

Lab Manual

Course Title : Design and Analysis of Algorithms

1. 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: 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	Output: Present 3 Not Present 4 Present 3
---	---

2. 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 $\text{arr}[i] + \text{arr}[j] = \text{arr}[k]$.

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	No sequence found.
5	2, 7, 8
1 5 84 209 341	1, 6, 9
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	

3. 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	2
5	4
1 51 84 21 31	
20	
10	
24 71 16 92 12 28 48 14 20 22	
4	

4. 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: 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	Output: -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
--	--

5. 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: 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	Output: -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
---	---

6. 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	NO
5	YES
28 52 83 14 75	NO
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	

7. 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	21 23 32 45 46 65 76 89
8	comparisons = 16
23 65 21 76 46 89 45 32	inversions =
10	21 32 34 46 51 54 65 76 78 97
54 65 34 76 78 97 46 32 51 21	comparisons = 22
15	inversions =
63 42 223 645 652 31 324 22 553 12 54 65 86 46 325	12 22 31 42 46 54 63 65 86 223 324 325 553 645 652
	comparisons = 43
	inversions =

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

9. 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: 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	Output: a – 3 No Duplicates Present l - 4
---	---

10. 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: 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	Output: 10 40 No Such Element Exist
--	--

11. 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	10 10 34 55
34 76 10 39 85 10 55	
12	
30 55 34 72 10 34 10 89 11 30 69 51	

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 $V \times V$ 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 .

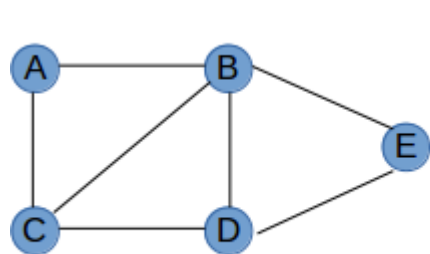


Figure (a)

	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)

12. 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: 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	Output: Yes Path Exists
--	-----------------------------------

13. 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: 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	Output: Not Bipartite
---	---------------------------------

14. 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: 5 0 1 1 0 0 0 0 0 1 1	Output: No Cycle Exists
--	-----------------------------------

0 1 0 1 0 0 0 0 0 1 0 0 0 0 0	
-------------------------------------	--

15. 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)

Design an algorithm and implement it using a program to solve 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 weighth.

Sample I/O Problem I and II:

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

16. 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 weight 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: 4 0 10 3 2 0 0 0 7 0 0 0 6 0 0 0 0 1 4 2	Output: Weight of shortest path from (1,4) with 2 edges : 9
---	---

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 $\text{AdjM}[u,v] = \text{INF}$.

Output Format:

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

Sample I/O Problem I:

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

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: 10 1 3 0 5 3 5 8 8 2 12 4 5 6 7 9 9 11 12 14 16	Output: 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: 7 2 1 3 2 2 2 1 2 3 8 6 2 5 3	Output: 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: 9 4 4 2 3 2 2 3 2 2	Output: yes 2
---	----------------------------

Week 11:

- I. Given a sequence of matrices, write an algorithm to find most efficient way to multiply these matrices together. To find the optimal solution, you need to find the order in which these matrices should be multiplied.

Input Format:

First line of input will take number of matrices n that you need to multiply.

For each line i in n, take two inputs which will represent dimensions aXb of matrix i.

Output Format:

Output will be the minimum number of operations that are required to multiply the list of matrices.

Sample I/O Problem I:

Input: 3 10 30 30 5 5 60	Output: 4500
---	------------------------

Solved Example: Consider a sequence of three matrices A of size 10X30, B of size 30X5, C of size 5X60. Then,

$(AB)C = (10*30*5) + (10*5*60) = 4500$ operations

$A(BC) = (30*5*60) + (10*30*60) = 27000$ operations.

Hence the output of the program must be 4500

Week 12:

- I. Given two sequences, Design an algorithm and implement it using a program to find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous.

Input Format:

First input line will take character sequence 1.

Second input line will take character sequence 2.

Output Format:

Output will be the longest common subsequence along with its length.

Sample I/O Problem I:

Input: Sequence1: AGGTAB Sequence2: GXTXAYB	Output: Longest Common Subsequence: GTAB length = 4
--	--

Solved Example: Consider two input strings “AGGTAB” and “GXTXAYB”. Then the length of longest common subsequence is 4 i.e. for subsequence “GTAB”.

- II. Given a knapsack of maximum capacity w . N items are provided, each having its own value and weight. 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 $\leq w$ and has maximum value. Here, you cannot break an item i.e. either pick the complete item or don't pick it. (0-1 property).

Input Format:

First line of input will provide number of items n .

Second line of input will take n space-separated integers describing weights for all items.

Third line of input will take n space-separated integers describing value for each item.

Last line of input will give the knapsack capacity.

Output Format:

Output will be maximum value that can be achieved and list of items selected along with their weight and value.

Sample I/O Problem I:

Input: 5 2 3 3 4 6 1 2 5 9 4 10	Output: Value = 16 Weights selected : 3 3 4 Values of selected weights : 2 5 9
--	--

Week 13:

- I. Given an array of characters, you have to find distinct characters from this array. Design an algorithm and implement it using a program to solve this problem using hashing. (Time Complexity = $O(n)$)

Input Format:

First line of input will give the size n of the character array.

Second line of input will give n space-separated elements to character array.

Output Format: Output will be the list of characters present in the array in alphabetical order and frequency of each character in the array.

Sample I/O Problem I:

Input: 20 a e d e f j t t z a z f t a e e k a e q	Output: a 4 d 1 e 5 f 2 j 1 k 1 q 1 t 3 z 2
--	---

Week 14: