**PRACTICAL 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 total number

of comparisons for each input case. (Time Complexity = O(n), where n is the size of input)

**ALGORITHM:MTHMTHM**

**START**

Step 1: Read the value of ‘n’ from user.

Step 2: Declare and input the elements in the array.

Step 3: Ask user for target element.

Step 4: Call the linear search function to check if target element is present in the array or not.

1. Using a loop traverse the array, if an element equal to target is found return the index of target.
2. If the entire array is traversed and target is not found, return -1.

Step 5: If the function returns -1, print “Target is not present”

Step 6: Else, print the index where the element is present.

**STOP**

**SOURCE CODE:**

#include <iostream>

#include <vector>

using namespace std;

int linear(vector<int> arr,int target){

for(int i=0;i<arr.size();i++){

if(arr[i]==target)

return i;

}

return -1;

}

int main(){

int n,val,target,testcases;

cout << "Enter number test-cases: ";

cin >> testcases;

while(testcases>0){

cout << "Enter the Size of array: ";

cin >> n;

vector<int> arr;

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

cin >> val;

arr.push\_back(val);

}

cout << "Enter target to search: ";

cin >> target;

int found=linear(arr,target);

if(found==-1)

cout << "The element " << target << " Not present...";

else

cout << "The element " << target << " is present at index: " << found;

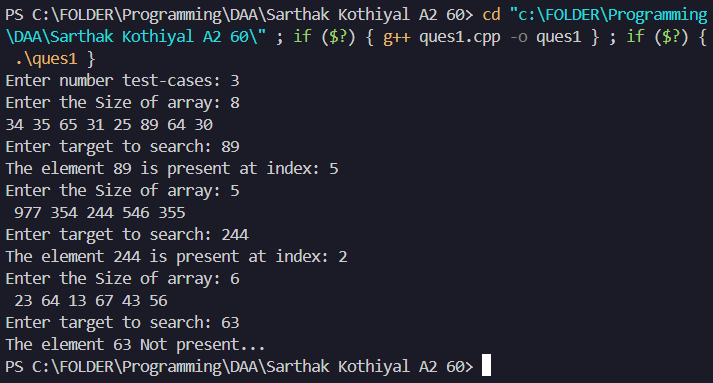
testcases--;

}

return 0;

}

**OUTPUT:**



**PRACTICAL 2**

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).

**ALGORITHM:**

**START**

Step 1: Read the value of ‘n’ from user, also set a counter variable for comparisons to 0.

Step 2: Declare and input the elements in the array.

Step 3: Ask user for target element.

Step 4: Call binary search function to check if the target is present in the array or not.

1. Set a low pointer to index 0 and high pointer to highest index of the array.
2. Use a loop until low<= high
3. Calculate mid element of the current search space: mid=low+(high-low)/2.
4. If arr[mid] == target, increment the counter and return index of target.
5. If arr[mid] > target, move high pointer to mid-1 and increment the counter.
6. If arr[mid] < target, move low pointer to mid+1 and increment the counter.

Step 5: If the loop terminates, it indicates that the target is not present in the array, return -1.

Step 6: If the function returns -1, print “Target in not present”.

Step 7: Else, print the index of the target element.

Step 8: Display the count to comparisons made.

**STOP**

**SOURCE CODE:**

#include <iostream>

#include <vector>

using namespace std;

int binary(vector<int> arr,int target,int &comp){

int low=0,high=arr.size()-1;

while(low<=high){

int mid=low+(high-low)/2;

if(arr[mid]==target){

comp++;

return mid;

}

else if(arr[mid]>target){

high=mid-1;

comp++;

}

else{

low=mid+1;

comp++;

}

}

return -1;

}

int main(){

int n,val,target,testcases;

cout << "Enter number of testcases: ";

cin >> testcases;

while(testcases>0){

int comp=0;

cout << "Enter the Size of array: ";

cin >> n;

vector<int> arr;

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

cin >> val;

arr.push\_back(val);

}

cout << "Enter target to search: ";

cin >> target;

int found=binary(arr,target,comp);

if(found==-1)

cout << "The element " << target << " Not present..." << endl;

else

cout << "The element " << target << " is present at index: " << found << endl;

cout << "Total number of Comparisions: " << comp << endl;

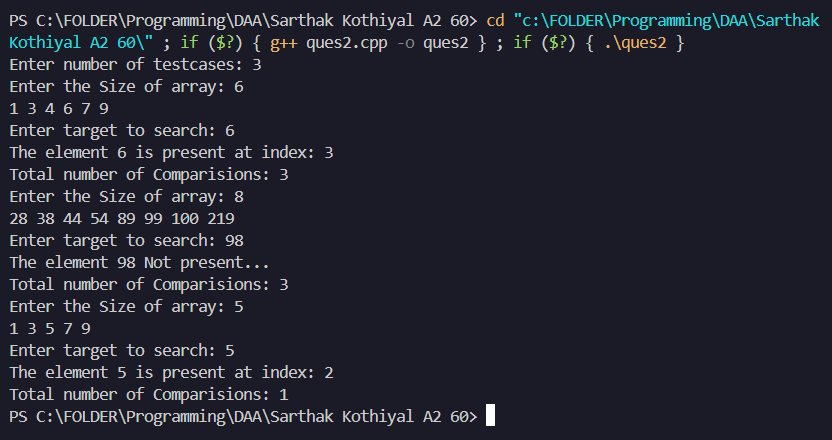
testcases--;

}

return 0;

}

**OUTPUT:**



**PRACTICAL 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[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)

**ALGORITHM:**

**START**

Step 1: Read the value of ‘n’ from user.

Step 2: Declare and input the elements in the array.

Step 3: Ask user for target element.

Step 4: Call jump search function to check if the target is present in the array or not.

1. Set up the window size = Root of (size of array).
2. Traverse the array using a loop and increment by window size at each iteration.
3. Exit the loop when pointer exceed the array size or arr[i] > target
4. Perform linear search on the last found window.
5. If arr[i]==target, return the index, else, return -1.

