**CONTENT**

|  |  |  |
| --- | --- | --- |
| **DATE** | **WEEK WISE QUESTIONS** | **Page Number** |
|  | **WEEK 1** |  |
| 4/3/22 | Question 1 | 1 |
| 4/3/22 | Algorithm 1 | 2 |
| 4/3/22 | Question 2 | 3 |
| 4/3/22 | Algorithm 2 | 4 |
| 4/3/22 | Question 3 | 5-6 |
| 4/3/22 | Algorithm 3 | 7 |
|  | **WEEK 2** |  |
| 11/3/22 | Question 1 | 8-9 |
| 11/3/22 | Algorithm 1 | 10 |
| 11/3/22 | Question 2 | 11-12 |
| 11/3/22 | Algorithm 2 | 13 |
| 11/3/22 | Question 3 | 14-15 |
| 11/3/22 | Algorithm 3 | 16 |
|  | **WEEK 3** |  |
| 25/3/22 | Question 1 | 17-18 |
| 25/3/22 | Algorithm 1 | 19 |
| 25/3/22 | Question 2 | 20-21 |
| 25/3/22 | Algorithm 2 | 22 |
| 25/3/22 | Question 3 | 23-24 |
| 25/3/22 | Algorithm 3 | 25 |
|  | **WEEK 4** |  |
| 1/4/22 | Question 1 | 26-27 |
| 1/4/22 | Algorithm 1 | 28 |
| 1/4/22 | Question 2 | 29-30 |
| 1/4/22 | Algorithm 2 | 31 |
| 1/4/22 | Question 3 | 32-33 |
| 1/4/22 | Algorithm 3 | 34 |
|  | **WEEK 5** |  |
| 8/4/22 | Question 1 | 35-36 |
| 8/4/22 | Algorithm 1 | 37 |
| 8/4/22 | Question 2 | 38-39 |
| 8/4/22 | Algorithm 2 | 40 |
| 8/4/22 | Question 3 | 41-42 |
| 8/4/22 | Algorithm 3 | 43 |
|  | **WEEK 6** |  |
| 2/5/22 | Question 1 | 44-45 |
| 2/5/22 | Algorithm 1 | 46 |
| 2/5/22 | Question 2 | 47-48 |
| 2/5/22 | Algorithm 2 | 49 |
| 2/5/22 | Question 3 | 50-51 |
| 2/5/22 | Algorithm 3 | 52 |
|  | **WEEK 7** |  |
| 13/5/22 | Question 1 | 53-54 |
| 13/5/22 | Algorithm 1 | 55 |
| 13/5/22 | Question 2 | 56-57 |
| 13/5/22 | Algorithm 2 | 58 |
| 13/5/22 | Question 3 | 59 |
| 13/5/22 | Algorithm 3 | 60 |
|  | **WEEK 8** |  |
| 18/5/22 | Question 1 | 61-62 |
| 18/5/22 | Algorithm 1 | 63 |
| 18/5/22 | Question 2 | 64-65 |
| 18/5/22 | Algorithm 2 | 66 |
| 18/5/22 | Question 3 | 67-68 |
| 18/5/22 | Algorithm 3 | 69 |
|  | **WEEK 9** |  |
| 19/5/22 | Question 1 | 70-71 |
| 19/5/22 | Algorithm 1 | 72 |
| 19/5/22 | Question 2 | 73-74 |
| 19/5/22 | Algorithm 2 | 75 |
| 19/5/22 | Question 3 | 76 |
| 19/5/22 | Algorithm 3 | 77 |
|  | **WEEK 10** |  |
| 28/5/22 | Question 1 | 78 |
| 28/5/22 | Algorithm 1 | 79 |
| 28/5/22 | Question 2 | 80 |
| 28/5/22 | Algorithm 2 | 81 |
| 28/5/22 | Question 3 | 82 |
| 28/5/22 | Algorithm 3 | 83 |
|  | **WEEK 11** |  |
| 28/5/22 | Question 1 | 84 |
| 28/5/22 | Algorithm 1 | 85 |
| 28/5/22 | Question 2 | 86 |
| 28/5/22 | Algorithm 2 | 87 |
| 28/5/22 | Question 3 | 88 |
| 28/5/22 | Algorithm 3 | 89 |

