Design and Analysis of Algorithms

Riya Bisht

November 21, 2021

Index

1 Week 1

1.1 Question 1

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 the total number of comparisons for each input case. Time complexity = $O(n)$, where n is the size of the input.

Program

```
#include<iostream>
using namespace std;
int main()
{

    int t;
    cin>>t;
    while (t--)
    {
        int n,key,comp=0;
        cin>>n;
        int a[n];
        for(int i=0;i<n;i++)
        cin>>a[i];
        cin>>key;
        for(int i=0;i<n;i++)
        { comp++;
        if(key==a[i])
            {
                cout<<"present"<<"  ";
                break;

            }
        }
        cout<<"no.of comparision "<<comp<<endl;


    }

    return 0;
}
```

Input

```
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
```

Output

```
present  no.of comparision 6
present  no.of comparision 3
no.of comparision 6
```

1.2 Question 2

Given an already shorted array of positive numbers, design an algorithm and implement it using a program to find whether given key element is presnet in the array or not. Also, find the total number of comparsions for each input case/ Time complexity = $O(n * log(n))$, where n is the size of input).

Program

```
#include<iostream>
using namespace std;
int main()
{
    int t;
    cin>>t;
    while (t--)
    {
        int n,key,flag=0;
        cin>>n;
        int a[n];
        for(int i=0;i<n;i++)
        {
            cin>>a[i];
        }
        cin>>key;
        int l_b=0,u_b=n-1,c=0;
        while(u_b >=l_b)
        {
            int mid=(u_b+l_b)/2;
            if(key == a[mid])
            {
                c++;
                cout<<"Present"<<" "<<c<<endl;
                flag=1;

                break;
            }
            else if(key<a[mid])
            {
                u_b=mid-1;
                c++;
            }
            else if(key>a[mid])
            {
                l_b=mid+1;
                c++;
            }

        }
        if(flag==0)
        cout<<"not present"<<" "<<c++<<endl;

    }

    return 0;
}
```

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
```

1.3 Question 3

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)

Program

```
#include <bits/stdc++.h>

using namespace std;

int linearSearch(int a[], int b, int e, int k)
{
    int no_of_comparison = 0;
    for (int i = b; i <= e; ++i)
    {
        ++no_of_comparison;
        if (a[i] == k)
        {
            cout << "Present " << no_of_comparison << endl;
            return 1;
        }
    }
    cout << "Not Present " << no_of_comparison << endl;
    return 0;
}

int exponentialSearch(int a[], int n, int key)
{
    if (a[0] == key)
    {
        cout << "Present 1" << endl;
        return 1;
    }
    int i = 1;
    while (i < n && a[i] < key)
        i *= 2;
    return linearSearch(a, i / 2, min(n - 1, i), key);
}

int main()
{

    int n;
    cin >> n;
    for (int z = 0; z < n; ++z)
    {
        int l;
        cin >> l;
        int a[l];
        for (int i = 0; i < l; ++i)
            cin >> a[i];
        int key;
        cin >> key;
        exponentialSearch(a, n, key);
    }
```

```
    return 0;
}
```

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

```
Not Present 1
Not Present 1
Not Present 1
```

2 Week 2

2.1 Question 1

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 = $O(logn)$)

Program

```
// 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))
```

```
#include<iostream>
#include<vector>
using namespace std;

int binary_search(int a[],int n,int key)
{
    int L_b=0;
    int U_b=n-1;
    while(U_b>=L_b)
    {
        int mid=(U_b+L_b)/2;

        if(a[mid]==key)
           return mid;

        else if(key<a[mid])
            U_b=mid-1;

        else
            L_b=mid+1;
    }

    return -1;
}
```

```
int main()
{
    int t;
    cin>>t;
    while (t--)
    {
        int n,x;
        cin>>n;
        int a[n];
        for(int i=0;i<n;i++)
        {
```

```
            cin>>a[i];
        }
        int key;
        cin>>key;


        int index=binary_search(a,n,key);

        if(index ==-1)
        cout<<"key not present"<<endl;
        else
        {

            int U_b=index;
            while (a[U_b+1]==key && U_b+1<n)
            {
                U_b++;
            }

            int L_b=index;

            while (a[L_b-1]==key && L_b-1>=0)
            {
                L_b--;
            }

            cout<<key<<"-"<<abs(U_b-L_b)+1<<endl;

        }

    }
    return 0;
}
```

Input

```
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
```

Output

```
981-2
75-3
```

2.2 Question 2

Find a squence of indices i ¡ j ¡ k in the sorted array such that arr[i]+arr[j]=arr[k]. Time Complexity: $O((n^2)logn)$, Space Complexity: $O(1)$

Program

```
#include<iostream>
//#include<algorithm>
using namespace std;
int binary_search(int a[],int L_b, int n,int key)
{
    int U_b=n-1;
    while(U_b>=L_b)
    {
        int mid=(U_b+L_b)/2;
        if(a[mid]==key)
        {
            return mid;
        }
        else if(key<a[mid])
        {
            U_b=mid-1;
        }
        else{
            L_b=mid+1;
        }

    }
    return -1;
}


int main()
{
    int t;
    cin>>t;
    while (t--)
    {
        int n;
        cin>>n;
        int a[n];
        for(int i=0;i<n;i++)
        {
            cin>>a[i];
        }

        for(int i=0;i<=n-3;i++)
        {
            //cout<<"hi";
            for(int j=i+1;j<n;j++)
            {
                int sum=0;
               // cout<<"in";
                sum=a[i]+a[j];
                //cout<<sum;
                if(sum>a[n-1])
                    break;
                int k=binary_search(a,j+1,n,sum);
```

```
                if(k!=-1)
                {cout<<i+1<<" "<<j+1<<" "<<k+1 <<endl;return 0;}

            }
        }
        cout<<" NOT Present"<<endl;