**STOP**

**SOURCE CODE:**

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

int linear(vector<int> arr,int a,int b,int target){

for(int i=a;i<=b;i++){

if(arr[i]==target){

return i;

}

else if(arr[i]>target){

return -1;

}

}

return -1;

}

int jump(vector<int> arr,int target){

int n=arr.size();

int window=sqrt(n);

int i=window;

while(i<n && arr[i]<target){

i+=window;

}

return linear(arr,i-window,min(i,n-1),target);

}

int main(){

int n,val,target,testcases;

cout << "Enter number of testcases: ";

cin >> testcases;

while(testcases>0){

vector<int> arr;

cout << "Enter size of array: ";

cin >> n;

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

cin >> val;

arr.push\_back(val);

}

cout << "Enter element to search: ";

cin >> target;

int found=jump(arr,target);

if(found==-1)

cout << "The Element " << target << " is Not present..." << endl;

else

cout << "The Element " << target << " is present at index: " << found << endl;

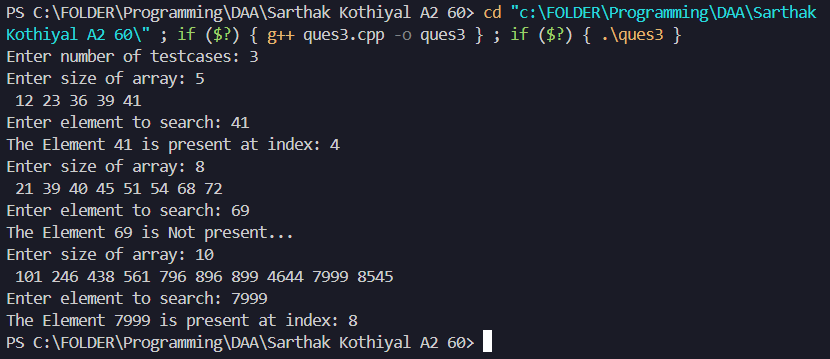
testcases--;

}

return 0;

}

**OUTPUT:**



**PRACTICAL 4**

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))

**ALGORITHM:**

**START**

Step 1: Read the value of ‘n’ from user.

Step 2: Declare and input the elements in the array.

Step 3: Ask user for target element.

Step 4: Find the first occurrence of the target element.

1. Set low and high pointer to 0 and arr.size() respectively also set index pointer, num to -1.
2. Perform binary search:
   * Find mid index, mid=low+(high-low)/2.
   * If arr[mid]<=target, set high pointer to mid-1, and if arr[mid]==target, num = mid.
   * Else, set low pointer to mid+1.
3. Return nums

Step 5: Similarly find the second occurrence of the target element.

1. Perform binary search:
   * Find mid index, mid=low+(high-low)/2.
   * If arr[mid]<=target, set low pointer to mid+1, and if arr[mid]==target, num = mid.
   * Else, set high pointer to mid-1.
2. Return nums.

Step 6: If first or second occurrence of the target element is -1, print “Element is not present”.

Step 7: Else, print the number of occurrences: difference of second and first occurrence.

**STOP**

**SOURCE CODE:**

#include <iostream>

#include <vector>

using namespace std;

int find\_lower(vector<int> nums,int target){

int low=0,high=nums.size()-1;

int num=-1;

while(low<=high){

int mid=low+(high-low)/2;

if(nums[mid]>=target){

if(nums[mid]==target)

num=mid;

high=mid-1;

}

else

low=mid+1;

}

return num;

}

int find\_higher(vector<int> nums,int target){

int low=0,high=nums.size()-1;

int num=-1;

while(low<=high){

int mid=low+(high-low)/2;

if(nums[mid]<=target){

if(nums[mid]==target)

num=mid;

low=mid+1;

}

else

high=mid-1;

}

return num;

}

int main(){

int n,val,target,testcases;

cout << "Enter number of testcases: ";

cin >> testcases;

while(testcases>0){

vector<int> arr;

cout << "Enter size of array: ";

cin >> n;

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

cin >> val;

arr.push\_back(val);

}

cout << "Enter element to search: ";

cin >> target;

int left=find\_lower(arr,target);

int right=find\_higher(arr,target);

if(left==-1 || right==-1)

cout << "The element " << target << " is Not present..." << endl;

else

cout << "The element " << target << " is present " << right-left+1 << " times..." << endl;

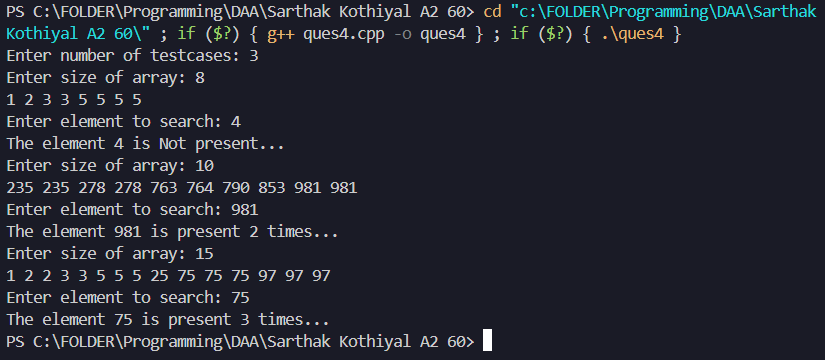
testcases--;

}

return 0;

}

**OUTPUT:**



**PRACTICAL 5**

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].

**ALGORITHM**

**START**

Step 1: Read the value of ‘n’ from user.

Step 2: Declare and input the elements in the array.

Step 3: Set up two pointers i and j and a flag to true.

Step 4: Traverse the array from k=n-1 till k>=2.

Step 5: Set sum=arr[k], i=0 and j=k-1.

Step 6: Traverse the array till i<j.

Step 7: If arr[i]+arr[j]==sum, flag=false, print the indices, and exit the program.

Step 8: If arr[i]+arr[j]>sum, increment i, else decrement j.

Step 9: If flag is true, no such sequence is present.

