School of Computer Engineering
KIIT deemed to be University
Laboratory Lesson Plan – Autumn 2023 (5th Semester)

Discipline: B.Tech (CSE), Section: CSE-20

Course Name and Code: Algorithms Laboratory-CS2098 (L-T-P-Cr: 0-0-2-1)

Instructor Name: Dr. Partha Sarathi Paul *Email:* parthasarathi.paulfcs@kiit.ac.in

Instructor Chamber: Faculty Block 101, Cabin-J, Block-C, Campus-14

Technical Assistants Names: Mr. Ranjit Kumar Sahoo (ranjit.sahoo@kiit.ac.in)

Mr. Siddhartha Sankar Mishra (siddhartha.mishra@kiit.ac.in)

Lab Class Hours:

Day	Time	Class Room
Saturday	04 PM - 06 PM	A-DL-002

Course Contents

- *Review of fundamentals of Data Structures*
- *Fundamentals of Algorithmic Problem Solving*: Analysis of time complexity of small algorithms through step/frequency count method (Sorting algorithms, GCD, Prime numbers, etc.)
- *Divide and Conquer Method:* Binary Search, Merge Sort, Quick Sort, Randomized Quick Sort
- *Heap & Priority Queues:* Building a heap, Heap sort algorithm, Min-Priority queue, Max-Priority queue
- *Greedy Techniques:* Fractional knapsack problem, Activity selection problem, Huffman's code
- *Dynamic Programming:* Matrix Chain Multiplication, Longest Common Subsequence
- *Graph Algorithms:* Dis-joint Set Data Structure, Representation Of Graph, Graph Traversals (BFS DFS), Single Source Shortest Path (Dijkstra's Algorithm, Bellman-Ford Algorithm), All Pair Shortest Path (Floyd-Warshall Algorithm), Minimum Cost Spanning Tree (Kruskal's Algorithm, Prim's Algorithm)

List of Experiments (Lab-Day wise)

Lab Day 1: Revision of Data Structures

1.1 *Aim of the experiment:* Write a program to find out the second smallest and second largest element stored in an array of n integers. n is the user input. The array takes input through random number generation within a given range.

Input: Array of Size n. Generate the array element through a random generator.

Output: second smallest, second largest

1.2 *Aim of the experiment:* Write a program to display an array of n integers (n>1), where every index of the array should contain the product of all elements in the input array except the element at the given index. Solve this problem by taking a single loop and without an additional array.

Input Array: 3 4 5 1 2

*Output Array***:** 40 30 24 120 60

Lab Day 2: Fundamentals of Algorithmic Problem Solving

2.1 *Aim of the experiment:* Write a function *generatePrimes(n)* that returns an array of prime numbers less than or equal to *n*. If your program contains one or more loops, count and print the number of times each loop is executed.

Size of input, $\mathbf{n_{input}}$	Observed number of times outer loop is executed, n_{outer}	Estimated parameter, **c1** = $\mathbf{n_{outer}}$ / $\mathbf{n_{input}}$	Observed number of times inner loop is executed (if any), n_{inner}	Estimated parameter, **c2** = $\mathbf{n_{inner}}$ / $\mathbf{n_{input}}$
10				
100				
1000				
10000				
100000				
1000000				

Check if the values of estimated parameters *c1* and *c2* are stable or slowly increasing according to the problem size. Explain your observation.

Input: Enter n - 10

Output: Prime numbers - 2 3 5 7

2.2 *Aim of the experiment:* Write a function *sine(x, d)* that returns the sine of input *x* accurately up to *d* decimal places computed using Taylor's series approximation of sine.

$\sin(x) = x - x^3/3! + x^5/5! - x^7/7! + \dots$

Count and print the number of times the main loop in your program executes.

Size of input, **d** decimal places	Observed number of times main loop executes, $\mathbf{n_{loop}}$	Estimated parameter, $\mathbf{c1 = \log(n_{loop})/d}$
2		
3		
4		
5		
6		

Check if the value of estimated parameter *c1* is stable or changing according to the problem size. Explain your observation.
Input: Enter x - 25.56
        Enter decimal places - 3
Output: 0.431

2.3 *Aim of the experiment:* Write a function *gcd(k)* which returns the GCD of k^{th} and $(k+1)^{th}$ Fibonacci numbers for $k>1$. Use Euclid's algorithm to calculate the GCD. Count the number of times the GCD loop executes for different values of *k* and record your observations in the following table.

input, **k**	Observed number of times GCD loop executes, $\mathbf{n_{loop}}$	Estimated parameter, $\mathbf{c1 = n_{loop}/k}$
10		
20		
30		
40		

Check if the value of estimated parameter *c1* is stable or changing according to the input value. Justify your observation.