**Week 1**

**Q1) 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)**

**#include** <bits/stdc++.h>

using **namespace** std;

**int** **main**()

**int** t;

    cin**>>**t;

**while**(t--) {

**int** n, key;

        cin**>>**n;

**int** arr[n];

**for**(**int** i = 0; i < n; i++) {

            cin**>>**arr[i];

        }

        cin**>>**key;

**int** j;

**for**(j = 0; j < n; j++) {

**if**(arr[j] == key) {

                cout**<<**"Present "**<<**j+1**<<**'\n';

**break**;

            }

        }

**if**(j == n) {

            cout**<<**"Not Present "**<<**j**<<**"\n";

        }

    }

}

**OUTPUT**

**Text

Description automatically generated**

**Q2) 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).**

#include <bits/stdc++.h>

using namespace std;

int main() {

int t;

cin>>t;

while(t--) {

int n, key;

cin>>n;

int arr[n];

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

cin>>arr[i];

}

cin>>key;

int l = 0, h = n - 1;

int i = 0, flag = 0;

while(l <= h) {

int mid = (l + h) / 2;

i++;

if(arr[mid] == key)

{

flag = 1;

cout<<"Present "<<i<<endl;

break;

}

else if(arr[mid] > key) h = mid - 1;

else l = mid + 1;

}

if(!flag) cout<<"Present "<<n/2<<endl;

}

return 0;

}

**OUTPUT**

Text

Description automatically generated

**Q3) 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[2k] and so on. Once the interval (arr[2k] < key < arr[ 2k+1] ) is found, perform a linear search operation from the index 2k 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.**

#include <bits/stdc++.h>

using namespace std;

void jump(int arr[], int n, int key) {

int start = 0, comp = 0, flag = 0;

int end = sqrt(n);

while(arr[end] <= key && end < n) {

comp++;

start = end;

end += sqrt(n);

if(end > n-1) end = n;

}

for(int i=start;i<end;i++) {

if(arr[i] == key) {

flag = true;

break;

}

}

if(flag) cout<<"Present "<<comp<<endl;

else cout<<"Not present"<<endl;

}

int main()

{

int n;

cin>>n;

while(n--)

{

int size;

cin>>size;

int arr[size];

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

int key;

cin>>key;

jump(arr, size, key);

}

return 0;

}

**OUTPUT**

Text

Description automatically generated with medium confidence

**Week 2**

**Q1) 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”.**

#include<iostream>

using namespace std;

int main() {

int n,x;

int count=0;

cout<<"enter the size of array:"<<endl;

cin>>n;

int A[n];

cout<<"\n Enter the element of array:"<<endl;

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

{

cin>>A[i];

}

cout<<"\n Enter the element you want to search"<<endl;

cin>>x;

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

{

if(A[i]==x)

{

count++;

}

}

cout<<"The frequency of key element is :"<<count;

return 0;

}

**OUTPUT**

Text

Description automatically generated

**Q2) 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”.**

#include <bits/stdc++.h>

using namespace std;

int main() {

int t;

cin>>t;

while(t--) {

int n;

cin>>n;

int arr[n];

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

cin>>arr[i];

vector<int> v;

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

int k = n - 1;

int j = i + 1;

while(j < n && k > j) {

if(arr[i] + arr[j] == arr[k]) {

v.push\_back(i+1);

v.push\_back(j+1);

v.push\_back(k+1);

break;

}

else if(arr[i] + arr[j] > arr[k]) {

j++;

k = n - 1;

}

else k--; }

}

if(v.empty())

cout<<"No sequence found"<<endl;

else {

for(auto it: v) {

cout<<it<<" ";

}

cout<<endl; }

}

}

**OUTPUT**

Graphical user interface

Description automatically generated with medium confidence

**Q3) 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.**

#include <bits/stdc++.h>

using namespace std;

int main() {

int t;

cin>>t;

while(t--) {

int n;

cin>>n;

int arr[n];

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

cin>>arr[i];

int key;

cin>>key;

sort(arr, arr + n);

int c = 0, l, h;