**STOP**

**SOURCE CODE:**

#include <iostream>

#include <vector>

using namespace std;

void print\_index(vector<int> arr){

int n=arr.size();

int i,j;

bool flag=true;

for(int k=n-1;k>=2;k--){

int sum=arr[k];

i=0;

j=k-1;

while(i<j){

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

flag=false;

cout << "The i, j, k indices are: ";

cout << i+1 << "," << j+1 << "," << k+1 << endl;

return;

}

else if(arr[i]+arr[j]>sum)

j--;

else

i++;

}

}

if(flag)

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

}

int main(){

int n,val,testcases;

cout << "Enter number of testcases: ";

cin >> testcases;

while(testcases>0){

cout << "Enter size of array: ";

cin >> n;

vector<int> arr;

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

cin >> val;

arr.push\_back(val);

}

print\_index(arr);

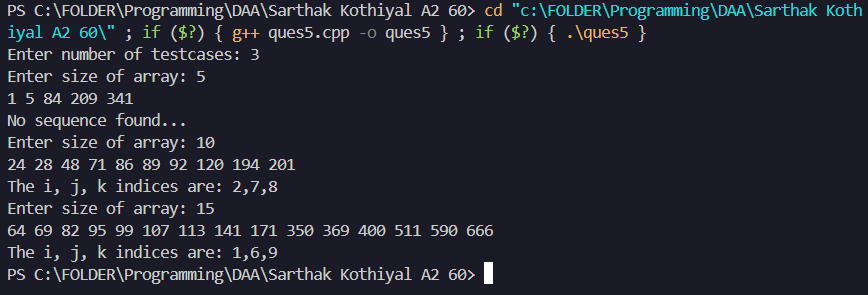
testcases--;

}

return 0;

}

**OUTPUT:**



**PRACTICAL 6**

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.

**ALGORITHM:**

**START**

Step 1: Read the value of ‘n’ from user, also set a counter variable for comparisons to 0.

Step 2: Declare and input the elements in the array.

Step 3: Ask user for target element.

Step 4: Count the pairs and display the count:

1. Initialize a set and counter variable to 0.
2. Insert all elements of the array to the set.
3. Traverse the set and if i+target is present in the set, increment counter by 1.
4. Return the counter value.

**STOP**

**SOURCE CODE:**

#include <iostream>

#include <vector>

#include <unordered\_set>

using namespace std;

int count\_pair(vector<int> arr,int target){

unordered\_set<int> st;

int c=0;

for(int i:arr){

st.insert(i);

}

for(int i:arr){

if(st.find(i+target)!=st.end()){

c++;

}

}

return c;

}

int main(){

int n,val,target,testcases;

cout << "Enter number of testcases: ";

cin >> testcases;

while(testcases>0){

vector<int> arr;

cout << "Enter size of array: ";

cin >> n;

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

cin >> val;

arr.push\_back(val);

}

cout << "Enter target: ";

cin >> target;

int result=count\_pair(arr,target);

cout << "Number of pairs whose difference is equal to " << target << " are: " << result << endl;

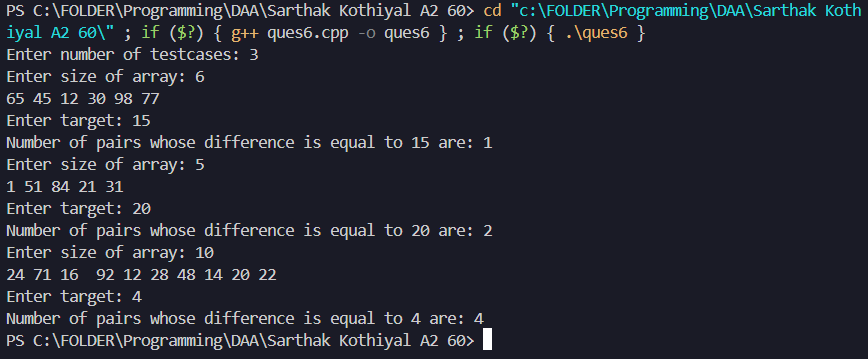
testcases--;

}

return 0;

}

**OUTPUT:**



**PRACTICAL 7**

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.

**ALGORITHM:**

**START**

Step 1: Read the value of ‘n’ from user.

Step 2: Declare and input the elements in the array.

Step 3: Print the element before sorting.

Step 4: Perform insertion sort on the array.

1. Initialize counter for number of shifts and comparisons.
2. Traverse the array from i=1 to i<arr.size(), initialize j=i for each iteration.
3. Traverse the array until j>0
4. Compare adjacent elements, incrementing comp counter with each comparison and if arr[j-1]>arr[j], swap(arr[j-1],arr[j]), decrement j and increment shift counter.
5. Push the shift and comp values into a vector and return the vector.

Step 5: Print the sorted array and the count of Number of Shifts and Comparisons.

**STOP**

**SOURCE CODE:**

#include <iostream>

#include <vector>

#include <algorithm>

#include <cmath>

using namespace std;

vector<int> insertion(vector<int> &arr){

vector<int> data;

int shift=0,comp=0;

for(int i=1;i<arr.size();i++){

int j=i;

while(j>0){

comp++;

if(arr[j-1]>arr[j]){

swap(arr[j-1],arr[j]);

shift++;

j--;

}

else{

break;

}

}

}

data.push\_back(shift);

data.push\_back(comp);

return data;

}

int main(){

int n,val,testcases;

cout << "Enter number of testcases: ";

cin >> testcases;

while(testcases>0){

vector<int> arr;

cout << "Enter size of array: ";

cin >> n;

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

cin >> val;

arr.push\_back(val);

}

cout << "Elements before sorting: [";

for(int i:arr){

cout << i << ",";

}

cout << "]" << endl;;

vector<int> data=insertion(arr);

cout << "Elements in sorted order: [";

for(int i:arr){

cout << i << ",";

}

cout << "]" << endl;

cout << "Number of Shifts: " << data[0] << endl;

cout << "Number of Comparisions: " << data[1];