Input: Enter k - 10
Output: 10th and 11th Fibonacci numbers - 34,55
GCD = 1

Lab Day 3: Fundamentals of Algorithmic Problem Solving (contd…)

3.1 *Aim of the experiment:* Write a function *recursiveMin(a, n)* that recursively computes the minimum value in the input array *a* and returns it. Initially, $n = |a|$.

```
recursiveMin(a, n) : a[0] if n==1                                  [base case]
                     min(a[n-1], recursiveMin(a, n-1))             [recursive
definition]
```

Count the depth of recursion for input *a* of different sizes and record your observations below.

Size of problem, $|\mathbf{a}|$	Observed depth of recursion, $\mathbf{d_{recur}}$	Estimated parameter, $\mathbf{c1 = d_{recur}/|a|}$
100		
10000		
1000000		

Check if the value of estimated parameter *c1* is stable or changing according to the input size. Justify your observation.

Input: Enter size of array - 100
Output: Minimum value = 12

3.2 *Aim of the experiment:* Guess game

A and B are playing a guessing game where B first thinks up an integer X (positive, negative or zero, and could be of arbitrarily large magnitude) and A tries to guess it. In response to A's guess, B gives exactly one of the following three replies:
a) Try a bigger number
b) Try a smaller number or
c) You got it.
Write a program by designing an efficient algorithm to minimize the number of guesses A has to make. An example (not necessarily an efficient one) below:
Assume B thinks up the number 35.
A's guess B's response.
10 Try a bigger number
20 Try a bigger number
30 Try a bigger number
40 Try a smaller number
35 You got it

3.3 *Aim of the experiment:* To find distinct integers in an array.
Write a function *unique(a)* that returns an array of unique elements in input array a. Run the function on inputs of following sizes and count the number of basic operations.

Size of problem, |**a**|	Observed number of basic operations during execution, $\mathbf{n_{op}}$	Estimated parameter, **c1** = $\mathbf{n_{op}/|a|^2}$
100		
10000		
1000000		

Check if the value of estimated parameter *c1* is stable or changing according to the input size. Justify your observation.

Lab Day 4: Divide and Conquer Method

4.1 *Aim of the experiment:* Write a program to implement Binary search to give the position of first appearance of the element being searched.

(i) Input:
Enter size of array: 10
Enter elements of the array:

7 9 2 11 19 17 12 5 7 12
Enter element to be searched: 12
Output:
12 found at position 7.

(ii) Input: Read from a text file containing numbers.
Enter element to be searched: 1678
Output:
1678 found at position 7.

4.2 *Aim of the experiment:* Write a program to implement Linear search to give the position of first appearance of the element being searched.

(i) Input:
Enter size of array: 10
Enter elements of the array:
7 9 2 11 19 17 12 5 7 12
Enter element to be searched: 15
Output:
15 not found.

(ii) Input: Read from a text file containing numbers.
Enter element to be searched: 1500
Output:
1500 not found.

4.3 *Aim of the experiment:* Write a program to implement quicksort on an array and count the number of comparisons.

Input:
Enter size of array: 10
Enter elements of the array:
7 9 2 11 19 17 12 5 7 12
Output:
Sorted array is:
2 5 7 7 9 11 12 12 17 19
26 comparisons

Lab Day 5: Divide and Conquer Method (contd…)

5.1 *Aim of the experiment:* Write a program to implement Insertion sort on an array and count the number of comparisons.

Input:
Enter size of array: 10
Enter elements of the array:
7 9 2 11 19 17 12 5 7 12
Output:
Sorted array is:
2 5 7 7 9 11 12 12 17 19
45 comparisons

5.2 *Aim of the experiment:* Write a program to implement Merge sort on an array and count the number of comparisons.

Input:
Enter size of array: 10
Enter elements of the array:
7 9 2 11 19 17 12 5 7 12
Output:
Sorted array is:
2 5 7 7 9 11 12 12 17 19
37 comparisons

Lab Day 6: Heap & Priority Queues

6.1 *Aim of the experiment:* You have been provided with the set of files having file names *datX.csv* where *X* stands for the input size. Files are available here (https://drive.google.com/drive/folders/1KD863j_AZmxFXcCc3DYFst6YvN8lPko2?usp=sharing)
Suppose *struct person* is defined as follows:

```
struct person
{
    int id;
    char *name;
    int age;
    int height;
    int weight;
};
```

These files contain comma-separated entries for the details of the students.

Write a program to read the data from these files and store them in a dynamically allocated array of *struct person*. *Sort* arrays of *struct person* based on the *height* field using Heap Sort. Plot the time taken and observe how they vary with the *size of the input*.

Input: Data file *datX.csv.* Structure format

Sample Input

0	Donald Scantling	39	77	231
1	Levi Maier	56	77	129
2	Barbara Donnelly	63	78	240
3	Dorothy Helton	47	72	229
4	Florence Smith	24	75	171
5	Erica Anyan	38	73	102
6	Norma Webster	23	75	163

Output: Plot of time taken by the Heap Sort algorithm with different sizes of input. X-Axis should contain different file names such as *dat10*, *dat1000*, *dat10000*,... and the Y-axis should contain the time taken by the Heap Sort algorithm on each respective file.

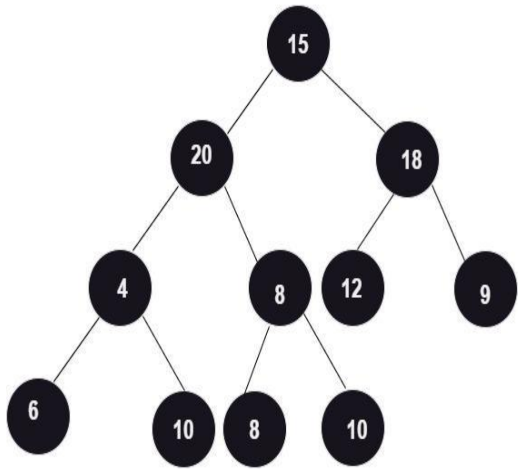
6.2 *Aim of the experiment:* Create a heap, which will follow the *max-heap* and *min-heap* properties at each alternative level. Assume the root node will follow the min-heap property. For example: At level 0 (root node) will follow the min-heap property, and level 1 will follow the max-heap property and so on.

Input: Enter size of array: 11

Enter elements of the array: 20, 15, 18, 10, 8, 12, 9, 6, 4, 8, 10

Output: 15, 20, 18, 4, 8, 12, 9, 6, 10, 8, 10

Heap Tree:



Lab Day 7: Greedy Techniques

7.1 *Aim of the experiment:* Write a program to find the maximum profit nearest to but not exceeding the given knapsack capacity using the Fractional Knapsack algorithm.

Notes# Declare a structure ITEM having members value and weight. Apply heap sort technique to sort the items in non-increasing order, according to their value/weight.

Input:

```
Enter the number of items: 3
Enter the Value of item : 27 , 14, 26
Enter the weight of item : 16, 12, 13
Enter the capacity of knapsack:18
```

Output:

```
Item No     Value          Weight        Amount to be taken
1           26.000000      13.000000     1.000000
2           27.000000      16.000000     0.312500
3           14.000000      12.000000     0.000000
```

Maximum profit: 34.437500

7.2 *Aim of the experiment:* Huffman coding assigns variable length codewords to fixed length input characters based on their frequencies/probabilities of occurrence. Given a set of characters along with their frequency of occurrences. Write a c program to perform the following functions.

Note# Declare a structure CHAR having members symbol and frequency. Create a Min-Priority Queue, keyed on frequency attributes.

Input:

```
        Enter the number of distinct characters: 6
        Enter the characters:       a     b     c     d     e     f
        Enter its  frequencies:     45    13    12    16    9     5
```

Output:

```
In-order traversal of the tree (Huffman): a  c  b  f  e  d
```

Lab Day 8: Dynamic Programming

8.1 *Aim of the experiment:* Write a program to implement the matrix chain multiplication problem using M-table & S-table to find optimal parenthesization of a matrix-chain product. Print the number of scalar multiplications required for the given input.

Note# Dimensions of the matrices can be inputted as a sequence and maintained using dynamic memory allocation.

