Lab Manual

Course Title: Design and Analysis of Algorithms

Week 1:

Note: Input, output format for problem I, II and III is same and is given at the end of this exercise.

I. Given an array of nonnegative integers, design a linear algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find total number of comparisons for each input case. (Time Complexity = O(n), where n is the size of input)

Sample I/O Problem - 1:

Input:	**Output:**
3 8 34 35 65 31 25 89 64 30 89 5 977 354 244 546 355 244 6 23 64 13 67 43 56 63	Present 6 Present 3 Not Present 6

II. Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find total number of comparisons for each input case. (Time Complexity = O(nlogn), where n is the size of input).

III. Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether a given key element is present in the sorted array or not. For an array arr[n], search at the indexes arr[0], arr[2], arr[4],. ,arr[2^k] and so on. Once the interval (arr[2^k] < key < arr[2^{k+1}]) is found, perform a linear search operation from the index 2^k to find the element key. (Complexity < O(n), where n is the number of elements need to be scanned for searching): **Jump Search**

Input format:
The first line contains number of test cases, T.
For each test case, there will be three input lines.
First line contains n (the size of array).
Second line contains n space-separated integers describing array.
Third line contains the key element that need to be searched in the array.

Output format:
The output will have T number of lines.
For each test case, output will be "**Present**" if the key element is found in the array, otherwise "**Not Present**".
Also for each test case output the number of comparisons required to search the key.

Sample I/O Problem - 2, 3:

Input:	Output:
3 5 12 23 36 39 41 41 8 21 39 40 45 51 54 68 72 69 10 101 246 438 561 796 896 899 4644 7999 8545 7999	Present 3 Not Present 4 Present 3

Week 2:

I. Given a sorted array of positive integers containing few duplicate elements, design an algorithm and implement it using a program to find whether the given key element is present in the array or not. If present, then also find the number of copies of given key. (Time Complexity = O(log n))

Input format:
The first line contains number of test cases, T.
For each test case, there will be three input lines.
First line contains n (the size of array).
Second line contains space-separated integers describing array.
Third line contains the key element that need to be searched in the array.

Output format:
The output will have T number of lines.
For each test case T, output will be the key element and its number of copies in the array if the key element is present in the array otherwise print " **Key not present**".

Sample I/O Problem I:

Input:	Output:
2 10 235 235 278 278 763 764 790 853 981 981 981 15 1 2 2 3 3 5 5 5 25 75 75 75 97 97 97 75	981 - 2 75 - 3

II. Given a sorted array of positive integers, design an algorithm and implement it using a program to find three indices i, j, k such that arr[i] + arr[j] = arr[k].

Input format:
The first line contains number of test cases, T.
For each test case, there will be two input lines.
First line contains n (the size of array).
Second line contains space-separated integers describing array.

Output:
The output will have T number of lines.
For each test case T, print the value of i, j and k, if found else print "**No sequence found**".

Sample I/O Problem II:

Input:	**Output:**
3 5 1 5 84 209 341 10 24 28 48 71 86 89 92 120 194 201 15 64 69 82 95 99 107 113 141 171 350 369 400 511 590 666	No sequence found. 2, 7, 8 1, 6, 9

III. Given an array of nonnegative integers, design an algorithm and a program to count the number of pairs of integers such that their difference is equal to a given key, K.

Input format:
The first line contains number of test cases, T.
For each test case, there will be three input lines.
First line contains n (the size of array).
Second line contains space-separated integers describing array.
Third line contains the key element.

Output format:
The output will have T number of lines.
For each test case T, output will be the total count i.e. number of times such pair exists.

Sample I/O Problem III:

Input:	**Output:**
2 5 1 51 84 21 31 20 10 24 71 16 92 12 28 48 14 20 22 4	2 4

Week 3:

I. Given an unsorted array of integers, design an algorithm and a program to sort the array using insertion sort. Your program should be able to find number of comparisons and shifts (shifts - total number of times the array elements are shifted from their place) required for sorting the array.

Input Format:
The first line contains number of test cases, T.
For each test case, there will be two input lines.
First line contains n (the size of array).
Second line contains space-separated integers describing array.