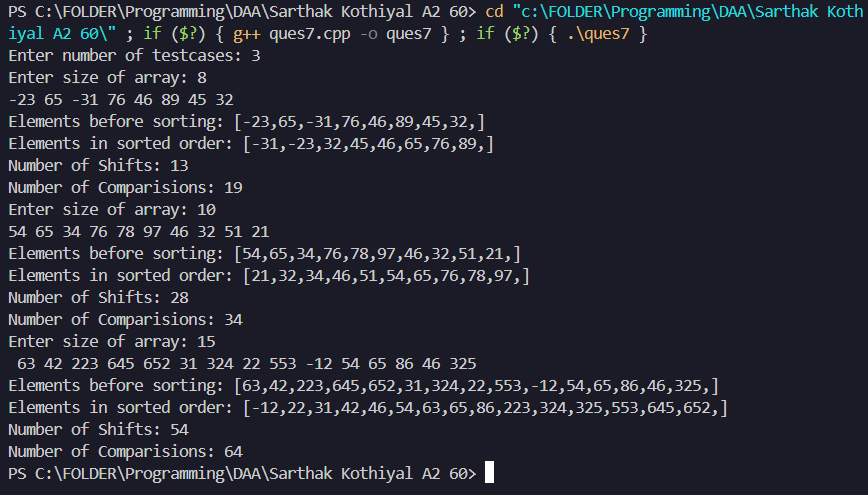
testcases--;

}

return 0;

}

**OUTPUT:**

****

**PRACTICAL 8**

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.

**ALGORITHM:**

**START**

Step 1: Read the value of ‘n’ from user.

Step 2: Declare and input the elements in the array.

Step 3: Print the element before sorting.

Step 4: Perform selection sort on the array.

1. Initialize counter for number of shifts and comparisons.
2. Traverse the array from i=0 to i<arr.size(), initialize min=i for each iteration.
3. Traverse the array from j=i+1 to j<arr.size()
4. Compare adjacent arr[min] with arr[j], incrementing comp counter with each comparison and if arr[j]<arr[min], min=j, if min!=i, swap(arr[i],arr[min]) and increment shift counter.
5. Push the shift and comp values into a vector and return the vector.

Step 5: Print the sorted array and the count of Number of Shifts and Comparisons.

**STOP**

**SOURCE CODE:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

vector<int> selection(vector<int> &arr){

vector<int> data;

int swp=0,comp=0;

for(int i=0;i<arr.size();i++){

int min=i;

for(int j=i+1;j<arr.size();j++){

comp++;

if(arr[j]<arr[min]){

min=j;

}

}

if(min!=i){

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

swp++;

}

}

data.push\_back(swp);

data.push\_back(comp);

return data;

}

int main(){

int n,val,testcases;

cout << "Enter number of testcases: ";

cin >> testcases;

while(testcases>0){

vector<int> arr;

cout << "Enter size of array: ";

cin >> n;

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

cin >> val;

arr.push\_back(val);

}

cout << "Elements before sorting: [";

for(int i:arr){

cout << i << ",";

}

cout << "]" << endl;

vector<int> data=selection(arr);

cout << "Elements in sorted order: [";

for(int i:arr){

cout << i << ",";

}

cout << "]" << endl;

cout << "Number of Swaps: " << data[0] << endl;

cout << "Number of Comparisions: " << data[1];

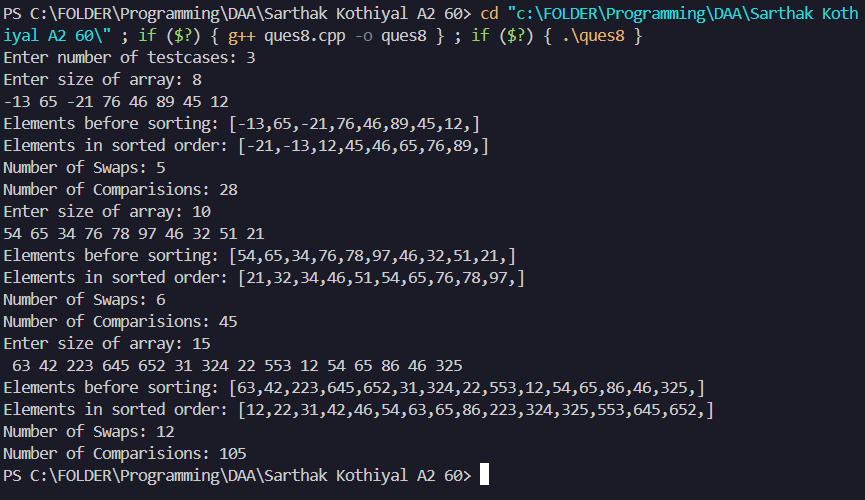
testcases--;

}

return 0;

}

**OUTPUT:**



**PRACTICAL 9**

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))

**ALGORITHM:**

**START**

Step 1: Read the value of ‘n’ from user.

Step 2: Declare and input the elements in the array.

Step 3: Sort the array.

Step 4: Search for duplicates:

1. Traverse the array from i=0 to i<arr.size()
2. If arr[i] == arr[i+1], return arr[i]
3. Else, if the entire array is traversed and duplicates are not found, return -1.

Step 5: If function returns -1, print “Array does not contain any duplicates”.

Step 6: Print the duplicate element.