Input: Enter the dimension sequence:

Row No:	*2*	*4*	*5*
Column No:	*4*	*5*	*6*

Output:

```
        M Table:
        0   1   2    3    4
        1   0   40   100  136
        2   0   0    120  105
        3   0   0    0    90
        4   0   0    0    0
```

S Table:

```
0   1   2   3   4
1   0   1   2   3
2   0   0   2   2
3   0   0   0   3
4   0   0   0   0
```

Output: Optimal parenthesization: (((A1   A2)   A3)   A4)

The optimal ordering of the given matrices requires 136 scalar multiplications.

8.2 *Aim of the experiment:* Given two strings

- Find out all possible Longest Common Subsequences.
- Calculate length of the LCS.

Input:

```
Enter the first string: 10010101
Enter the second string: 010110110
```

Output:

```
LCS:
   100110
   101011
   101101
   010101
   010101
   001010
   100110
```

LCS Length: 6

Lab Day 9: Graph Algorithms - Graph traversal

9.1 *Aim of the experiment:* Consider an undirected graph where each edge weights 2 units. Each of the nodes is labeled consecutively from 1 to n. The user will input a list of edges for describing an undirected graph. After representation of the graph, from a given starting position

- Display the *breadth-first search traversal.*
- Determine and display the shortest distance to each of the other nodes using the *breadth-first search* algorithm. Return an array of distances from the start node in node number order. If a node is unreachable, return -1 for that node.

Input Format:

- The first line contains two space-separated integers 'n' and 'm', the number of nodes and edges in the graph.

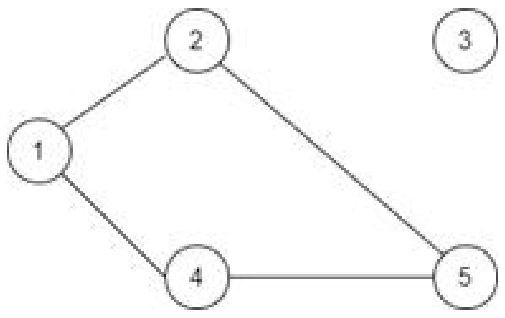
- Each line 'i' of the 'm' subsequent lines contains two space-separated integers 'u' and 'v', that describe an edge between nodes 'u' and 'v'.

- The last line contains a single integer 's', the node number to start from.

Output Format:

- The first line shows the result of the BFS traversal.

- The last line shows an array of distances from node 's' to all other nodes.



Input:

```
5 4
1 2
1 4
4 5
2 5
1
```

Output:

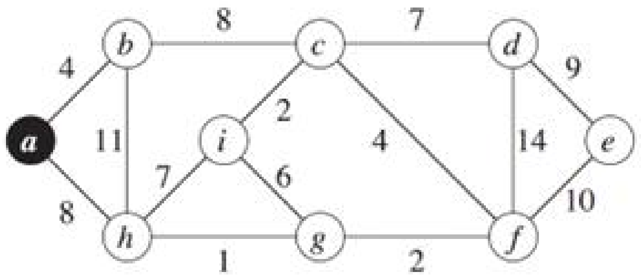
```
BFS Traversal: 1 2 4 5
Distance [2 -1 2 4]
```

Lab Day 10: Graph Algorithms - Minimum cost spanning tree

10.1 *Aim of the experiment:* Given an undirected weighted connected graph G(V, E) and starring vertex 's'. Maintain a Min-Priority Queue 'Q' from the vertex set V and apply Prim's algorithm to

- Find the minimum spanning tree T(V, E'). Display the cost adjacency matrix of 'T'.
- Display total cost of the minimum spanning tree T.

Note# Nodes will be numbered consecutively from 1 to n (user input), and edges will have varying weight. The graph G can be inputted as a cost adjacency matrix. The expected output could be the cost adjacency matrix of the minimum spanning tree and total cost of the tree as per the sample output.