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

l = i;

h = n - 1;

while(l < h) {

if(arr[h] - arr[l] == key) {

c++;

h--;

l++;

}

else if(arr[h] - arr[l] > key) h--;

else l++;

}

}

cout<<c<<endl;

}

}

**OUTPUT**

Text

Description automatically generated

**Week 3**

**Q1) 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.**

#include<bits/stdc++.h>

using namespace std;

void insertionSort(int a[], int n){

int ctr=0,flag=0;

for(int i=1;i<n;i++){

ctr++;

int j;

j=i-1;

int x=a[i];

while (j>-1 && a[j]>x){

ctr++;

a[j+1]=a[j];

flag++;

j--;

}

a[j+1]=x;

}

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

cout<<a[i]<<" ";

}

cout<<endl;

cout<<" Shifts = "<<ctr<<endl;

cout<<"Comparison = "<<flag<<endl;

}

int main(){

int n;

cout<<"enter the no. of test cases"<<endl;

cin>>n;

while(n--)

{

int c;

cout<<"enter the no. of elements in an array"<<endl;

cin>>c;

int a[c];

for(int i=0;i<c;i++)

{

cin>>a[i];

}

insertionSort(a,c);

}

}

**OUTPUT**

**Text

Description automatically generated**

**Q2) 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.**

#include <iostream>

using namespace std;

void Sel\_Sort(int[], int);

int main()

{

int T;

cin >> T;

for (int i = 0; i < T; i++)

{

int n;

cin >> n;

int A[1000];

for (int j = 0; j < n; j++)

{

cin >> A[j];

}

Sel\_Sort(A, n);

}

}

void Sel\_Sort(int A[], int n)

{

int comp = 0, swaps = 0;

int min, temp = 0;

for (int i = 0; i < n - 1; i++)

{

min = i;

for (int j = i + 1; j < n; j++)

{

comp++;

if (A[min] > A[j])

{

min = j;

}

}

swaps++;

swap(A[min], A[i]);

}

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

{

cout << A[i] << " ";

}

cout << "\ncomparisons = " << comp << endl

<< "swaps = " << swaps << endl;

}

**OUTPUT**

**Text

Description automatically generated**

**Q3) 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’.**

#include<bits/stdc++.h>

using namespace std;

void findDuplicate(int a[], int x)

{

for(int i=0;i<x;i++){

for(int j=i+1;j<x;j++){

if(a[i]==a[j]){

cout<<"YES"<<endl;

}

break;

}

}

cout<<"NO"<<endl;

}

int main(){

int n;

cin>>n;

while(n--){

int x;

cin>>x;

int a[x];

for(int i=0;i<x;i++){

cin>>a[i];

}

findDuplicate(a ,x);

}

}

**OUTPUT**

Text

Description automatically generated

**Week 4**

**Q1) 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.**

#include <bits/stdc++.h>

using namespace std;

int c = 0;

void mergeArray(int arr[], int l, int mid, int h) {

int n1 = mid - l + 1;

int n2 = h - mid;

int a[n1], b[n2];

for(int i = 0; i < n1; i++) {

a[i] = arr[l + i];

}

for(int i = 0; i < n2; i++) {

b[i] = arr[mid + 1 + i];

}

int i = 0, j = 0, k = l;

while(i < n1 && j < n2) {

c++;

if(a[i] <= b[j]) {

arr[k] = a[i];

i++;

}

else {

arr[k] = b[j];

j++;

}

k++;

}

while(i < n1) {

arr[k] = a[i];

i++; k++;

}

while(j < n2) {

arr[k] = b[j];

j++; k++;

}

}

void mergeSort(int arr[], int l, int h) {

if(l < h) {

int mid = l + (h - l) / 2;

mergeSort(arr, l, mid);

mergeSort(arr, mid + 1, h);

mergeArray(arr, l, mid, h);

}

}

void display(int arr[], int n) {

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

cout<<arr[i]<<" ";

}

cout<<endl;

}

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];

}

mergeSort(arr, 0, n - 1);

display(arr, n);

cout<<"comparisons = "<<c<<endl;

delete []arr;