    }

return 0;
}
```

Input

```
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
```

Output

```
 NOT Present
2 7 8
```

2.3 Question 3

Given a key and an array return the count of pairs a,b such that a-b = key Time Complexity: O(N), Space Complexity: O(N)

Program

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{
    int t;
    cin>>t;
    while (t--)
    {
        int n,diff=0,count=0;
        cin>>n;
        vector<int>a;
        for(int i=0;i<n;i++)
        {
            int x;
            cin>>x;
            a.push_back(x);
        }
        int key;
        cin>>key;


        for (int i = 0; i <=n-2; i++)
        {
            for(int j=i+1;j<=n-1;j++)
            {
                if(abs(a[i]-a[j])==key)
                {count++;}
            }
        }
        cout<<count;

    }
    return 0;

}
```

Input

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

Output

```
24
```

3 Week 3

3.1 Question 1

Given an unsorted array, sort the array using insertion sort Time Complexity: $O(n^2)$, Space Complexity: $O(1)$

Program

```
#include <bits/stdc++.h>

using namespace std;

void insertionSort(int arr[], int l)
{
    int noc = 0, shift = 0;
    for (int i = 1; i < l; ++i)
    {
        int temp = arr[i], j = i - 1;
        while (++noc && j >= 0 && arr[j] > temp)
        {
            ++shift;
            arr[j + 1] = arr[j];
            --j;
        }
        arr[j + 1] = temp;
    }
    for (int i = 0; i < l; ++i)
        cout << arr[i] << " ";
    cout << endl;
    cout << "comparisons = " << noc << endl;
    cout << "shifts = " << shift << endl;
}

int main()
{

    int no;
    cin >> no;
    for (int f = 0; f < no; f++)
    {
        int l;
        cin >> l;
        int arr[l];
        for (int i = 0; i < l; ++i)
            cin >> arr[i];
        insertionSort(arr, l);
    }

    return 0;
}
```

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 = 20
shifts = 13
21 32 34 46 51 54 65 76 78 97
comparisons = 37
shifts = 28
-12 22 31 42 46 54 63 65 86 223 324 325 553 645 652
comparisons = 68
shifts = 54
```

3.2 Question 2

Given an unsorted array, sort the array using selection sort Time Complexity: $O(n^2)$, Space Complexity: $O(1)$

Program

```
// This algorithm we are selecting a position and finding an element for
   that position

#include <iostream>
using namespace std;
int main()
{
    int t;
    cin>>t;
    while (t--)
    {


    int n;
    cin >> n;
    int * arr = new int [n];
    for (int i = 0; i < n; i++)
        cin >> arr[i];

    int c_comp = 0, C_swap = 0;
    for (int i = 0; i < n - 1; i++)
    {
        int k = i, j = i + 1;
        for (j; j < n; j++)
        {
            c_comp++;
            if (arr[k] > arr[j])
                k = j;
        }
        C_swap++;
        int temp = arr[i];
        arr[i] = arr[k];
        arr[k] = temp;
    }
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
    cout << " No of comparision :: " << c_comp << endl;
    cout << " No of swaps :: " << C_swap << endl;
    }

    return 0;
}
```

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
 No of comparision :: 28
 No of swaps :: 7
21 32 34 46 51 54 65 76 78 97
 No of comparision :: 45
 No of swaps :: 9
12 22 31 42 46 54 63 65 86 223 324 325 553 645 652
 No of comparision :: 105
 No of swaps :: 14
```

3.3 Question 3

Given an unsorted array containing 0 or more duplicates, find whether there are any duplicates or not Time Complexity: O(nlogn), Space Complexity: O(1)

Program

```
#include <iostream>
using namespace std;
void swap(int &x, int &y)
{
    int temp = x;
    x = y;
    y = temp;
}

int partioning(int a[], int l, int h)
{

    // cout<<"l"<<l<<"h"<<h;
    int pivot = a[l];
    int i = l;
    int j = h;

    do
    {

        do
        {
            ++i;
        } while (a[i] <= pivot);
        do
        {
            --j;
        } while (a[j] > pivot);

        if (i < j)
            swap(a[i], a[j]);

    } while (i < j);
    // cout<<j<<" "<<a[j]<<"pivot ::"<<pivot<<endl;
    swap(a[j], a[l]);
    //cout<<j<<" ";
    return j;
}
void Quick_sort(int a[], int l, int h)
{
    if (l < h)
    {
        int j = partioning(a, l, h);
        //cout<<j;
        Quick_sort(a, l, j);
        Quick_sort(a, j + 1, h);
    }
}
int main()
{
    int t;
    cin >> t;
```

```
    while (t--)
    {

        int n;
        cin >> n;
        n=n+1;
        int *a = new int[n];
        for (int i = 0; i < n-1; i++)
            cin >> a[i];

        a[n-1] = INT32_MAX;
        Quick_sort(a, 0, n-1);


        int flag = 0;
        for (int i = 0; i < n - 2; i++)
        {
            if (a[i] == a[i + 1])
            {
                flag = 1;
                break;
            }
        }
        if (flag == 1)
            cout << "YES" << endl;
        else
            cout << "NO" << endl;
    }
    return 0;
}
```

Input

```
3
5
28 52 83 14 75
10
75 65 1 65 2 86 2 75 8 7
15
75 35 86 57 98 23 73 1 64 8 11 90 61 19 20
```

Output

```
NO
YES
NO
```

4 Week 4

4.1 Question 1

Sort an array using MergeSort and calculate the number of comparisons made Also calculate the number of inversions present in the array Time Complexity: O(nlogn), Space Complexity: O(n)