Input:

Number of Vertices: 9

	1	2	3	4	5	6	7	8	9
1	0	4	0	0	0	0	0	8	0
								1	0
2	4	0	8	0	0	0	0	1	
3	0	8	0	7	0	4	0	0	2
4	0	0	7	0	9	14	0	0	0
5	0	0	0	9	0	10	0	0	0
6	0	0	4	14	10	0	2	0	0
7	0	0	0	0	0	2	0	1	6
8	8	11	0	0	0	0	1	0	7
9	0	0	2	0	0	0	6	7	0

Starting Vertex: 1

Output:

	1	2	3	4	5	6	7	8	9

```
1 |  0    4    0    0    0    0    0  8  0
2 |  4    0    0    0    0    0    0  0  0
3 |  0    0    0    7    0    4    0  0  2
4 |  0    0    7    0    9    0    0  0  0
5 |  0    0    0    9    0    0    0  0  0
6 |  0    0    4    0    0    0    2  0  0
7 |  0    0    0    0    0    2    0  1  0
8 |  8    0    0    0    0    0    1  0  0
9 |  0    0    2    0    0    0    0  0  0
```

Total Weight of the Spanning Tree: 37

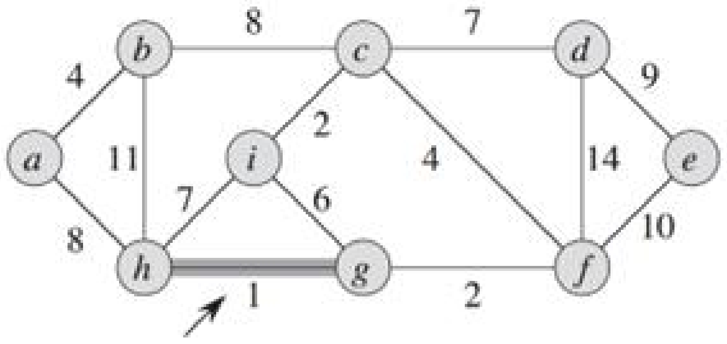
10.2 *Aim of the experiment:* Given an undirected weighted connected graph G(V, E). Apply Krushkal's algorithm to

- Find the minimum spanning tree T(V, E') and Display the selected edges of G.
- Display total cost of the minimum spanning tree T.

Note# Nodes will be numbered consecutively from 1 to n (user input), and edges will have varying weight. The weight matrix of the graph can be represented from the user's input in the given format. The expected output could be the selected edge and the corresponding cost of the edge as per the sample output.

Input Format:

- The first line contains two space-separated integers 'n' and 'm', the number of nodes and edges in the graph.
- Each line 'i' of the 'm' subsequent lines contains three space-separated integers 'u', 'v' and 'w', that describe an edge (u, v) and weight 'w'.



Input:

```
9 14
```

```
1 2 4
1 8 8
2 3 8
2 8 11
3 4 7
3 6 4
3 9 2
4 5 9
4 6 14
5 6 10
6 7 2
7 8 1
7 9 6
8 9 7
```

Output:

```
Edge        Cost
8--7        1
7--6        2
3--9        2
1--2        4
3--6        4
3--4        7
1--8        8
4--5        9
```

Total Weight of the Spanning Tree: 37

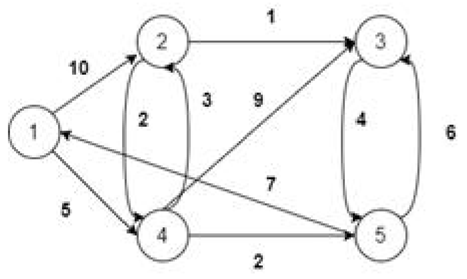
Lab Day 11: Graph Algorithms - Shortest path

11.1 *Aim of the experiment:* Given a directed graph G (V, E) and a starting vertex ‘s’.

- Determine the lengths of the shortest paths from the starting vertex ‘s’ to all other vertices in the graph G using Dijkstra’s Algorithm.

- Display the shortest path from the given source 's' to all other vertices.

Note# Nodes will be numbered consecutively from 1 to n (user input), and edges will have varying distances or lengths. The graph G can be inputted as a non-negative cost adjacency matrix. The expected output could be as per the sample format.



Input:

Number of Vertices: 5

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 10 & 0 & 5 & 0 \\ 2 & 0 & 0 & 1 & 2 & 0 \\ 3 & 0 & 0 & 0 & 0 & 4 \\ 4 & 3 & 0 & 9 & 0 & 2 \\ 5 & 7 & 0 & 6 & 0 & 0 \end{array}$$

Source Vertex: 1

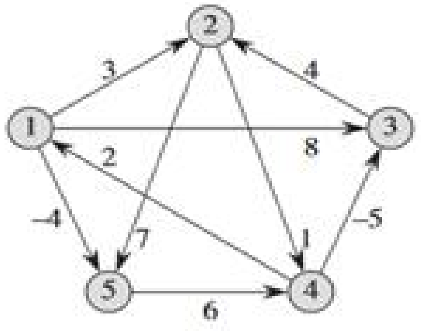
Output:

Source	Destination	Cost	Path
1	1	0	-
1	2	8	1->4->2
1	3	9	1->4->2->3
1	4	5	1->4
1	5	7	1->4->5

11.2 *Aim of the experiment:* Given a directed weighted graph G (V, E) where weight indicates distance. Vertices will be numbered consecutively from 1 to n (user input), and edges will have varying distances or lengths.