**STOP**

**SOURCE CODE:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

void merge(vector<int>& arr,int low,int mid,int high){

vector<int> temp;

int i=0;

int left=low,right=mid+1;

while(left<=mid && right<=high){

if(arr[left]<=arr[right]){

temp.push\_back(arr[left++]);

}

else{

temp.push\_back(arr[right++]);

}

}

while(left<=mid){

temp.push\_back(arr[left++]);

}

while(right<=high){

temp.push\_back(arr[right++]);

}

for(int i=low;i<=high;i++){

arr[i]=temp[i-low];

}

}

void SORT(vector<int> &arr,int low,int high){

if(low<high){

int mid=(low+high)/2;

SORT(arr,low,mid);

SORT(arr,mid+1,high);

merge(arr,low,mid,high);

}

}

int check(vector<int> arr){

for(int i=0;i<arr.size()-1;i++){

if(arr[i]==arr[i+1])

return arr[i];

}

return -1;

}

int main(){

int n,val,testcases,target;

cout << "Enter number of testcases: ";

cin >> testcases;

while(testcases>0){

vector<int> arr;

cout << "Enter size of array: ";

cin >> n;

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

cin >> val;

arr.push\_back(val);

}

SORT(arr,0,arr.size()-1);

int result=check(arr);

if(result==-1)

cout << "Array Does Not contain any Duplicate Elements..." << endl;

else

cout << "Array contain Duplicates of: " << result << endl;

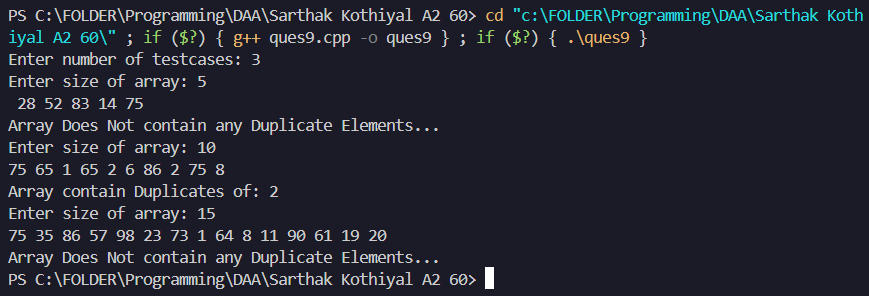
testcases--;

}

return 0;

}

**OUTPUT:**



**PRACTICAL 10**

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.

**ALGORITHM:**

**START**

Step 1: Read the value of ‘n’ from user.

Step 2: Declare and input the elements in the array, also set comparison and inversion counter to 0.

Step 3: Print the element before sorting.

Step 4: Perform merge sort on the array:

1. Set low=0 and high=arr.size()-1, perform a loop till low<high.
2. Find the middle index, mid=(low+high)/2.
3. Recursively divide left half (low to mid).
4. Recursively divide right half (mid+1 to high).
5. Merge the two sorted halves while counting comparisons and inversions.

Step 5: Merge the two unsorted halves:

1. left = low (points to the start of the left half).
2. right = mid + 1 (points to the start of the right half).
3. Create an empty list temp[] to store the merged elements.
4. While left <= mid and right <= high
5. Increment comparison counter (comp).
6. if arr[left] <= arr[right], copy arr[left] to temp[] and increment left.
7. Else (i.e., arr[left] > arr[right]):
8. Copy arr[right] to temp[] and increment right.
9. Increment inversion counter (inver) by (mid - left + 1).
10. Copy Remaining Elements
11. Replace arr with sorted value sin temp[].

Step 5: Print the sorted array and the count of Number of inversions and Comparisons.

**STOP**

**SOURCE CODE:**

#include <iostream>

#include <vector>

using namespace std;

void merge(vector<int>& arr,int low,int mid,int high,int &comp,int &inver){

vector<int> temp;

int left=low,right=mid+1;

while(left<=mid && right<=high){

if(arr[left]<=arr[right]){

temp.push\_back(arr[left++]);

}

else{

temp.push\_back(arr[right++]);

inver+=(mid-left+1);

}

comp++;

}

while(left<=mid){

temp.push\_back(arr[left++]);

}

while(right<=high){

temp.push\_back(arr[right++]);

}

for(int i=low;i<=high;i++){

arr[i]=temp[i-low];

}

}

void divide(vector<int>& arr,int low,int high,int &comp,int &inver){

if(low<high){

int mid=(low+high)/2;

divide(arr,low,mid,comp,inver);

divide(arr,mid+1,high,comp,inver);

merge(arr,low,mid,high,comp,inver);

}

}

int main(){

int n,val,testcases;

cout << "Enter number of test cases: ";

cin >> testcases;

while(testcases>0){

int comp=0,inver=0;

vector<int> arr;

cout << "Enter size of array: ";

cin >> n;

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

cin >> val;

arr.push\_back(val);

}

cout << "Elements before sorting: [";

for(int i:arr){

cout << i << ",";

}

cout << "]" << endl;

divide(arr,0,arr.size()-1,comp,inver);

cout << "Elements after sorting: [";

for(int i:arr){

cout << i << ",";

}

cout << "]" << endl;

cout << "Number of Comparisons: " << comp << endl;

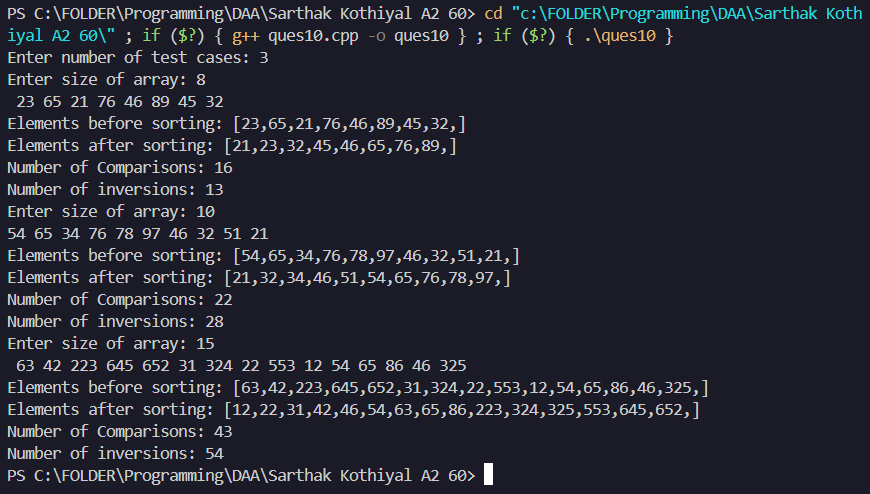
cout << "Number of inversions: " << inver << endl;

testcases--;

}

}

**OUTPUT:**



**PRACTICAL 11**

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.