}

}

**OUTPUT**

Text

Description automatically generated

**Q2) 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.**

#include <bits/stdc++.h>

using namespace std;

int c = 0, s = 0;

int partition(int arr[], int l, int h) {

int x = (rand() % (l - h)) + l;

if(h != x) {

s++;

swap(arr[x], arr[h]);

}

int pivot = arr[h];

int i = l - 1;

for(int j = l; j <= h - 1; j++) {

if(arr[j] <= pivot) {

i++;

s++;

swap(arr[i], arr[j]);

}

}

s++;

swap(arr[i + 1], arr[h]);

return i + 1;

}

void quickSort(int arr[], int l, int h) {

if(l < h) {

int pivot = partition(arr, l, h);

quickSort(arr, l, pivot - 1);

quickSort(arr, pivot + 1, h);

}

}

void display(int arr[], int n) {

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

cout<<arr[i]<<" ";

}

cout<<endl;

}

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];

}

quickSort(arr, 0, n - 1);

display(arr, n);

cout<<"swaps = "<<s<<endl;

}

}

**OUTPUT**

Text

Description automatically generated

**Q3) 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”.**

#include<bits/stdc++.h>

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 k;

cin>>k;

priority\_queue<int> pq;

for(int i = 0; i < k; i++) {

pq.push(arr[i]);

}

for(int i = k; i < n; i++) {

pq.push(arr[i]);

if(pq.size() > k) pq.pop();

}

if(pq.empty()) cout<<"Not present"<<endl;

else cout<<pq.top()<<endl;

delete []arr;

}

}

**OUTPUT**

**Text

Description automatically generated**

**Week 5**

**Q1) 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”.**

#include<iostream>

using namespace std;

char findmax(char a[], int n)

{

char m=a[0];

for(int i=1;i<n;i++)

{

if(a[i]>m)

{

m=a[i];

}

}

return m;

}

void count\_sort(char a[],int n)

{

int i,\*c,j,dup,temp;

char max=findmax(a,n);

int m=int(max);

c = new int[m+1];

for(i=0;i<m+1;i++)

{

c[i]=0;

}

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

{

c[int(a[i])]++;

}

dup=c[i];

for(int i=0;i<m+1;i++)

{

if(c[i]>dup)

{

dup=c[i];

temp=i;

}

}

char ch=(char)temp;

if(dup>1)

{

cout<<ch<<" :"<<dup<<endl;

}

else

{

cout<<"No duplicates found"<<endl;

}

}

int main()

{

int t,n;

cout<<"enter number of test case"<<endl;

cin>>t;

while(t>0)

{

cout<<"enter number of elements in array"<<endl;

cin>>n;

char a[n];

cout<<"enter "<<n<<" number of elements"<<endl;

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

{

cin>>a[i];

}

count\_sort(a,n);

t--;

}

}

**OUTPUT**

Text

Description automatically generated

**Q2) 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”.**

#include <bits/stdc++.h>

using namespace std;

int main()

{

int t;

cin >>t;

while(t--)

{

int n,sum,flag=0;

cin >> n;

int ar[n];

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

cin >> ar[i];

cin >> sum;

sort(ar,ar+n);

int a=0,b=n-1;

while(a<b)

{

if(ar[a]+ar[b]>sum)

--b;

else if(ar[a]+ar[b]<sum)

++a;

else

{

cout << ar[a] << "&" << ar[b] << ", ";++a;--b;++flag;

}

}

if(!flag)

cout << "NO Such Pair Exist" << "\n";

}

}

**OUTPUT**

Text

Description automatically generated

**Q3) 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.**

#include <bits/stdc++.h>

using namespace std;

int main()

{

int n,m;

cin >> n ;

int ar[n];

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

cin >> ar[i];

cin >> m;

int ar2[m];

for (int i = 0; i < m; ++i)

cin >> ar2[i];

int a=0,b=0;

while(a<n && b<m)

{

if(ar[a] < ar2[b])

++a;

else if(ar[a] > ar2[b])

++b;

else

{

cout << ar[a] << " ";++a;++b;}

}

cout << "\n";

}

**OUTPUT**

Text

Description automatically generated with medium confidence

**Week 6**

**Q1) 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'.**

#include <bits/stdc++.h>

using namespace std;

void dfs(vector<int> arr[], int source, int V, bool \*visited)

{

    visited[source] = true;

    for (int i = 0; i < V; i++)

    {

        if (arr[source][i] != 0 && !visited[i])

        {

            dfs(arr, i, V, visited);

        }

    }

}

bool checkPath(vector<int> arr[], int V, int source, int destination)

{

    bool visited[V];

    for (int i = 0; i < V; i++)

        visited[i] = false;

    dfs(arr, source, V, visited);

    return visited[destination];

}

int main()

{

    int n;

    cin >> n;

    vector<int> arr[n];

    int temp;

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            cin >> temp;

            arr[i].push\_back(temp);

        }

    }

    int source, destination;

    cin >> source >> destination;

    if (checkPath(arr, n, source - 1, destination - 1))

    {

        cout << "Yes Path Exists.\n";

    }

    else

    {

        cout << "No Such Path Exists.\n";

    }

    return 0;

}

**OUTPUT**

**Calendar

Description automatically generated**

**Q2) 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'.**

#include <bits/stdc++.h>

using namespace std;

bool isBipartiteUtil(vector<int> G[], int src, int colorArr[], int V)

{

    colorArr[src] = 1;

    queue<int> q;

    q.push(src);

    while (!q.empty())

    {

        int u = q.front();

        q.pop();

        if (G[u][u] == 1)

            return false;

        for (int v = 0; v < V; ++v)

        {

            if (G[u][v] != 0 && colorArr[v] == -1)

            {

                colorArr[v] = 1 - colorArr[u];

                q.push(v);

            }

            else if (G[u][v] != 0 && colorArr[v] == colorArr[u])

                return false;

        }

    }

    return true;

}

bool isBipartite(vector<int> G[], int V)

{

    int colorArr[V];

    for (int i = 0; i < V; ++i)

        colorArr[i] = -1;

    for (int i = 0; i < V; i++)

        if (colorArr[i] == -1)

            if (isBipartiteUtil(G, i, colorArr, V) == false)

                return false;

    return true;

}

int main()

{

    int n;

    cin >> n;

    vector<int> G[n];

    int temp;

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            cin >> temp;

            G[i].push\_back(temp);

        }

    }

    if (isBipartite(G, n))

    {

        cout << "Yes Bipartite\n";

    }

    else

    {

        cout << "Not Bipartite\n";

    }

    return 0;

}

**OUTPUT**

**Calendar

Description automatically generated**

**Q3) 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'.**

#include <bits/stdc++.h>

using namespace std;

bool CheckCycle(int node, vector<int> adj[], int vis[], int dfsvis[])

{

    vis[node] = 1;

    dfsvis[node] = 1;

    for (auto it : adj[node])

    {

        if (!vis[it])

        {

            if (CheckCycle(it, adj, vis, dfsvis))

                return true;

        }

        else if (dfsvis[it])

            return true;

    }

    dfsvis[node] = 0;

    return false;

}

bool isCycle(vector<int> adj[], int N)

{

    int vis[N + 1], dfsVis[N + 1];

    memset(vis, 0, sizeof(vis));

    memset(dfsVis, 0, sizeof(dfsVis));

    for (int i = 1; i <= N; i++)

    {

        if (!vis[i])

        {

            if (CheckCycle(i, adj, vis, dfsVis))

                return true;

        }

    }

    return false;

}

int main()

{

    int n, m;

    cin >> n >> m;

    vector<int> adj[n + 1];

    for (int i = 1; i <= m; i++)

    {

        int u, v;

        cin >> u >> v;

        adj[u].push\_back(v);

    }

    if (isCycle(adj, n))

        cout << "Cycle Exists" << endl;

    else

        cout << "No Cycle Exists" << endl;

    return 0;

}

**OUTPUT**

**Graphical user interface, text, application

Description automatically generated**

**Week 7**

**Q1) 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 into location of**

**his house and his friend’s 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 friend’s location to Akshay’s house)**

**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 weight.**

#include<iostream>

#include<bits/stdc++.h>

using namespace std;

int minDisIndex(int \*dis,bool \*vis,int v)

{

int i;

int minDis=INT\_MAX;

int minIndex=-1;

for(i=0;i<v;i++)

{

if(vis[i]==false && dis[i]<=minDis)

{

minDis=dis[i];

minIndex=i;

}

}

return minIndex;

}

void dijkstra(vector<vector<int>> mat,int v,int s)

{

int dis[v];

bool vis[v];

int parent[v];

int i,j;

for(i=0;i<v;i++)

{

dis[i]=INT\_MAX;

vis[i]=false;

parent[i]=-1;

}

dis[s]=0;

parent[s]=s;

for(i=0;i<v;i++)

{

int m=minDisIndex(dis,vis,v);

vis[m]=true;

for(j=0;j<v;j++)

{

if(dis[m]!=INT\_MAX && !vis[j] && mat[m][j])

{

if(dis[j]>dis[m]+mat[m][j])

{

dis[j]=dis[m]+mat[m][j];

parent[j]=m;

}

}

}

}

for(i=0;i<v;i++) {

if(i==s) {

cout<<i+1<<" : "<<dis[i]<<endl;

continue;

}

cout<<i+1;

j=i;

while(parent[j]!=s) {

cout<<" "<<parent[j]+1;

j=parent[j];

}

cout<<" "<<s+1<<" : "<<dis[i]<<endl;

}

}

int main()

{

int i,j;

int v;

cin>>v;

vector<vector<int>> mat(v,vector<int> (v));

for(i=0;i<v;i++)

for(j=0;j<v;j++)

cin>>mat[i][j];

int s;

cin>>s;

dijkstra(mat,v,s-1);

return 0;

}

**OUTPUT**

**A screen shot of numbers

Description automatically generated with low confidence**

**Q2) 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 weight.**

#include <bits/stdc++.h>

using namespace std;

void calulate(vector<int> &pa, int i)

{

cout << i + 1 << " ";

if (pa[i] >= 0)

calulate(pa, pa[i]);

}

void find\_path(int \*\*graph, int m, int sour)

{

vector<int> dis(m, INT\_MAX), pa(m, -1);

dis[sour] = 0;

for (int ki = 0; ki < m - 1; ki++)

{

for (int i = 0; i < m; i++)

{

for (int j = 0; j < m; j++)

{

if (graph[i][j] != 0)

{

if (dis[j] > dis[i] + graph[i][j])

{

dis[j] = dis[i] + graph[i][j];

pa[j] = i;

}

}

}

}

}

for (int i = 0; i < m; i++)

{

calulate(pa, i);

cout << ": " << dis[i] << endl;

}

}

int main()

{

int m, source, ed;

cin >> m;

int \*\*graph = (int \*\*)malloc(m \* sizeof(int \*));

for (int i = 0; i < m; i++)

graph[i] = (int \*)malloc(m \* sizeof(int));

for (int i = 0; i < m; i++) {

for (int j = 0; j < m; j++) {

cin >> graph[i][j];

}

}

cin >> source;

find\_path(graph, m, source - 1);

}

**OUTPUT**

A screen shot of a calculator

Description automatically generated with low confidence

**Q3) 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”.**

#include <bits/stdc++.h>

using namespace std;

int shortest\_weigt(int \*\*graph, int ver, int u, int v, int k)

{

if (k <= 0)

return INT\_MAX;

if (k == 0 && u == v)

return 0;

if (k == 1 && graph[u][v] != INT\_MAX)

return graph[u][v];

int res = INT\_MAX;

for (int i = 0; i < ver; i++) {

if (graph[u][i] != 0 && u != i && v != i) {

int recu = shortest\_weigt(graph, ver, i, v, k - 1);

if (recu != INT\_MAX)

res = min(res, graph[u][i] + recu);

}

}

return res;

}

int main()

{

int ver, u, v, k, ans;

cin >> ver;

int \*\*graph = (int \*\*)malloc(ver \* sizeof(int \*));

for (int i = 0; i < ver; i++)

graph[i] = (int \*)malloc(sizeof(int) \* ver);

for (int i = 0; i < ver; i++)

for (int j = 0; j < ver; j++)

cin >> graph[i][j];

cin >> u >> v >> k;

ans = shortest\_weigt(graph, ver, u - 1, v - 1, k);

cout << "Weight of shortest path from (" << u

<< "," << v << ") with " << k << " edges :" << ans;