Output Format:
The output will have T number of lines.
For each test case T, there will be three output lines.
First line will give the sorted array.
Second line will give total number of comparisons.
Third line will give total number of shift operations required.

Sample I/O Problem I:

Input:	**Output:**
3 8 -23 65 -31 76 46 89 45 32 10 54 65 34 76 78 97 46 32 51 21 15 63 42 223 645 652 31 324 22 553 -12 54 65 86 46 325	-31 -23 32 45 46 65 76 89 comparisons = 13 shifts = 20 21 32 34 46 51 54 65 76 78 97 comparisons = 28 shifts = 37 -12 22 31 42 46 54 63 65 86 223 324 325 553 645 652 comparisons = 54 shifts = 68

II. Given an unsorted array of integers, design an algorithm and implement a program to sort this array using selection sort. Your program should also find number of comparisons and number of swaps required.

Input Format:
The first line contains number of test cases, T.
For each test case, there will be two input lines.
First line contains n (the size of array).
Second line contains space-separated integers describing array.

Output Format:
The output will have T number of lines.
For each test case T, there will be three output lines.
First line will give the sorted array.
Second line will give total number of comparisons.
Third line will give total number of swaps required.

Sample I/O Problem II:

Input:	**Output:**
3 8 -13 65 -21 76 46 89 45 12 10 54 65 34 76 78 97 46 32 51 21 15 63 42 223 645 652 31 324 22 553 12 54 65 86 46 325	-21 -13 12 45 46 65 76 89 comparisons = 28 swaps = 7 21 32 34 46 51 54 65 76 78 97 comparisons = 45 swaps = 9 12 22 31 42 46 54 63 65 86 223 324 325 553 645 652 comparisons = 105 swaps = 14

III. Given an unsorted array of positive integers, design an algorithm and implement it using a program to find whether there are any duplicate elements in the array or not. (use sorting) (Time Complexity = O(n log n))

Input Format:
The first line contains number of test cases, T.
For each test case, there will be two input lines.
First line contains n (the size of array).
Second line contains space-separated integers describing array.

Output Format:

The output will have T number of lines.
For each test case, output will be **'YES'** if duplicates are present otherwise '**NO**'.

Sample I/O Problem III:

Input:	**Output:**
3 5 28 52 83 14 75 10 75 65 1 65 2 6 86 2 75 8 15 75 35 86 57 98 23 73 1 64 8 11 90 61 19 20	NO YES NO

Week 4:

I. Given an unsorted array of integers, design an algorithm and implement it using a program to sort an array of elements by dividing the array into two subarrays and combining these subarrays after sorting each one of them. Your program should also find number of comparisons and inversions during sorting the array.

Input Format:
The first line contains number of test cases, T.
For each test case, there will be two input lines.
First line contains n (the size of array).
Second line contains space-separated integers describing array.

Output Format:
The output will have T number of lines.
For each test case T, there will be three output lines.
First line will give the sorted array.
Second line will give total number of comparisons.
Third line will give total number of inversions required.

Sample I/O Problem I:

Input:	**Output:**
3 8 23 65 21 76 46 89 45 32 10 54 65 34 76 78 97 46 32 51 21 15 63 42 223 645 652 31 324 22 553 12 54 65 86 46 325	21 23 32 45 46 65 76 89 comparisons = 16 inversions = 21 32 34 46 51 54 65 76 78 97 comparisons = 22 inversions = 12 22 31 42 46 54 63 65 86 223 324 325 553 645 652 comparisons = 43 inversions =

II. Given an unsorted array of integers, design an algorithm and implement it using a program to sort an array of elements by partitioning the array into two subarrays based on a pivot element such that one of the sub array holds values smaller than the pivot element while another sub array holds values greater than the pivot element. Pivot element should be selected randomly from the array. Your program should also find number of comparisons and swaps required for sorting the array.

Input Format:

The first line contains number of test cases, T.
For each test case, there will be two input lines.
First line contains n (the size of array).
Second line contains space-separated integers describing array.