**ALGORITHM:**

**START**

Step 1: Read the value of ‘n’ from user.

Step 2: Declare and input the elements in the array, also set comparison and swaps counter to 0.

Step 3: Print the element before sorting.

Step 4: Perform quick sort on the array:

1. Set low=0 and high=arr.size()-1, perform a loop till low<high.
2. Find the middle index, mid=(low+high)/2.
3. Find pivot element.
4. Recursively perform quick for on left part of the array low to pi-1.
5. Recursively perform quick for on right part of the array pi+1 to high.

Step 5: Finding pivot element:

1. Find a random index and swap it with last index, and increment swp with 1.
2. Set last element as pivot element.
3. Traverse using a loop from j=low till j<high
4. Increment comp with each comparison, if arr[j]<pi, swap(arr[i],arr[j]), increment swp.
5. Swap(arr[i+1],arr[high]), increment swp
6. Return pivot index.

Step 5: Print the sorted array and the count of Number of Swaps and Comparisons.

**STOP**

**SOURCE CODE:**

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

using namespace std;

int pivot(vector<int>& arr,int low,int high,int &swp,int &comp){

int ran=low+rand()%(high-low+1);

swap(arr[ran],arr[high]);

swp++;

int pi=arr[high];

int i=low-1;

for(int j=low;j<high;j++){

comp++;

if(arr[j]<pi){

i++;

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

swp++;

}

}

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

swp++;

return i+1;

}

void quick(vector<int>& arr,int low,int high,int &swp,int &comp){

if(low<high){

int p=pivot(arr,low,high,swp,comp);

quick(arr,low,p-1,swp,comp);

quick(arr,p+1,high,swp,comp);

}

}

int main(){

srand(time(0));

int n, val,testcase;

cout << "Enter number of testcases: ";

cin >> testcase;

while(testcase>0){

int swp=0,comp=0;

vector<int> arr;

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

cin >> n;

cout << "enter the element of array:";

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

cin >> val;

arr.push\_back(val);

}

cout << "Elements before sorting: [";

for(int i:arr){

cout << i << ",";

}

cout << "]" << endl;

quick(arr,0,n-1,swp,comp);

cout << "Elements after sorting: [";

for(int i:arr){

cout << i << ",";

}

cout << "]" << endl;

cout << "Number of Swaps: " << swp << endl;

cout << "Number of Comparisons: " << comp << endl;

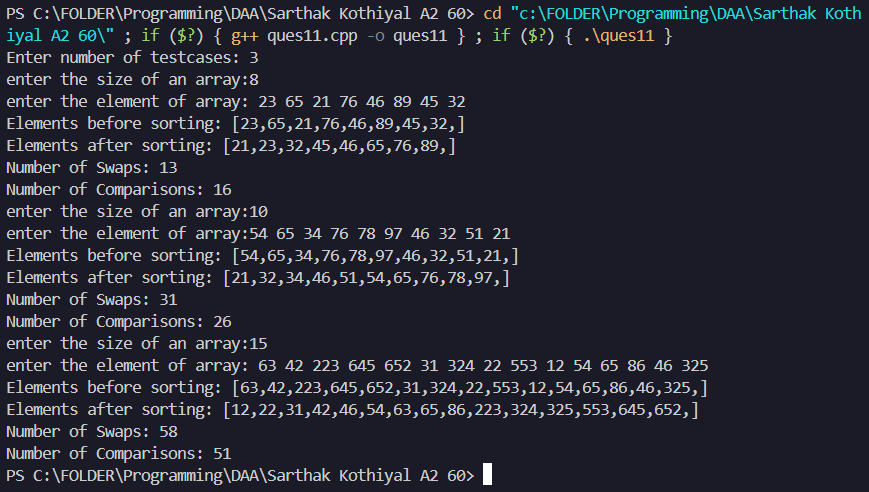
testcase--;

}

return 0;

}

**OUTPUT:**



**PRACTICAL 12**

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))

**ALGORITHM:**

**START**

Step 1: Read the value of ‘n’ from user.

Step 2: Declare and input the elements in the array.

Step 3: Read value of ‘k’ to find the kth smallest element.

Step 4: Find the kth smallest element:

1. If value to k is greater than size of array, return INT\_MIN.
2. Find min and max element of the array.
3. Create a frequency array to store count of each element.
4. Traverse the array and store frequency count of each element.
5. Update freq[i] so it represents position of each element.
6. Create a temporary array temp to store the sorted elements.
7. Traverse the original array in reverse order, placing elements at their respective sorted positions using freq[].
8. Return the kth element from the sorted temp array.

**STOP**

**SOURCE CODE:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int find(vector<int> arr,int k){

if(k>arr.size())

return INT\_MIN;

int min=\*min\_element(arr.begin(),arr.end());

int max=\*max\_element(arr.begin(),arr.end());

int size=max-min+1;

int \*freq=(int\*)calloc(size,sizeof(int));

for(int i=0;i<arr.size();i++){

freq[arr[i]-min]++;

}

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

freq[i]=freq[i-1]+freq[i];

}

int \*temp=(int\*)malloc(size\*sizeof(int));

for(int i=arr.size()-1;i>=0;i--){

int index=--freq[arr[i]-min];

temp[index]=arr[i];

}

return temp[k-1];

}

int main(){

int testcases;

cout << "Enter number of testcase: ";

cin >> testcases;

while(testcases>0){

int n,val,k;

cout << "Enter size of array: ";

cin >> n;

vector<int> arr;

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

cin >> val;

arr.push\_back(val);

}

cout << "Enter value of 'k' to search the kth smallest element: ";

cin >> k;

int result=find(arr,k);

cout << "'kth' smallest element is: " << result << endl;