Program

```
#include <bits/stdc++.h>

using namespace std;

int NComparison, NInversion;

void mergeArr(int arr[], int l, int s, int h)
{
    int n = h - l + 1, temp[n], i = l, j = s + 1, k = 0, noi = 0;
    while (i <= s && j <= h)
    {
        NComparison++;
        if (arr[i] <= arr[j])
            temp[k++] = arr[i++];
        else
        {
            temp[k++] = arr[j++];
            NInversion += s - i + 1;
        }
    }
    for (; i <= s; ++i, ++k)
        temp[k] = arr[i];
    for (; j <= h; ++j, ++k)
        temp[k] = arr[j];

    for (int i = l, j = 0; i <= h; ++i, j++)
        arr[i] = temp[j];
}

void mergeSort(int arr[], int l, int h)
{
    int noc = 0;
    if (l < h)
    {
        int splitInd = (l + h) / 2;
        mergeSort(arr, l, splitInd);
        mergeSort(arr, splitInd + 1, h);
        mergeArr(arr, l, splitInd, h);
    }
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int no, p = 0;
    cin >> no;
    for (int f = 0; f < no; f++)
```

```
    {
        NComparison = NInversion = 0;
        int n;
        cin >> n;
        int arr[n];
        for (int i = 0; i < n; ++i)
            cin >> arr[i];
        mergeSort(arr, 0, n - 1);
        for (int i = 0; i < n; ++i)
            cout << arr[i] << " ";
        cout << "\ncomparisons = " << NComparison << endl;
        cout << "inversions = " << NInversion << endl;
    }

    return 0;
}
```

Input

```
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
```

Output

```
21 23 32 45 46 65 76 89
comparisons = 16
inversions = 13
21 32 34 46 51 54 65 76 78 97
comparisons = 22
inversions = 28
12 22 31 42 46 54 63 65 86 223 324 325 553 645 652
comparisons = 43
inversions = 54
```

4.2 Question 2

Sort an array using QuickSort and calculate the number of swaps made Time Complexity: O(nlogn), Space Complexity: O(n)

Program

```
#include <bits/stdc++.h>

using namespace std;

int NComparison, NSwap;

int makePivot(int arr[], int l, int h)
{
    int p = arr[h];
    int i = l - 1, j = l;
    for (; j < h; ++j)
    {
        ++NComparison;
        if (arr[j] <= p)
        {
            ++NSwap;
            int t = arr[++i];
            arr[i] = arr[j];
            arr[j] = t;
        }
    }
    NSwap++;
    int t = arr[++i];
    arr[i] = arr[j];
    arr[j] = t;
    return i;
}

void quickSort(int arr[], int l, int h)
{
    if (l < h)
    {
        int pivot = makePivot(arr, l, h);
        quickSort(arr, l, pivot - 1);
        quickSort(arr, pivot + 1, h);
    }
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int no;
    cin >> no;
    for (int f = 0; f < no; f++)
    {
        NComparison = NSwap = 0;
        int n;
        cin >> n;
        int arr[n];
        for (int i = 0; i < n; ++i)
```

```
            cin >> arr[i];
        quickSort(arr, 0, n - 1);
        for (int i = 0; i < n; ++i)
            cout << arr[i] << " ";
        cout << "\nComparisons = " << NComparison << endl;
        cout << "Swaps = " << NSwap << endl;
    }

    return 0;
}
```

Input

```
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
```

Output

```
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
```

4.3 Question 3

Given an unsorted array, find out the k-th smallest number Time Complexity: O(n), Space Complexity: O(1)

Program

```
#include <bits/stdc++.h>

using namespace std;

int findKSmallest(int arr[], int l, int h, int k)
{
    int p = arr[h], i = l - 1, j = l;
    for (; j <= h; ++j)
    {
        if (arr[j] <= p)
        {
            int t = arr[++i];
            arr[i] = arr[j];
            arr[j] = t;
        }
    }
    if (i == k)
        return arr[i];
    else if (i < k)
        return findKSmallest(arr, i + 1, h, k);
    else
        return findKSmallest(arr, l, i - 1, k);
}

int main()
{

    int no;
    cin >> no;
    for (int f = 0; f < no; f++)
    {
        int n;
        cin >> n;
        int arr[n];
        for (int i = 0; i < n; ++i)
            cin >> arr[i];
        int k;
        cin >> k;
        k = k <= n ? k - 1 : n - 1;
        cout << findKSmallest(arr, 0, n - 1, k) << endl;
    }

    return 0;
}
```

Input

```
2
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
```

Output

```
123
78
```

5 Week 5

5.1 Question 1

Given an unsorted array of alphabets containing duplicate elements. Find the alphabet with maximum frequency Time Complexity: $O(n)$, Space Complexity: $O(1)$

Program

```
#include <bits/stdc++.h>

using namespace std;

int main()
{

    int no;
    cin >> no;
    for (int f = 0; f < no; f++)
    {
        int n;
        cin >> n;
        int arr[26] = {0};
        for (int i = 0; i < n; ++i)
        {
            char x;
            cin >> x;
            arr[x - 'a']++;
        }
        int ind = -1, max = INT_MIN;
        for (int i = 0; i < 26; ++i)
        {
            if (arr[i] != 0 && arr[i] != 1 && arr[i] > max)
            {
                ind = i;
                max = arr[i];
            }
        }
        if (ind >= 0)
            cout << char('a' + ind) << " - " << arr[ind] << endl;
        else
            cout << "No Duplicates Present" << endl;
    }
    return 0;
}
```

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
```

5.2 Question 2

Given an unsorted array of integers, find whether two elements exist such that their sum is equal to the given key element. Time Complexity: $O(nlogn)$, Space Complexity: $O(n)$

Program

```
#include <bits/stdc++.h>

using namespace std;

void mergeArr(int arr[], int l, int s, int h)
{
    int n = h - l + 1, temp[n], i = l, j = s + 1, k = 0, noi = 0;
    while (i <= s && j <= h)
    {
        if (arr[i] <= arr[j])
            temp[k++] = arr[i++];
        else
        {
            temp[k++] = arr[j++];
        }
    }
    for (; i <= s; ++i, ++k)
        temp[k] = arr[i];
    for (; j <= h; ++j, ++k)
        temp[k] = arr[j];
    for (int i = l, j = 0; i <= h; ++i, j++)
        arr[i] = temp[j];
}

void mergeSort(int arr[], int l, int h)
{
    int noc = 0;
    if (l < h)
    {
        int splitInd = (l + h) / 2;
        mergeSort(arr, l, splitInd);
        mergeSort(arr, splitInd + 1, h);
        mergeArr(arr, l, splitInd, h);
    }
}

void findPair(int arr[], int l, int h, int k)
{
    int flag = 1;
    while (l <= h)
    {
        if (arr[l] + arr[h] > k)
            h--;
        else
        {
            if (arr[l] + arr[h] == k)
            {
                cout << arr[l] << " " << arr[h] << ", ";
                flag = 0;
            }
            l++;
        }
```