Output Format:
The output will have T number of lines.
For each test case T, there will be three output lines.
First line will give the sorted array.
Second line will give total number of comparisons.
Third line will give total number of swaps required.

Sample I/O Problem II:

Input:	**Output:**
3 8 23 65 21 76 46 89 45 32 10 54 65 34 76 78 97 46 32 51 21 15 63 42 223 645 652 31 324 22 553 12 54 65 86 46 325	21 23 32 45 46 65 76 89 comparisons = 14 swaps = 10 21 32 34 46 51 54 65 76 78 97 comparisons = 29 swaps = 21 12 22 31 42 46 54 63 65 86 223 324 325 553 645 652 comparisons = 45 swaps = 39

III. Given an unsorted array of integers, design an algorithm and implement it using a program to find Kth smallest or largest element in the array. (Worst case Time Complexity = O(n))

Input Format:
The first line contains number of test cases, T.
For each test case, there will be three input lines.
First line contains n (the size of array).
Second line contains space-separated integers describing array.
Third line contains K.

Output Format:
The output will have T number of lines.
For each test case, output will be the Kth smallest or largest array element.
If no Kth element is present, output should be "**not present**".

Sample for Kth smallest:

Input:	**Output:**
3 10 123 656 54 765 344 514 765 34 765 234 3 15 43 64 13 78 864 346 786 456 21 19 8 434 76 270 601 8	123 78

Week 5:

I. Given an unsorted array of alphabets containing duplicate elements. Design an algorithm and implement it using a program to find which alphabet has maximum number of occurrences and

print it. (Time Complexity = O(n)) (Hint: Use counting sort)

Input Format:
The first line contains number of test cases, T.
For each test case, there will be two input lines.
First line contains n (the size of array).
Second line contains space-separated integers describing array.

Output:
The output will have T number of lines.
For each test case, output will be the array element which has maximum occurrences and its total number of occurrences.
If no duplicates are present (i.e. all the elements occur only once), output should be "**No Duplicates Present**".

Sample I/O Problem I:

Input:	**Output:**
3 10 a e d w a d q a f p 15 r k p g v y u m q a d j c z e 20 g t l l t c w a w g l c w d s a a v c l	a – 3 No Duplicates Present l - 4

II. Given an unsorted array of integers, design an algorithm and implement it using a program to find whether two elements exist such that their sum is equal to the given key element. (Time Complexity = O(n log n))

Input Format:
The first line contains number of test cases, T.
For each test case, there will be two input lines.
First line contains n (the size of array).
Second line contains space-separated integers describing array.
Third line contains key

Output Format:
The output will have T number of lines.
For each test case, output will be the elements arr[i] and arr[j] such that arr[i]+arr[j] = key if exist otherwise print '**No Such Elements Exist**".

Sample I/O Problem II:

Input:	**Output:**
2 10 64 28 97 40 12 72 84 24 38 10 50 15 56 10 72 91 29 3 41 45 61 20 11 39 9 12 94 302	10 40 No Such Element Exist

III. You have been given two sorted integer arrays of size m and n. Design an algorithm and implement it using a program to find list of elements which are common to both. (Time Complexity = O(m+n))

Input Format:
First line contains m (the size of first array).
Second line contains m space-separated integers describing first array.
Third line contains n (the size of second array).
Fourth line contains n space-separated integers describing second array.

Output Format:
Output will be the list of elements which are common to both.

Sample I/O Problem III:

Input:	**Output:**
7 34 76 10 39 85 10 55 12 30 55 34 72 10 34 10 89 11 30 69 51	10 10 34 55

Note: Consider the following input format in the form of adjacency matrix for graph based questions (directed/undirected/weighted/unweighted graph).

Input Format: Consider example of below given graph in Figure (a).
A boolean matrix AdjM of size V X V is defined to represent edges of the graph. Each edge of graph is represented by two vertices (start vertex u, end vertex v). That means, an edge from u to v is represented by making AdjM[u,v] and AdjM[v,u] = 1. If there is no edge between u and v then it is represented by making AdjM[u,v] = 0. Adjacency matrix representation of below given graph is shown in Figure (b). Hence edges are taken in the form of adjacency matrix from input. In case of weighted graph, an edge from u to v having weight w is represented by making AdjM[u,v] and AdjM[v,u] = w.