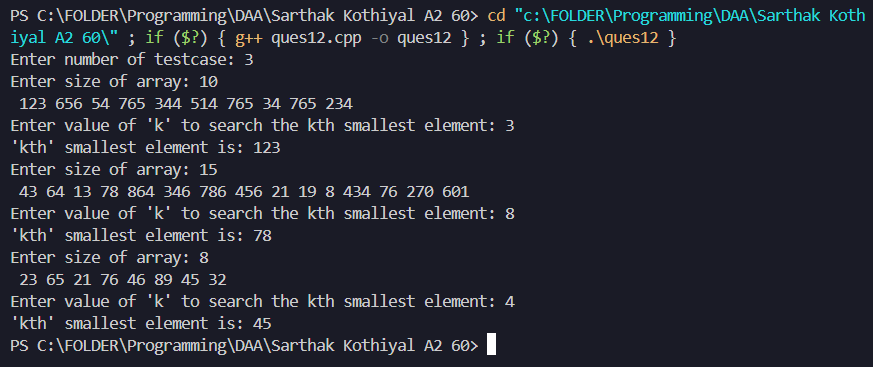
testcases--;

}

return 0;

}

**OUTPUT:**



**PRACTICAL 13**

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

**ALGORITHM:**

**START**

Step 1: Declare and input the String from the user.

Step 2: Find the maximum occurring character in the string:

1. Create a frequency array.
2. Traverse the string and update freq of each character.
3. Set minIndex = -1 and min =1;
4. Traverse the frequency array and if arr[i]>min, update min=arr[i] and minIndex=i.
5. Return i

Step 3: if result == -1, print “No duplicates exists”.

Step 4: Else, Print the most occurring character.

**STOP**

**SOURCE CODE:**

#include <iostream>

using namespace std;

int find(string str){

char \*arr=(char\*)calloc(26,sizeof(char));

for(char ch:str)

arr[ch-'a']++;

int min=1;

int index=-1;

for(int i=0;i<26;i++){

if(arr[i]>min){

min=arr[i];

index=i;

}

}

return index;

}

int main(){

int testcases;

cout << "Enter number of testcase: ";

cin >> testcases;

while(testcases>0){

string str;

cout << "Enter a sequence of charactes: ";

cin >> str;

int result=find(str);

if(result==-1)

cout << "No duplicates...";

else

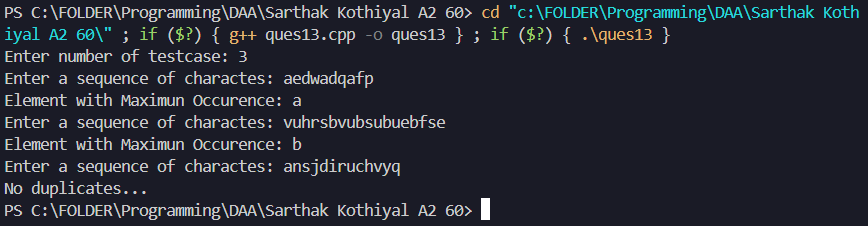
cout << "Element with Maximun Occurence: " << (char)(result+'a') << endl;

testcases--;

}

return 0;}

**OUTPUT:**



**PRACTICAL 14**

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))

**ALGORITHM:**

**START**

Step 1: Read the value of ‘n’ from user.

Step 2: Declare and input the elements in the array.

Step 3: Read target element from the user.

Step 4: Find two number whose sum is equal to target:

1. Sort the array.
2. Set up low and high pointer to 0 and arr.size()-1 respectively.
3. Use a while loop till low < high.
4. Calculate sum=arr[low]+arr[high]
5. If sum==target, print the two number.
6. If sum < target, increment low.
7. Else, decrement high.
8. If the loop terminates, print “no such pair exists”.

**STOP**

**SOURCE CODE:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

void twosum(vector<int> arr,int target){

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

int low=0,high=arr.size()-1;

while(low<high){

int sum=arr[low]+arr[high];

if(sum==target){

cout << "Values: " << arr[low] << " and " << arr[high] << " sum up to target: " << target << endl;

return;

}

else if(sum<target){

low++;

}

else{

high--;

}

}

cout << "No such Elements exist in the Array..." << endl;

}

int main(){

int testcases,n,val,target;

cout << "Enter number of testcase: ";

cin >> testcases;

while(testcases>0){

cout << "Enter size of array: ";

cin >> n;

vector<int> arr;

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

cin >> val;

arr.push\_back(val);

}

cout << "Enter target element: ";

cin >> target;

twosum(arr,target);

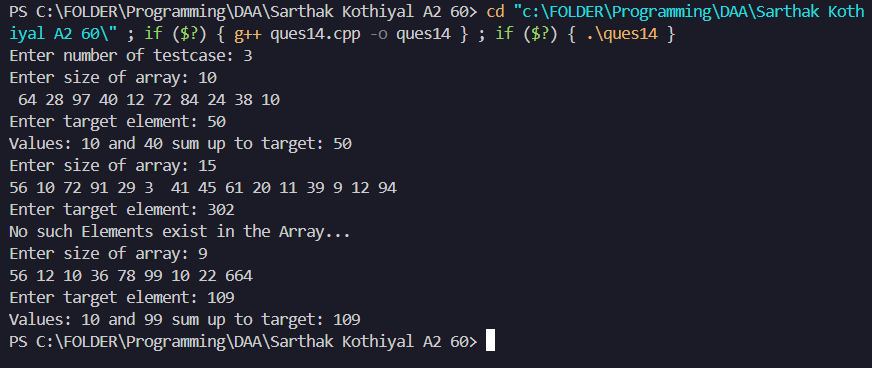
testcases--;

}

return 0;

}

**OUTPUT:**



**PRACTICAL 15**

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))

**ALGORITHM:**

**START**

Step 1: Read the value of ‘m’ and ‘n’ from user.

Step 2: Declare and input the elements in two different arrays: arr1 and arr2.

Step 3: Store the common elements in another array, result:

1. Set two pointers i and j to 0.
2. Traverse both the arrays till i<arr1.size() and j<arr2.size().
3. If arr1[i]==arr2[j], push the element to result.
4. If arr[i]<arr[j], increment i.
5. Else, increment j.

Step 4: Display the contents of result.