```
    }
    if (flag)
        cout << "No such pair exist";
    cout << endl;
}

int main()
{

    int no;
    cin >> no;
    for (int f = 0; f < no; f++)
    {
        int n;
        cin >> n;
        int arr[n];
        for (int i = 0; i < n; ++i)
            cin >> arr[i];
        int key;
        cin >> key;
        mergeSort(arr, 0, n - 1);
        findPair(arr, 0, n - 1, key);
    }
    return 0;
}
```

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, 12 38,
No such pair exist
```

5.3 Question 3

Given 2 sorted arrays of size m and n respectively Find out common elements among the arrays. Time Complexity: $O(m+n)$, Space Complexity: $O(1)$

Program

```
#include <bits/stdc++.h>

using namespace std;

int main()
{

    int m;
    cin >> m;
    int arr[m];
    for (int i = 0; i < m; ++i)
        cin >> arr[i];
    int n;
    cin >> n;
    int arr2[n];
    for (int i = 0; i < n; ++i)
        cin >> arr2[i];
    int i, j;
    i = j = 0;
    while (i != m && j != n)
    {
        if (arr[i] == arr2[j])
        {
            cout << arr[i++] << " ";
            j++;
        }
        else if (arr[i] < arr2[j])
            ++i;
        else
            j++;
    }
    return 0;
}
```

Input

```
7
10 10 34 39 55 76 85
12
10 10 11 30 30 34 34 51 55 69 72 89
```

Output

```
10 10 34 55
```

6 Week 6

6.1 Question 1

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).

Program

```
#include <bits/stdc++.h>

using namespace std;

void markVisited(int **G, int n, int k)
{
    for (int i = 0; i < n; ++i)
        G[i][k] = 2;
}

bool dfs(int **G, int n, int s, int d)
{
    vector<int> st;
    st.push_back(s);
    markVisited(G, n, s);
    while (!st.empty())
    {
        int x = st.back();
        st.pop_back();
        if (G[x][d] == 1)
            return true;
        else
        {
            for (int i = 0; i < n; ++i)
                if (G[x][i] == 1)
                    st.push_back(i);
            markVisited(G, n, x);
        }
    }
    return false;
}

int main()
{
    int n;
    cin >> n;

    int **arr;
    arr = (int **)malloc(n * sizeof(int *));

    for (int i = 0; i < n; ++i)
        arr[i] = (int *)malloc(n * sizeof(int));

    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            cin >> arr[i][j];

    int s, d;
    cin >> s >> d;
    bool isPresent = dfs(arr, n, s - 1, d - 1);
```

```
    if (isPresent)
        cout << "Yes Path Exists" << endl;
    else
        cout << "No Path Does Not Exist" << endl;
    return 0;
}
```

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
```

6.2 Question 2

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

Program

```
#include <bits/stdc++.h>

using namespace std;

void markVisited(int **G, int n, int k)
{
    for (int i = 0; i < n; ++i)
        if (G[i][k] == 1)
            G[i][k] = 2;
}

bool bipartite(int **G, int n)
{
    queue<int> Queue;
    Queue.push(0);
    int color[n] = {1};
    markVisited(G, n, 0);
    while (!Queue.empty())
    {
        int u = Queue.front();
        Queue.pop();
        int curCol = color[u] * -1;
        for (int i = 0; i < n; ++i)
        {
            if (G[u][i] != 0)
            {
                if (color[i] == 0)
                    color[i] = curCol;
                else if (color[i] != curCol)
                    return false;
                if (G[u][i] == 1)
                {
                    Queue.push(i);
                    markVisited(G, n, i);
                }
            }
        }
    }
    return true;
}

int main()
{


    int n;
    cin >> n;

    int **arr;
    arr = (int **)malloc(n * sizeof(int *));

    for (int i = 0; i < n; ++i)
```

```
        arr[i] = (int *)malloc(n * sizeof(int));

    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            cin >> arr[i][j];

    int s, d;
    cin >> s >> d;
    bool isBipartite = bipartite(arr, n);
    if (isBipartite)
        cout << "Bipartite" << endl;
    else
        cout << "Not Bipartite" << endl;
    return 0;
}
```

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
```

7 Week 7

7.1 Question 1

Use Dijkstra's Algorithm to find the shortest path to all vertices from a given source vertex Time Complexity: $O(ElogE)$ for(adjacency list) and $O(V^2)$ for adjacency matrix, Space Complexity: $O(V)$

Program

```
#include <bits/stdc++.h>
#define INF INT_MAX
#define NIL -1
using namespace std;

void printParent(vector<int> &parent, int u)
{
    cout << u + 1 << " ";
    if (parent[u] >= 0)
        printParent(parent, parent[u]);
}

void djikstra(int **Graph, int V, int src)
{
    vector<int> parent(V, NIL);
    vector<int> weight(V, INF);
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<
   int, int>>> min_heap;
    weight[src] = 0;
    min_heap.push(make_pair(weight[src], src));
    while (!min_heap.empty())
    {
        int u = min_heap.top().second;
        min_heap.pop();
        for (int v = 0; v < V; ++v)
        {
            if (Graph[u][v] != 0)
            {
                if (weight[v] > weight[u] + Graph[u][v])
                {
                    weight[v] = weight[u] + Graph[u][v];
                    min_heap.push(make_pair(weight[v], v));
                    parent[v] = u;
                }
            }
        }
    }

    for (int i = 0; i < V; ++i)
    {
        printParent(parent, i);
        if (weight[i] == INF)
            cout << ": INF" << endl;
        else
            cout << ": " << weight[i] << endl;
    }
}

int main()
{
```

```
    int V;
    cin >> V;
    int **Graph = (int **)malloc(V * sizeof(int *));
    for (int i = 0; i < V; ++i)
        Graph[i] = (int *)malloc(V * sizeof(int));

    for (int i = 0; i < V; ++i)
        for (int j = 0; j < V; ++j)
            cin >> Graph[i][j];

    int src;
    cin >> src;
    djikstra(Graph, V, src - 1);
    return 0;
}
```

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 : 5
5 2 3 1 : 7
```

7.2 Question 2

Use Bellman Ford's Algorithm to find the shortest path to all vertices from a given source vertex Time Complexity: $O(E * (V - 1))$ for edge list representation and $O(V^2 * (V - 1))$ for adjacency matrix Space Complexity: $O(V)$

Program

```
#include <bits/stdc++.h>
#define INF INT_MAX
#define NIL -1
using namespace std;

void printParent(vector<int> &parent, int u)
{
    cout << u + 1 << " ";
    if (parent[u] >= 0)
        printParent(parent, parent[u]);
}

void bellmanFord(int **Graph, int V, int src)
{
    vector<int> weight(V, INF);
    vector<int> parent(V, NIL);
    weight[src] = 0;
    for (int k = 0; k < V - 1; ++k)
    {
        for (int u = 0; u < V; ++u)
        {
            for (int v = 0; v < V; ++v)
            {
                if (Graph[u][v] != 0)
                {
                    if (weight[u] != INF && weight[v] > weight[u] + Graph[
   u][v])
                    {
                        weight[v] = weight[u] + Graph[u][v];
                        parent[v] = u;
                    }
                }
            }
        }
    }

    for (int i = 0; i < V; ++i)
    {
        printParent(parent, i);
        if (weight[i] == INF)
            cout << ": INF" << endl;
        else
            cout << ": " << weight[i] << endl;
    }
}

int main()
{

    int V;
    cin >> V;
```

```
    int **Graph;
    Graph = (int **)malloc(V * sizeof(int *));
    for (int i = 0; i < V; ++i)
        Graph[i] = (int *)malloc(V * sizeof(int));
    for (int i = 0; i < V; ++i)
        for (int j = 0; j < V; ++j)
            cin >> Graph[i][j];
    int src;
    cin >> src;
    bellmanFord(Graph, V, src - 1);
    return 0;
}
```

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 : 5
5 2 3 1 : 7
```

7.3 Question 3

Find the smallest weight of a path from source vertex to destination vertex of length exactly k Time Complexity: $O(k * V^3)$ Space Complexity: $O(k * n^2)$

Program

```
#include <bits/stdc++.h>

#define ll long long
#define INF INT_MAX

using namespace std;

ll SDKedge(ll **Graph, int V, int u, int v, int k)
{
    if (k <= 0)
        return INF;
    else if (k == 0)
    {
        if (u == v)
            return 0;
        else
            return INF;
    }
    else if (k == 1)
    {
        if (u != v)
            return Graph[u][v];
        else
            return INF;
    }

    ll ans = INF;
    for (int i = 0; i < V; ++i)
        if (i != u && i != v && Graph[u][i] != 0)
            ans = min(ans, Graph[u][i] + SDKedge(Graph, V, i, v, k - 1));
    return ans;
}

int main()
{

    int V;
    cin >> V;
    ll **Graph = (ll **)malloc(V * sizeof(ll *));
    for (int i = 0; i < V; ++i)
        Graph[i] = (ll *)malloc(V * sizeof(ll));
    for (int i = 0; i < V; ++i)
        for (int j = 0; j < V; ++j)
            cin >> Graph[i][j];
    int u, v, k;
    cin >> u >> v >> k;

    ll ans = SDKedge(Graph, V, u - 1, v - 1, k);

    if (ans == INF)
        cout << "No Path of length K is available" << endl;
    else
```

```
        cout << "Weight of shortest path from (" << u << "," << v << ")
    with " << k << " edges : " << ans << endl;

    return 0;
}
```

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
```

8 Week 8

8.1 Question 1

Use Prims Algorithm to Calculate Minimum Spanning Tree of a graph Time Complexity: $O((V+E)*log(V))$
Space Complexity: $O(V+E)$

Program

```
#include <bits/stdc++.h>

#define ll long long
#define INF INT_MAX

using namespace std;

int prims(int **arr, int n)
{
    vector<bool> visited(n, false);
    vector<int> weight(n, INF);
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<
   int, int>>> min_heap;
    int src = 0;
    weight[src] = 0;
    min_heap.push(make_pair(weight[src], src));
    while (!min_heap.empty())
    {
        int u = min_heap.top().second;
        min_heap.pop();
        if (!visited[u])
        {
            visited[u] = true;
            for (int v = 0; v < n; ++v)
            {
                if (!visited[v] && arr[u][v] != 0 && arr[u][v] < weight[v
   ])
                {
                    weight[v] = arr[u][v];
                    min_heap.push(make_pair(weight[v], v));
                }
            }
        }
    }

    int sum = 0;
    for (auto i : weight)
        sum += i;
    return sum;
}

int main()
{

    int n;
    cin >> n;
    int **arr;
    arr = (int **)malloc(n * sizeof(int *));
    for (int i = 0; i < n; ++i)
        arr[i] = (int *)malloc(n * sizeof(int));
```

```
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            cin >> arr[i][j];
    cout << "Minimum spanning weight : " << prims(arr, n);
    return 0;
}
```

Input

```
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
```

Output

```
Minimum spanning weight : 39
```

8.2 Question 2

Use Kruskal's Algorithm to Calculate Minimum Spanning Tree of a graph Time Complexity: $O(E * log(E))$ Space Complexity: $O(V + E)$

Program

```
#include <bits/stdc++.h>
using namespace std;

