

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

- 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	123
10	78
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	

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: 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
---	---

- 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: 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
--	--

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	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 .

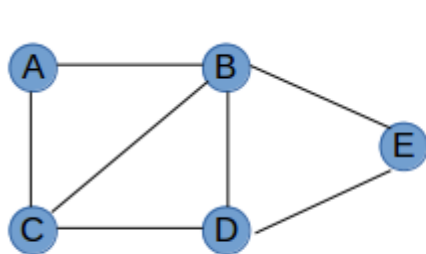


Figure (a)

	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)

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: 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
--	-----------------------------------

- 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: 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
---	---------------------------------

- 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: 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	

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 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
--	---

- 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.