- Determine the length of the shortest path between every pair of vertices using Floyd-Warshall's algorithm.
- Display the intermediate vertices on the shortest-path from the given pair of vertices (u,v).

Note# The graph G can be inputted as a cost adjacency matrix. The expected output could be a shorted-path weight matrix and the path consisting of intermediate vertices.



Input:

Number of Vertices: 5

	1	2	3	4	5
1	0	3	8	0	-4
2	0	0	0	1	7
3	0	4	0	0	0
4	2	0	-5	0	0
5	0	0	0	6	0

Enter the source and destination vertex: 2 5

Output:

```
        1       2       3       4       5
1       0       1      -3       2      -4
2       3       0      -4       1      -1
3       7       4       0       5       3
4       2      -1      -5       0      -2
5       8       5       1       6       0
```

Shortest Path from vertex 2 to vertex 5: 2-->4-->1-->5

Path weight: -1

Lab Day 12 : Lab Sessional

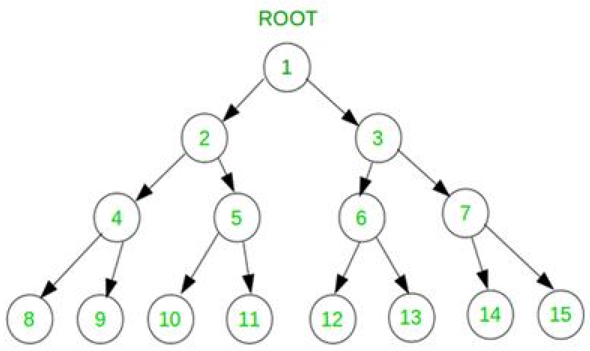
Grading Policies

Evaluation Steps	**Evaluation Component**	**Evaluator**	**Marks**
Lab Internal (Before Mid-term)	Program Execution	Lab TA	10
	Lab Record	Lab TA	05
	Viva/Quiz	Lab Instructor	15
Lab Internal (After Mid-term)	Program Execution	Lab TA	10
	Lab Record	Lab TA	05
	Viva/Quiz	Lab Instructor	15
Lab Sessional/Project			40

Practice Problem Sets

1. Given a binary tree, print all the nodes between two given levels in a binary tree.

Input: nodes between levels 2 and 3



Output: 2 3 4 5 6 7

2. Given an array arr[] of size N having distinct numbers sorted in increasing order and the array has been right rotated (i.e. the element will be cyclically shifted to the starting position of the array) k number of times, the task is to find the value of k.

```
Input:     arr[]    =    {15,    18,    2,    3,    6,    12}
Output:                                                     2
```

Explanation: Initial array must be {2, 3, 6, 12, 15, 18}. We get the given array after rotating the initial array twice. K

3. Write a program to remove brackets from an algebraic string containing '+', '-' operators.

Input: 3 + (7 – (6 - 7) + 10)

Output: 3 + 7 – 6 +7 + 10

4. A dominant element in an integer array is a number whose value is greater than the sum of all the elements present before that element. Write a program to print all the dominant elements in an integer array.

Input: A = {10, -5, 6, 9, -2, 17}

Output: 6 17

5. Given two integers x and n. Write a program to design your own power function to compute x^n following the divide and conquer approach.

6. Write a program to build a max heap from a given array. Print all the elements larger than a given element of the max heap. Delete the second largest element from the max heap.

7. Write a program to encode your full name using Huffman coding.

8. Given N items where each item has some weight and profit associated with it. We are also given a bag with capacity W, [i.e., the bag can hold at most **W** weight in it]. The target is to put the items into the bag such that the sum of profits associated with them is the maximum possible. Only the whole item can be placed in the bag.

E.g.

Input: N = 3, W = 5, profit[] = {1, 2, 3}, weight[] = {4, 5, 1}

Output: 4

9. Given an undirected graph where every node has some positive real identifier. Find the sum of identifiers of the node.

10. Given an undirected graph. Write a program to print the nodes of a cycle if present.

11. Suppose a sorted array of n distinct integers is given. Write a program to find any element A[i] such that A[i] equals to i where i is the index.

12. Given an undirected weighted connected graph, find the Really Special SubTree in it. The Really Special SubTree is defined as a subgraph consisting of all the nodes in the graph and:

- There is only one exclusive path from a node to every other node.
- The subgraph is of minimum overall weight (sum of all edges) among all such subgraphs.
- No cycles are formed

To create the Really Special SubTree, always pick the edge with the smallest weight. Determine if including it will create a cycle. If so, ignore the edge. If there are edges of equal weight available:

- Choose the edge that minimizes the sum u+v+wt where u and v are vertices and wt are weight edges.
- If there is still a collision, choose any of them.

Print the overall weight of the tree formed using the rules. For example, given the following edges:

u	v	wt
1	2	2
2	3	3
3	1	5

First choose 1->2 at weight 2. Next choose 2->3 at weight 3. All nodes are connected without cycles for a total weight of 3+2=5.

Function:

- g_nodes: an integer that represents the number of nodes in the tree.
- g_from: an array of integers that represents beginning edge node numbers.
- g_to: an array of integers that represent ending edge node number.
- g_weight: an array of integers that represent the weight of each edge.

Input format

The first line has two space-separated integers **g_nodes** and **g_edges** , the number of nodes and edges in the graph. The next **g_edges** lines each consist of three space-separated integers

g_from, g_to and **g_weight**, where **g_from** and **g_to** denote the two nodes between which the undirected edge exists and **g_weight** denotes the weight of that edge.

Constraints

- **2<= g_nodes<=3000**
- **1<=g_edges<= N*(N-1)/2**
- **1<=g_from, g_to<=N**
- **0<=g_weight<=10^5**

NOTE If there are edges between the same pair of nodes with different weights, they are to be considered as is, like multiple edges.

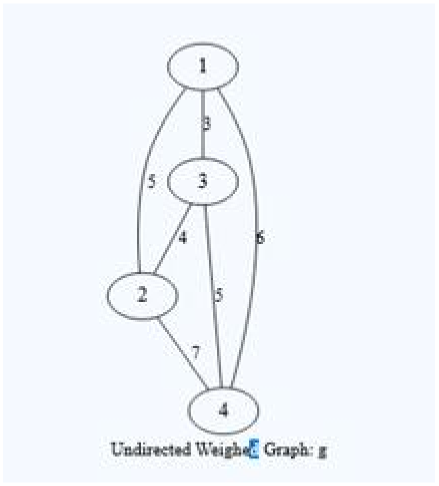
Output format

Print a single integer denoting the total weight of the Really Special SubTree.

Sample input

4 6

1 2 5
1 3 3
4 1 6
2 4 7
3 2 4
3 4 5



Sample output: 12.

Explanation

The graph given in the test case is shown above.

Applying Kruskal's algorithm, all of the edges are sorted in ascending order of weight.

After sorting, the edge choices are available as:

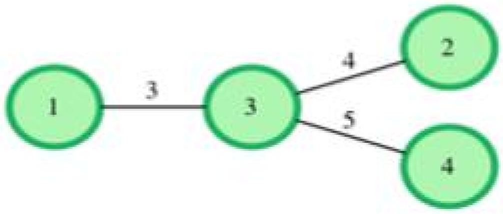
1->3(w=3), 2->3(w=4), 1->2(w=4), 3->4(w=5), 1->4(w=6) and 2->4(w=7).

Select 1->3 (w=3) because it has the lowest weight without creating a cycle.

Select 2->3(w=4) because it has the lowest weight without creating a cycle.

The edge 1->2(w=4) would form a cycle, so it is ignored.

Select 3->4(w=5) to finish the MST yielding a total weight of 3+4+5 =12.



13. A traveler has arrived in a nation with n cities, and he wants to see each one while spending as little as feasible. There are m two-way roads in the country. A road connects two cities and it has a cost. The ith road connects cities u_i, $v_{i,}$ and cost $c_{i.}$ In contrast to the standard TSP issue, in this case there is a penalty associated with changing paths. The viscosity for each road is defined as the ith road has viscosity $g_{i.}$ Switching from a road with viscosity x to a road with viscosity y adds cost. Find the minimum cost needed to see all of the cities.