// Use Kruskal's Algorithm to Calculate Minimum Spanning Tree of a graph
// Time Complexity: O(E*log(E))
// Space Complexity: O(V+E)

struct UF{
    vector<int> e;
    UF(int n){ e.assign(n, -1); }
    bool sameSet(int a, int b) {return find(a) == find(b);}
    int size(int x) { return -e[find(x)];}
    int find(int x) {return e[x] < 0? x: e[x] = find(e[x]);}
    bool join(int a, int b){
        a  = find(a); b = find(b);
        if(a == b) return false;
        if(e[a] > e[b]) swap(a,b);
        e[a] += e[b];
        e[b] = a;
        return true;
    }
};

struct Edge{
    int weight;
    int a,b;
    Edge(int aa, int bb, int w){
        weight = w;
        a = aa;
        b = bb;
    }
};

int kruskalsMST(vector<vector<int>> &g){
    int n = g.size();
    UF uf(n+1);
    vector<Edge> edges;
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            if(g[i][j] == 0) continue;
            Edge temp(i, j, g[i][j]);
            edges.push_back(temp);
        }
    }
    sort(edges.begin(), edges.end(), [&](Edge &a, Edge &b){
        return a.weight < b.weight;
    });
    int ans = 0;
    int c = 0;
    for(int i=0;i<edges.size();i++){
        int a = edges[i].a;
        int b = edges[i].b;
```

```
        int w = edges[i].weight;
        if(!uf.sameSet(a,b)){
            ans += w;
            uf.join(a,b);
            c++;
        }
        if(c == n-1) break;
    }
    return ans;
}


int main(){


    int n;
    cin >> n;
    vector<vector<int>> g(n+1, vector<int> (n+1, 0));
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            cin >> g[i][j];
        }
    }

    int ans = kruskalsMST(g);
    cout << "Minimum Spanning Weight: ";
    cout << ans << '\n';

    return 0;
}
```

Input

```
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
```

Output

```
Minimum Spanning Weight: 37
```

8.3 Question 3

Calculate Maximum Spanning Tree of a graph Time Complexity: $O(E*log(V))$ Space Complexity: $O(V+E)$

Program

```
#include <bits/stdc++.h>
#define NIL -1

using namespace std;

int findParent(vector<int> parent, int u)
{
    if (parent[u] < 0)
        return u;
    return findParent(parent, parent[u]);
}

bool UnionByWeight(vector<int> &parent, int u, int v)
{
    int pu = findParent(parent, u);
    int pv = findParent(parent, v);
    if (pu != pv)
    {
        if (parent[pu] <= parent[pv])
        {
            parent[pu] += parent[pv];
            parent[pv] = pu;
        }
        else
        {
            parent[pv] += parent[pu];
            parent[pu] = pv;
        }
        return true;
    }
    return false;
}

int kruskals(int **graph, int n)
{
    vector<pair<int, pair<int, int>>> G;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (graph[i][j] != 0)
                G.push_back(make_pair(graph[i][j], make_pair(i, j)));
    sort(G.begin(), G.end(), greater<pair<int, pair<int, int>>>());
    vector<int> parent(n, NIL);
    int s = 0;
    for (auto i : G)
    {
        int u = i.second.first;
        int v = i.second.second;
        int w = i.first;
        if (UnionByWeight(parent, u, v))
            s += w;
    }
    return s;
}
```

```
int main()
{

    int n;
    cin >> n;
    int **graph;
    graph = (int **)malloc(n * sizeof(int *));
    for (int i = 0; i < n; ++i)
        graph[i] = (int *)malloc(n * sizeof(int));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            cin >> graph[i][j];
    cout << "Minimum spanning weight : " << kruskals(graph, n) << endl;
    return 0;
}
```

Input

```
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
```

Output

```
Minimum spanning weight : 59
```

9 Week 9

9.1 Question 1

Implement Floyd-Warshall Shortest path algorithm Time Complexity: $O(n^3)$ Space Complexity: $O(1)$

Program

```
//Floyd Warshall Undirected Graph

#include <bits/stdc++.h>
#define INF INT_MAX

using namespace std;

#define ll long long

ll **floydWarshall(ll **Graph, ll V)
{
    ll **result = (ll **)malloc(V * sizeof(ll *));
    for (ll i = 0; i < V; ++i)
        result[i] = (ll *)malloc(V * sizeof(ll));

    for (ll i = 0; i < V; ++i)
        for (ll j = 0; j < V; ++j)
            result[i][j] = Graph[i][j];

    for (ll i = 0; i < V; ++i)
        for (ll j = 0; j < V; ++j)
            for (ll k = 0; k < V; ++k)
                if (result[i][k] != INF && result[k][j] && result[i][j] >
   result[i][k] + result[k][j])
                    result[i][j] = result[i][k] + result[k][j];

    return result;
}

int main()
{

    ll V;
    cin >> V;

    ll **Graph = (ll **)malloc(V * sizeof(ll *));
    for (ll i = 0; i < V; ++i)
        Graph[i] = (ll *)malloc(V * sizeof(ll));

    for (ll i = 0; i < V; ++i)
        for (ll j = 0; j < V; ++j)
            if (i != j)
                Graph[i][j] = INF;
            else
                Graph[i][j] = 0;

    for (int i = 0; i < V; ++i)
    {
        for (int j = 0; j < V; ++j)
        {
            ll x;
```

```
            cin >> x;
            if (i != j && x != 0)
                Graph[i][j] = x;
        }
    }

    ll **result = floydWarshall(Graph, V);

    cout << "Shortest Distance Matrix:" << endl;
    for (ll i = 0; i < V; ++i)
    {
        for (ll j = 0; j < V; ++j)
            if (result[i][j] != INF)
                cout << result[i][j] << "\t";
            else
                cout << "INF"
                     << "\t";
        cout << endl;
    }

    return 0;
}
```

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  5   5   INF
INF 0   INF INF INF
INF INF 0   INF INF
INF INF INF 0   INF
INF INF INF INF 0
```