}

**OUTPUT**

**Text

Description automatically generated**

**Week 8**

**Q1) 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)**

**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**

#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;

}

**OUTPUT**

Text

Description automatically generated

**Q2) 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 Kruskal 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**

#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());

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;

}

**OUTPUT**

Text

Description automatically generated

**Q3) 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.**

#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;

}

**OUTPUT**

**Text

Description automatically generated with low confidence**

**Week 9**

**Q1) 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.**

#include <bits/stdc++.h>

using namespace std;

int main()

{

int n, i, j, k, w;

cin >> n;

int graph[n][n];

string temp;

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

{

for (j = 0; j < n; j++)

{

cin >> temp;

if (temp != "INF")

{

graph[i][j] = stoi(temp);

} else {

graph[i][j] = 1e8;

}

}

}

for (k = 0; k < n; k++)

{

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

{

for (j = 0; j < n; j++)

{

if (graph[i][k] + graph[k][j] < graph[i][j])

{

graph[i][j] = graph[i][k] + graph[k][j];

}

}

}

}

cout << "The shortest path matrix: " << endl;

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

{

for (j = 0; j < n; j++)

{

if(graph[i][j] >= 1e8) cout << "INF";

else cout << graph[i][j];

cout << " ";

}

cout << endl;

}

return 0;

}

**OUTPUT**

**Text

Description automatically generated**

**Q2) Given a knapsack of maximum capacity w. N items are provided each having its own value and weight. You have to design and 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 xi of items i, where 0<xi<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 will contain N space-separated array containing values of all N items.**

**Last line of 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.**

#include <bits/stdc++.h>

using namespace std;

int main()

{

int n;

cin >> n;

vector<double> items(n);

vector<double> val(n);

vector<vector<double>> job;

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

{

cin >> items[i];

}

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

{

cin >> val[i];

job.push\_back({val[i] / items[i], items[i], (double)(i + 1)});

}

double k;

cin >> k;

sort(job.rbegin(), job.rend());

vector<pair<double, double>> ls;

float profit = 0;

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

{

if (job[i][1] >= k)

{

profit += k \* job[i][0];

ls.push\_back(make\_pair(k, job[i][2]));

break;

}

else

{

profit += job[i][1] \* job[i][0];

}

ls.push\_back(make\_pair(job[i][1], job[i][2]));

k = k - job[i][1];

}

cout << "Maximum Value : " << profit << endl;

cout << "Item - Weight" << endl;

for (auto it : ls)

cout << it.second << " - " << it.first << endl;

return 0;

}

**OUTPUT**

**Text

Description automatically generated**

**Q3) 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 minimum computation cost required to merge all elements of array.**

#include <bits/stdc++.h>

using namespace std;

int main()

{

int n;

cin >> n;

vector<int> a(n);

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

{

cin >> a[i];

}

priority\_queue<int, vector<int>, greater<int>> minheap;

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

minheap.push(a[i]);

}

int ans = 0;

while (minheap.size() >1)

{

int e1 = minheap.top();

minheap.pop();

int e2 = minheap.top();

minheap.pop();

ans += e1 + e2;

minheap.push(e1 + e2);

}

cout << ans;

return 0;

}

**OUTPUT**

**Graphical user interface, text

Description automatically generated**

**Week 10**

**Q1) 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.**

#include<bits/stdc++.h>

using namespace std;

int main() {

int n;

cin>>n;

int i,s[n],f[n];

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

cin>>s[i];

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

cin>>f[i];

vector<vector<int>> a;

vector<int> act;

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

a.push\_back({f[i],s[i],i+1});

sort(a.begin(),a.end());

int e=INT\_MIN,c=0;

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

{

if(a[i][1]>=e)

{

e=a[i][0];

c++;

act.push\_back(a[i][2]);

}

}

cout<<"No. of non-conflicting activities : "<<c<<endl;

cout<<"List of selected activities : ";

for(i=0;i<act.size();i++)

cout<<act[i]<<",";

return 0;

}

**OUTPUT**

**Text

Description automatically generated**

**Q2) 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.**

#include<bits/stdc++.h>

using namespace std;

int main()