NOTE:

- *The traveler is comfortable with seeing a city or a road several times.*
- *This problem is an approximation problem and you will get a higher score if you print a path with a lower cost.*

Input Format:

- The first line contains two integers n, m.
- The next m lines contain the description of roads where the ith line contains v_i, $u_{i,}$ c_i, and $g_{i,.}$

Output Format:

Print the order in which the traveler will see the cities. In the first line, print k that denotes the number of roads in the path ($n-1<=k<=10^6$). In the next line, print the numbers of the roads he will pass in order.

Constraint: $3<=n, m<=10^5$. $1<= c_i, g_i<= 10^9$ country is connected.

Sample Input	Sample output
4 4	3
1 2 10 20	1 2 4
2 3 30 40	
3 4 50 60	
3 4 60 10	

Explanation

The cost of the path is $10+\sqrt{20^2+40^2}+30+\sqrt{40^2+10^2}+60$**. While cost of the path [1,2,3] is** $10+\sqrt{20^2+40^2}+30+\sqrt{40^2+60^2}+50$**.**

14. Given a binary search tree (BST), find the lowest common ancestor (LCA) node of two given nodes in the BST. The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow a node to be a descendant of itself).

Input:

bst: a binary tree representing the existing BST.
p: the value of node p as described in the question
q: the value of node q as described in the question

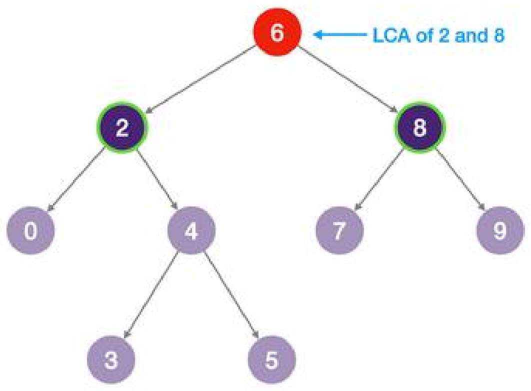
Output:

The value of the LCA between nodes p and q

Example 1:

Input:
bst = [6,2,8,0,4,7,9,x,x,3,5]
p = 2
q = 8
Output: 6

Explanation:



The ancestors of node p with value 2 are node 2 and node 6. The ancestors of node q with value 8 are node 8 and node 6. The lowest common ancestors between these two nodes is 6.

15. Given a n x n matrix where each of the rows and columns are sorted in ascending order, find the kth smallest element in the matrix. (Using heap sort).
Note that it is the kth smallest element in the sorted order, not the kth distinct element.

Example:
Input:

```
matrix = [
  [ 1,  5,  9],
  [10, 11, 13],
  [12, 13, 15]
],
k = 8,
```

Output: 13

Explanation: The elements in the matrix are [1,5,9,10,11,12,13,**13**,15], and the 8th smallest number is 13.

16. For this problem, given a list of tasks and a list of requirements, compute a sequence of tasks that can be performed, such that we complete every task once while satisfying all the requirements.(Topological sorting).
Each requirement will be in the form of a list [a, b], where task a needs to be completed first before task b can be completed, There is guaranteed to be a solution.

Example 1
Input:

tasks = ["a", "b", "c", "d"]

requirements = [["a", "b"], ["c", "b"], ["b", "d"]]

Output: ["a", "c", "b", "d"]

17. Write a program to calculate the number of shifts an insertion sort performs when sorting an array?

If a[i] is the number of elements over which the ith element of the array has to shift, then the total number of shifts will be a[1]+a[2]+...a[n].

Example:
arr[4,3,2,1]

Array	**Shifts**
[4,3,2,1]	
[3,4,2,1]	**1**
[2,3,4,1]	**2**
[1,2,3,4]	**3**

Total shifts = 1+2+3=6

Input Format
The first line contains an integer n , the length of an arr. Here arr is an array. The next line contains n integers separated by space of arr[i].

Constraints

- **1 <= n<= 100000**
- **1<= a[i]<= 10000000**

Sample Input:
5
2 1 3 1 2
Sample output: 4

Explanation:
```
Array : 2 1 3 1 2 -> 1 2 3 1 2->1 1 2 3 2 -> 1 1 2 2 3
Moves:   -         1      -         2     -       1          =4
```