Input format for this graph is shown in Figure (c).
First input line will obtain number of vertices V present in graph.
After first line, V input lines are obtained. For each line i in V, it contains V space separated boolean integers representing whether an edge is present between i and all V.

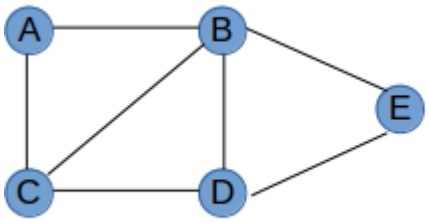

Figure (a)

	A	B	C	D	E
A	0	1	1	0	0
B	1	0	1	1	1
C	1	1	0	1	0
D	0	1	1	0	1
E	0	1	0	1	0

Figure (b)

```
5
0 1 1 0 0
1 0 1 1 1
1 1 0 1 0
0 1 1 0 1
0 1 0 1 0
```

Figure (c)

Week 6:

I. Given a (directed/undirected) graph, design an algorithm and implement it using a program to find if a path exists between two given vertices or not. (Hint: use DFS)

Input Format:
Input will be the graph in the form of adjacency matrix or adjacency list.
Source vertex number and destination vertex number is also provided as an input.

Output Format:
Output will be '**Yes Path Exists**' if path exists, otherwise print '**No Such Path Exists**'.

Sample I/O Problem I:

Input:	**Output:**
5 0 1 1 0 0 1 0 1 1 1 1 1 0 1 0 0 1 1 0 1 0 1 0 1 0 1 5	Yes Path Exists

II. Given a graph, design an algorithm and implement it using a program to find if a graph is bipartite or not. (Hint: use BFS)

Input Format:
Input will be the graph in the form of adjacency matrix or adjacency list.

Output Format:
Output will be '**Yes Bipartite**' if graph is bipartite, otherwise print '**Not Bipartite**'.

Sample I/O Problem II:

Input:	**Output:**
5 0 1 1 0 0 1 0 1 1 1 1 1 0 1 0 0 1 1 0 1 0 1 0 1 0	Not Bipartite

III. Given a directed graph, design an algorithm and implement it using a program to find whether cycle exists in the graph or not.

Input Format:
Input will be the graph in the form of adjacency matrix or adjacency list.

Output Format:
Output will be '**Yes Cycle Exists**' if cycle exists otherwise print '**No Cycle Exists**'.

Sample I/O Problem III:

Input:	**Output:**
5 0 1 1 0 0 0 0 0 1 1	No Cycle Exists

0 1 0 1 0 0 0 0 0 1 0 0 0 0 0	

Week 7:

Note: Input, output format along with sample input output for problem I and II is same and is provided at the end of problem II.

I. After end term examination, Akshay wants to party with his friends. All his friends are living as paying guest and it has been decided to first gather at Akshay's house and then move towards party location. The problem is that no one knows the exact address of his house in the city. Akshay as a computer science wizard knows how to apply his theory subjects in his real life and came up with an amazing idea to help his friends. He draws a graph by looking in to location of his house and his friends' location (as a node in the graph) on a map. He wishes to find out shortest distance and path covering that distance from each of his friend's location to his house and then whatsapp them this path so that they can reach his house in minimum time. Akshay has developed the program that implements Dijkstra's algorithm but not sure about correctness of results. Can you also implement the same algorithm and verify the correctness of Akshay's results? (Hint: Print shortest path and distance from friends' location to Akshay's house)

II. Design an algorithm and implement it using a program to solve previous question's problem using Bellman- Ford's shortest path algorithm.

Input Format:
Input will be the graph in the form of adjacency matrix or adjacency list.
Source vertex number is also provided as an input.

Output Format:
Output will contain V lines.
Each line will represent the whole path from destination vertex number to source vertex number along with minimum path weigth.