9.2 Question 2

Implement Fractional Knapsack Time Complexity: $O(n \log n)$ Space Complexity: $O(n)$

Program

```
#include <bits/stdc++.h>
using namespace std;
bool compare(pair<pair<int, int>, int> a, pair<pair<int, int>, int> b) {
    return a.first.second * (1.0 / a.first.first) > b.first.second * (1.0
   / b.first.first);
}
int main() {
    int n;
    cin >> n; // no of items
    int W[n], V[n];
    for (int i = 0; i < n; i++)
        cin >> W[i];
    for (int i = 0; i < n; i++)
        cin >> V[i];
    int k;
    cin >> k;

    vector<pair<pair<int, int>, int>> items;

    for (int i = 0; i < n; i++) {
        items.push_back(make_pair(make_pair(W[i], V[i]), i + 1));
    }
    sort(items.begin(), items.end(), compare);

    float profit = 0.00;
    vector<pair<int, int>> selected;

    for (int i = 0; i < n; i++) {
        if (items[i].first.first <= k) {
            selected.push_back({items[i].second, items[i].first.first});
            k -= items[i].first.first;
            profit += items[i].first.second;
        } else {

            profit += (k) * (items[i].first.second * (1.0 / items[i].first
   .first));
            selected.push_back({items[i].second, k});

            break;
        }
    }

    cout << "Maximum value::" << profit << endl;
    cout << "item weight::" << endl;
    for (auto i : selected) {
        cout << i.first << " " << i.second << endl;
    }

    return 0;
}
```

Input

```
6
6 10 3 5 1 3
6 2 1 8 3 5
16
```

Output

```
Maximum value::22.3333
item weight::
5 1
6 3
4 5
1 6
3 1
```

9.3 Question 3

Given an array of integers representing sizes of files, merge them in such a way that the time required to merge them is mininum Two files take $a_i + a_j$ amount of time for any $1 <= i < j <= n$ Time Complexity: $O(n \log n)$ Space Complexity: $O(n)$

Program

```
//Floyd Warshall Undirected Graph

#include <bits/stdc++.h>
#define INF INT_MAX

using namespace std;

#define ll long long

ll **floydWarshall(ll **Graph, ll V)
{
    ll **result = (ll **)malloc(V * sizeof(ll *));
    for (ll i = 0; i < V; ++i)
        result[i] = (ll *)malloc(V * sizeof(ll));

    for (ll i = 0; i < V; ++i)
        for (ll j = 0; j < V; ++j)
            result[i][j] = Graph[i][j];

    for (ll i = 0; i < V; ++i)
        for (ll j = 0; j < V; ++j)
            for (ll k = 0; k < V; ++k)
                if (result[i][k] != INF && result[k][j] && result[i][j] >
   result[i][k] + result[k][j])
                    result[i][j] = result[i][k] + result[k][j];

    return result;
}

int main()
{
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif
    ll V;
    cin >> V;

    ll **Graph = (ll **)malloc(V * sizeof(ll *));
    for (ll i = 0; i < V; ++i)
        Graph[i] = (ll *)malloc(V * sizeof(ll));

    for (ll i = 0; i < V; ++i)
        for (ll j = 0; j < V; ++j)
            if (i != j)
                Graph[i][j] = INF;
            else
                Graph[i][j] = 0;

    for (int i = 0; i < V; ++i)
    {
```

```
        for (int j = 0; j < V; ++j)
        {
            ll x;
            cin >> x;
            if (i != j && x != 0)
                Graph[i][j] = x;
        }
    }

    ll **result = floydWarshall(Graph, V);

    cout << "Shortest Distance Matrix:" << endl;
    for (ll i = 0; i < V; ++i)
    {
        for (ll j = 0; j < V; ++j)
            if (result[i][j] != INF)
                cout << result[i][j] << "\t";
            else
                cout << "INF"
                    << "\t";
        cout << endl;
    }

    return 0;
}
```

Input

```
10
10 5 100 50 20 15 5 20 100 10
6
5 10 15 20 50 100
```

Output

10 Week 10

10.1 Question 1

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. Time Complexity: $O(nlogn)$ Space Complexity: $O(n)$

Program

```
#include <bits/stdc++.h>
using namespace std;
int main() {

    int n;
    cin >> n;
    vector<pair<int, pair<int, int>>> arr(n);
    for (int i = 0; i < n; i++) {
        cin >> arr[i].second.first;
    }
    for (int i = 0; i < n; i++) {
        cin >> arr[i].first;
        arr[i].second.second = i + 1;
    }
    sort(arr.begin(), arr.end());
    int prev = -1;
    vector<int> ans;
    for (int i = 0; i < n; i++) {
        if (arr[i].second.first > prev) {
            prev = arr[i].first;
            ans.push_back(arr[i].second.second);
        }
    }
    cout << "No. of non-conflicting activities: " << ans.size() << '\n';
    cout << "List of selected activities: ";
    for (auto &x : ans) {
        cout << x << ' ';
    }
    cout << '\n';

    return 0;
}
```

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
```

10.2 Question 2

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. Time Complexity: $O(nlogn)$ Space Complexity: $O(1)$

Program

```
#include <bits/stdc++.h>

using namespace std;

typedef vector<pair<pair<int, int>, int>> pairv;

int main()
{
    int n;
    cin >> n;
    int duration[n], deadLine[n], maxD = -1;
    for (int i = 0; i < n; ++i)
        cin >> duration[i];
    for (int i = 0; i < n; ++i)
    {
        cin >> deadLine[i];
        maxD = max(deadLine[i], maxD);
    }
    pairv job;
    for (int i = 0; i < n; ++i)
        job.push_back(make_pair(make_pair(deadLine[i], duration[i]), i +
   1));
    sort(job.begin(), job.end());
    int available[maxD], visited[maxD] = {0}, count = 0;
    for (int i = 0; i < maxD; ++i)
        available[i] = i + 1;
    vector<int> selected;
    for (auto i : job)
    {
        int du = i.first.second, de = i.first.first - 1, ind = i.second;
        if (available[de] >= du)
        {
            for (int j = maxD - 1; j > de; --j)
                available[j] -= du;
            int x = 0;
            while (x != du)
            {
                available[de] -= du - x;
                if (available[de] > 0 && !visited[de])
                    ++x;
                --de;
            }
            ++count;
            selected.push_back(ind);
            visited[de] = 1;
        }
    }
    cout << "Max Number of Tasks: " << count << endl;
    cout << "Selected Task Numbers: ";
    cout << selected[0];
```

