should run for large values of 'n'. The example is only for illustration purposes.

Example 1

Input format:

n 8
m 3
0 0 1 2 0 0 1 0

Output format:

Success
Dominant Species Count 5
Dominant Species Indices 0 1 4 5 7

Example 2

Input format:

n 16
m 6
3 0 1 2 3 4 3 5 3 3 2 3 3 0 3 3

Output format:

Success
Dominant Species Count 9
Dominant Species Indices 0 4 6 8 9 11 12 14 15

Example 3

Input format:

n 16
m 6
0 0 1 2 3 4 1 5 5 4 2 3 2 0 3 1

Output format:

Failure

Instructions and policies

1. When submitting, please ***rename the folder*** according to your ***roll number***.
2. Do ***delete all executables*** and ***test files*** before submitting your assignment on LMS.
3. Folder convention ***should not be changed.*** If you make any changes, the auto grader will ***grade your assignment 0***.
4. You must submit your ***own*** work. You may discuss the problems with other classmates but must not reveal the solution to others or copy someone's work. Remember to acknowledge other classmates if discussions with them has helped you.
5. You should name your code files using the following convention: qx.cpp
6. If the assignment includes any theoretical questions, then type your answer to those questions and submit a separate pdf file for each using the naming convention above.

7. Upload all your files in the corresponding assignment folder on LMS. **There will be a 20% deduction for assignments submitted up to one day late (the late deduction is only applicable to the questions submitted late, not on the whole assignment). Assignments submitted 24 hours after the deadline will not be marked.**
8. There will be vivas during grading of the assignment. The TAs will announce a schedule and ask you to sign up for viva time slots. Failure to show up for vivas will result in a **70% marks reduction** in the assignment.
9. In the questions where you are asked to create test cases. Think carefully about good test cases that check different conditions and corner cases. The examples given in the assignment are for clarity and illustration purposes. You should not assume that those are the only test cases your code should work for. Your code should be able to scale up to larger input sizes and more complex scenarios.
10. Do not make arbitrary assumptions about the input or the structure of the problem without clarifying them first with the Instructor or the TAs.
11. **We will use automated code testing so pay close attention to the input and output formats.**
12. **Make sure that your code compiles and runs on Ubuntu. You may choose to develop your code in your favourite OS environment, however, the code must be properly tested on Linux before submission. During vivas, your code should not have any compatibility issues. It's a good idea to use gcc -Wall option flag to generate the compiler's warnings. Pay attention to those warnings as fixing them will lead to a much better and robust code.**
13. For full credit, comment your code clearly and state any assumptions that you have made. There are marks for writing well-structured and efficient code.
14. Familiarize yourself with LUMS policy on plagiarism and the penalties associated with it. **We will use a tool to check for plagiarism in the submissions.**