Sample I/O Problem I and II:

Input:	**Output:**
5 0 4 1 0 0 0 0 0 0 4 0 2 0 4 0 0 0 0 0 4 0 0 0 0 0 1	1 : 0 2 3 1 : 3 3 1 : 1 4 3 1 : 3 5 2 3 1 : 7

III. Given a directed graph with two vertices (source and destination). Design an algorithm and implement it using a program to find the weight of the shortest path from source to destination with exactly k edges on the path.

Input Format:
First input line will obtain number of vertices V present in the graph.
Graph in the form of adjacency matrix or adjacency list is taken as an input in next V lines.

Next input line will obtain source and destination vertex number.
Last input line will obtain value k.

Output Format:
Output will be the weigth of shortest path from source to destination having exactly k edges.
If no path is available then print "**no path of length k is available**".

Sample I/O Problem III:

Input:	**Output:**
4 0 10 3 2 0 0 0 7 0 0 0 6 0 0 0 0 1 4 2	Weight of shortest path from (1,4) with 2 edges : 9

Week 8:

Note: Input, output format along with sample input output for problem I and II is same and is provided at the end of problem II.

I. Assume that a project of road construction to connect some cities is given to your friend. Map of these cities and roads which will connect them (after construction) is provided to him in the form of a graph. Certain amount of rupees is associated with construction of each road. Your friend has to calculate the minimum budget required for this project. The budget should be designed in such a way that the cost of connecting the cities should be minimum and number of roads required to connect all the cities should be minimum (if there are N cities then only N-1 roads need to be constructed). He asks you for help. Now, you have to help your friend by designing an algorithm which will find minimum cost required to connect these cities. (use Prim's algorithm)

II. Implement the previous problem using Kruskal's algorithm.

Input Format:
The first line of input takes number of vertices in the graph.
Input will be the graph in the form of adjacency matrix or adjacency list.

Output Format:
Output will be minimum spanning weight

Sample I/O Problem I and II:

Input:	**Output:**
7 0 0 7 5 0 0 0 0 0 8 5 0 0 0 7 8 0 9 7 0 0 5 0 9 0 15 6 0 0 5 7 15 0 8 9 0 0 0 6 8 0 11 0 0 0 0 9 11 0	Minimum Spanning Weight: 39

III. Assume that same road construction project is given to another person. The amount he will earn from this project is directly proportional to the budget of the project. This person is greedy, so he decided to maximize the budget by constructing those roads who have highest construction cost. Design an algorithm and implement it using a program to find the maximum budget required for the project.

Input Format:
The first line of input takes number of vertices in the graph.
Input will be the graph in the form of adjacency matrix or adjacency list.

Output Format:
Out will be maximum spanning weight.

Sample I/O Problem III:

Input:	**Output:**
7 0 0 7 5 0 0 0 0 0 8 5 0 0 0 7 8 0 9 7 0 0 5 0 9 0 15 6 0 0 5 7 15 0 8 9 0 0 0 6 8 0 11 0 0 0 0 9 11 0	Maximum Spanning Weight: 59

Week 9:

I. Given a graph, Design an algorithm and implement it using a program to implement Floyd-Warshall all pair shortest path algorithm.

Input Format:
The first line of input takes number of vertices in the graph.
Input will be the graph in the form of adjacency matrix or adjacency list. If a direct edge is not present between any pair of vertex (u,v), then this entry is shown as AdjM[u,v] = INF.

Output Format:
Output will be shortest distance matrix in the form of V X V matrix, where each entry (u,v) represents shortest distance between vertex u and vertex v.

Sample I/O Problem I:

Input:	**Output:**
5 0 10 5 5 INF INF 0 5 5 5 INF INF 0 INF 10 INF INF INF 0 20 INF INF INF 5 0	Shortest Distance Matrix: 0 10 15 5 15 INF 0 5 5 5 INF INF 0 15 10 INF INF INF 0 20 INF INF INF 5 0

II. Given a knapsack of maximum capacity w. N items are provided, each having its own value and weight. You have to Design an algorithm and implement it using a program to find the list of the selected items such that the final selected content has weight w and has maximum value. You can take fractions of items,i.e. the items can be broken into smaller pieces so that you have to carry

only a fraction x_i of item i, where $0 \leq x_i \leq 1$.

Input Format:
First input line will take number of items N which are provided.
Second input line will contain N space-separated array containing weights of all N items.
Third input line will contain N space-separated array containing values of all N items.
Last line of the input will take the maximum capacity w of knapsack.

Output Format:
First output line will give maximum value that can be achieved.
Next Line of output will give list of items selected along with their fraction of amount which has been taken.

Sample I/O Problem II:

Input:	**Output:**
6 6 10 3 5 1 3 6 2 1 8 3 5 16	Maximum value : 22.33 item-weight 5-1 6-3 4-5 1-6 3-1

III. Given an array of elements. Assume arr[i] represents the size of file i. Write an algorithm and a program to merge all these files into single file with minimum computation. For given two files A and B with sizes m and n, computation cost of merging them is O(m+n). (Hint: use greedy approach)

Input Format:
First line will take the size n of the array.
Second line will take array s an input.

Output Format:
Output will be the minimum computation cost required to merge all the elements of the array.

Sample I/O Problem III:

Input:	**Output:**
10 10 5 100 50 20 15 5 20 100 10	960

Solved example: Consider arr[5] = { 10, 5, 100, 50, 20, 15}. As per the brute force approach, first of all merge first two files (having 10 and 5 file size).
Cost of merging will be = 10+5=15.
List will become {15, 100, 50, 20, 15}.
Similarly, again merging first two files (i.e. having 15 and 100 file size).
Cost of merging will be = 15+100=115.
List will become {115, 50, 20, 15}.
For the subsequent steps the list becomes, (165, 20, 15}, {185, 15} and {200}.
Therefore total cost of merging = 15+115+165+185+200 = 680.
But this is not minimum computation cost. To find minimum cost, consider the order arr[5] = {5, 10, 15, 20, 50, 100}. By applying the same approach, the total cost of merging = 15+30+50+100+200 = 395.

Week 10:

I. Given a list of activities with their starting time and finishing time. Your goal is to select maximum number of activities that can be performed by a single person such that selected activities must be non-conflicting. Any activity is said to be non-conflicting if starting time of an activity is greater than or equal to the finishing time of the other activity. Assume that a person can only work on a single activity at a time.

Input Format:
First line of input will take number of activities N.
Second line will take N space-separated values defining starting time for all the N activities.
Third line of input will take N space-separated values defining finishing time for all the N activities.

Output Format:
Output will be the number of non-conflicting activities and the list of selected activities.

Sample I/O Problem I:

Input:	**Output:**
10 1 3 0 5 3 5 8 8 2 12 4 5 6 7 9 9 11 12 14 16	No. of non-conflicting activities: 4 List of selected activities: 1, 4, 7, 10

II. Given a long list of tasks. Each task takes specific time to accomplish it and each task has a deadline associated with it. You have to design an algorithm and implement it using a program to find maximum number of tasks that can be completed without crossing their deadlines and also find list of selected tasks.

Input Format:
First line will give total number of tasks n.
Second line of input will give n space-separated elements of array representing time taken by each task.
Third line of input will give n space-separated elements of array representing deadline associated with each task.

Output Format:
Output will be the total number of maximum tasks that can be completed.

Sample I/O Problem II:

Input:	**Output:**
7 2 1 3 2 2 2 1 2 3 8 6 2 5 3	Max number of tasks = 4 Selected task numbers : 1, 2, 3, 6

III. Given an unsorted array of elements, design an algorithm and implement it using a program to find whether majority element exists or not. Also find median of the array. A majority element is an element that appears more than n/2 times, where n is the size of array.

Input Format:
First line of input will give size n of array.

Second line of input will take n space-separated elements of array.

Output Format:
First line of output will be '**yes**' if majority element exists, otherwise print '**no**'.
Second line of output will print median of the array.

Sample I/O Problem III:

Input:	**Output:**
9 4 4 2 3 2 2 3 2 2	yes 2