```
    int l = selected.size();
    for (int i = 1; i < l; ++i)
    {
        cout << ", " << selected[i];
    }
    return 0;
}
```

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, 6, 3
```

10.3 Question 3

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. Time Complexity: $O(n \log n)$ Space Complexity: $O(1)$

Program

```
#include <bits/stdc++.h>

using namespace std;

int binary_search(vector<int> &a, int key, bool find_first_occurence)
{
    int l = a.size();
    int b = 0, e = l - 1;
    while (b <= e)
    {
        int mid = (b + e) / 2;
        if (a[mid] == key)
        {
            if (find_first_occurence)
            {
                if (mid == 0 || a[mid - 1] < a[mid])
                    return mid;
                else
                    e = mid - 1;
            }
            else
            {
                if (mid == l - 1 || a[mid + 1] > a[mid])
                    return mid;
                else
                    b = mid + 1;
            }
        }
        else if (a[mid] > key)
            e = mid - 1;
        else
            b = mid + 1;
    }
    return -1;
}

int main()
{

    int n;
    cin >> n;
    vector<int> a;
    for (int i = 0; i < n; ++i)
    {
        int x;
        cin >> x;
        a.push_back(x);
    }
    sort(a.begin(), a.end());
    int c = 0, i = 0;
    while (i < n)
```

```
    {
        int fo = binary_search(a, a[i], true), lo = binary_search(a, a[i],
    false);
        int nc = lo - fo + 1;
        if (nc > c)
            c = nc;
        i = lo + 1;
    }
    cout << (n * .5 < c ? "yes" : "no") << endl;
    if (n & 1)
        cout << a[n / 2] << endl;
    else
        cout << (a[n / 2] + a[n / 2 + 1]) / 2.0;
    return 0;
}
```

Input

```
9
4 4 2 3 2 2 3 2 2
```

Output

```
yes
2
```

11 Week 11

11.1 Question 1

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. Time Complexity: $O(n^2)$ Space Complexity: $O(n)$

Program

```
#include <bits/stdc++.h>
using namespace std;

int dp[1003][1003];
int solve(vector<int> &arr, int i, int j) {
    if (i >= j) return 0;
    if (dp[i][j] != -1) return dp[i][j];
    dp[i][j] = INT_MAX;
    for (int k = i; k < j; k++) {
        dp[i][j] = min(dp[i][j],
                       solve(arr, i, k) + solve(arr, k + 1, j) +
                           arr[i - 1] * arr[k] * arr[j]);
    }
    return dp[i][j];
}

int main() {
    int n;
    cin >> n;
    vector<int> arr(n + 1);
    for (int i = 1; i <= n; i++) {
        cin >> arr[i - 1];
        cin >> arr[i];
    }
    memset(dp, -1, sizeof(dp));
    cout << solve(arr, 1, n) << '\n';
    return 0;
}
```

Input

```
3
10 30
30 5
5 60
```

Output

```
4500
```

11.2 Question 2

Given a set of available types of coins. Let suppose you have infinite supply of each type of coin. For a given value N, you have to Design an algorithm and implement it using a program to find number of ways in which these coins can be added to make sum value equals to N. Time Complexity: $O(n * sm)$ Space Complexity: $O(n * sm)$

Program

```
#include <bits/stdc++.h>
using namespace std;

int memo(int i, int current, vector<vector<int>> &dp, vector<int> &coin,
    int n) {
     if (i == n) {
         return (current == 0);
     }
     if (current < 0)
         return 0;
     if (current == 0)
         return 1;
     if (dp[i][current] != -1)
         return dp[i][current];
     return dp[i][current] = memo(i + 1, current, dp, coin, n) +
                             memo(i, current - coin[i], dp, coin, n);
}
int main() {
     int n;
     cin >> n;
     int target;
     vector<int> coins(n);
     for (int i = 0; i < n; i++) {
         cin >> coins[i];
     }
     cin >> target;
     vector<vector<int>> dp(n, vector<int>(target + 1, -1));
     cout << memo(0, target, dp, coins, n);
}
```

Input

```
4
2 5 6 3
10
```

Output

```
5
```

11.3 Question 3

Given a set of elements, you have to partition the set into two subsets such that the sum of elements in both subsets is same. Design an algorithm and implement it using a program to solve this problem. Time Complexity: $O(n^2)$ Space Complexity: $O(n^2)$

Program

```
#include <bits/stdc++.h>
using namespace std;
int memo(int i, int current, vector<vector<int>> &dp, vector<int> arr, int
    n) {
    if (current == 0)
        return 1;
    if (current < 0)
        return 0;
    if (i == n)
        return 0;
    if (dp[i][current] != -1)
        return dp[i][current];
    return dp[i][current] = memo(i + 1, current - arr[i], dp, arr, n) ||
                            memo(i + 1, current, dp, arr, n);
}
int main() {
    int n;
    cin >> n;
    vector<int> arr(n);
    int sum = 0;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
        sum += arr[i];
    }
    if (sum % 2 == 1) {
        cout << "No" << endl;
    } else {
        vector<vector<int>> dp(n + 1, vector<int>(sum / 2 + 1, -1));
        if (memo(0, sum / 2, dp, arr, n)) {
            cout << "Yes" << endl;
        } else {
            cout << "No" << endl;
        }
    }
}
```

Input

```
7
1 5 4 11 5 14 10
```

Output

```
Yes
```