{

int n;

cin>>n;

int i,t[n],f[n];

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

cin>>t[i];

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

cin>>f[i];

vector<vector<int>> a;

vector<int> act;

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

a.push\_back({f[i],f[i]-t[i],i+1});

sort(a.begin(),a.end());

int e=INT\_MIN,c=0;

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

{

if(a[i][1]>=e)

{

e=a[i][0];

c++;

act.push\_back(a[i][2]);

}

}

sort(act.begin(),act.end());

cout<<"Max number of tasks : "<<c<<endl;

cout<<"Selected task Numbers : ";

for(i=0;i<act.size();i++)

cout<<act[i]<<",";

return 0;

}

**OUTPUT**

**Text

Description automatically generated**

**Q3) 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.**

#include<bits/stdc++.h>

using namespace std;

int main()

{

int n;

cin>>n;

int i,a[n],c,j;

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

cin>>a[i];

bool f=0;

sort(a,a+n);

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

{

c=1;

j=i+1;

while(j<n && a[j++]==a[i])

c++;

if(c>n/2)

{

cout<<"yes\n";

f=1;

break;

}

i=j-1;

}

if(f==0)

cout<<"no\n";

if(n%2!=0)

cout<<a[n/2];

else

cout<<((float)a[n/2]+a[n/2-1])/2;

return 0;

}

**OUTPUT**

**Text

Description automatically generated**

**Week 11**

**Q1) 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.**

#include<bits/stdc++.h>

using namespace std;

long matChainOrder(int \*p,int n) {

int m[n][n];

int i,j,k,l,q;

for(i=1;i<n;i++)

m[i][i]=0;

for(l=2;l<n;l++)

{

for(i=1;i<n-l+1;i++)

{

j=i+l-1;

m[i][j]=INT\_MAX;

for(k=i;k<=j-1;k++)

{

q=m[i][k]+m[k+1][j]+p[i-1]\*p[k]\*p[j];

if(q<m[i][j])

m[i][j]=q;

}

}

}

return m[1][n-1];

}

int main()

{

int n;

cin>>n;

int p[n+1];

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

{

cin>>p[i]>>p[i+1];

}

cout<<matChainOrder(p,n+1);

return 0;

}

**OUTPUT**

Graphical user interface, application

Description automatically generated

**Q2) 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.**

**Input Format:**

**First line of input will take number of coins that are available.**

**Second line of input will take the value of each coin.**

**Third line of input will take the value N for which you need to find sum.**

**Output Format:**

**Output will be the number of ways.**

#include<bits/stdc++.h>

using namespace std;

int main()

{

int n,amt;

cin>>n;

int i,j,a[n];

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

cin>>a[i];

cin>>amt;

int ans[amt+1];

for(i=1;i<=amt;i++)

ans[i]=0;

ans[0]=1;

for(j=0;j<n;j++)

{

for(i=1;i<=amt;i++)

{

if(a[j]<=i)

ans[i]+=(ans[i-a[j]]);

}

}

cout<<ans[amt];

return 0;

}

**OUTPUT**

**Graphical user interface

Description automatically generated with low confidence**

**Q3) 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.**

**Input Format:**

**First line of input will take number of elements n present in the set.**

**Second line of input will take n space-separated elements of the set.**

**Output Format:**

**Output will be 'yes' if two such subsets found otherwise print 'no'.**

#include<bits/stdc++.h>

using namespace std;

int main() {

int n;

cin>>n;

int i,j,a[n];

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

cin>>a[i];

int sum=0;

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

sum+=a[i];

if(sum%2!=0)

{

cout<<"no";

return 0;

}

sum=sum/2;

bool s[n+1][sum+1];

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

{

for(j=0;j<=sum;j++)

{

if(j==0)

s[i][j]=1;

else if(i==0)

s[i][j]=0;

else

{

if(a[i-1]>j)

s[i][j]=s[i-1][j];

else

s[i][j]=(s[i-1][j] || s[i-1][j-a[i-1]]);

}

}

}

if(s[n][sum])

cout<<"yes";

else

cout<<"no";

return 0;

}

**OUTPUT**

Text

Description automatically generated