**STOP**

**SOURCE CODE:**

#include <iostream>

#include <vector>

using namespace std;

vector<int> intersection(vector<int> arr1,vector<int> arr2){

vector<int> arr3;

int i=0,j=0;

while(i<arr1.size() && j<arr2.size()){

if(arr1[i]==arr2[j]){

arr3.push\_back(arr1[i]);

i++;j++;

}

else if(arr1[i]<arr2[j]){

i++;

}

else{

j++;

}

}

return arr3;

}

int main(){

int m,n,testcases,val;

cout << "Enter number of testcases: ";

cin >> testcases;

while(testcases>0){

vector<int> arr1,arr2;

cout << "Enter size of first array: ";

cin >> m;

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

cin >> val;

arr1.push\_back(val);

}

cout << "Enter size of second array: ";

cin >> n;

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

cin >> val;

arr2.push\_back(val);

}

vector<int> result=intersection(arr1,arr2);

cout << "Common elements in both arrays: ";

for(int i:result){

cout << i << " ";

}

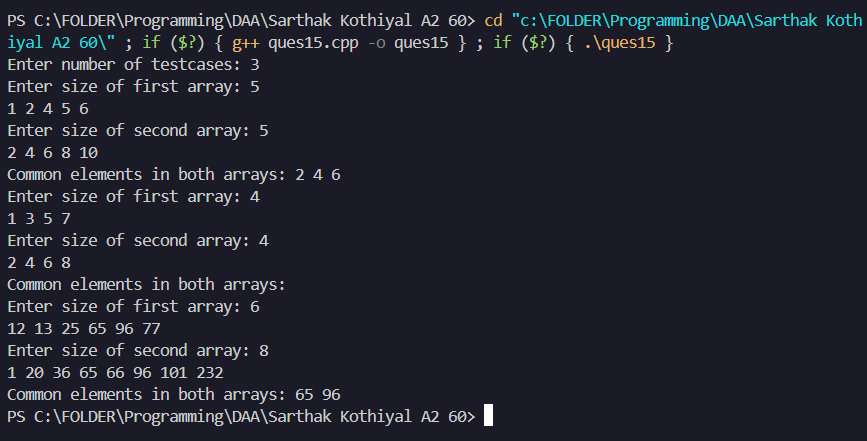
testcases--;

}

return 0;

}

**OUTPUT:**



**PRACTICAL 16**

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.

**ALGORITHM:**

**START**

Step 1: Read the value of ‘v’ and ‘e’ from user.

Step 2: Declare and input the elements in adjacency matrix for graph.

Step 3: Provide source and destination source.

Step 4: Check for path using dfs from source to destination.

* Create a visited array and stack for dfs.
* Push source node to stack and mark as visited
* Until stack is not empty :-
* Pop from stack
* If poped element is destination, return true
* Visit all connected node to poped element
* Push them to stack if they are not visited and mark them as visited
* After stack is empty, return false

**STOP**

**SOURCE CODE:**

#include <iostream>

#include <vector>

#include <stack>

using namespace std;

bool checkpath(vector<vector<int>>& graph, int source, int destination,int n) {

    vector<int> visited(n+1,0);

    stack<int> st;

    st.push(source);

    visited[source]=1;

    while(!st.empty()){

        int node=st.top();

        st.pop();

        if(node==destination)

            return true;

        for(int i=n-1;i>=0;i--){

            if(graph[node][i] && visited[i]!=1){

                st.push(i);

                visited[i]=1;

            }

        }

    }

    return false;

}

int main(){

    int v,e;

    cout << "Enter number of vertices and edges: ";

    cin >> v >> e;

    vector<vector<int>> graph(v,vector<int>(v,0));

    for(int i=0;i<v;i++){

        int m,n;

        cin >> m >> n;

        graph[m][n]++;

        graph[n][m]++;

    }

    int s,d;

    cout << "Provide Source and Destination: ";

    cin >> s >> d;

    if(checkpath(graph,s,d,v)){

        cout << "Path Exist..." << endl;

    }

    else{

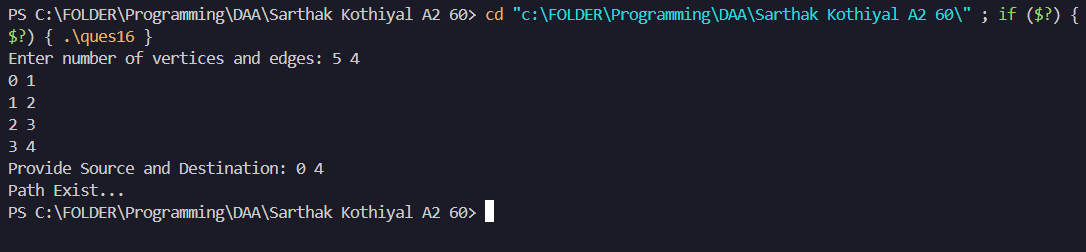
        cout << "Path Do Not Exist..." << endl;

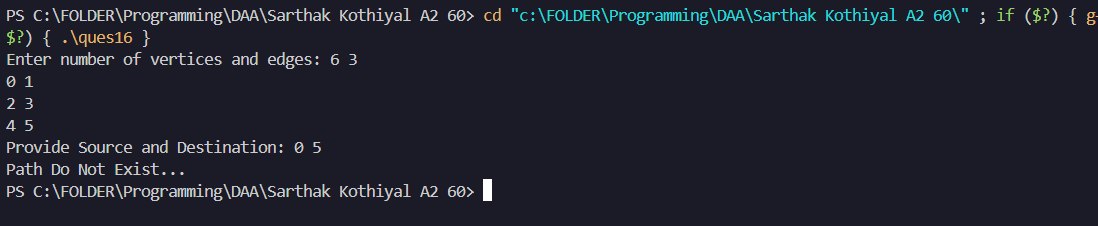
    }

    return 0;

}

**OUTPUT:**





**PRACTICAL 17**

Given a graph, design an algorithm and implement it using a program to find if a graph is bipartite or not

**ALGORITHM:**

**START**

Step 1: Read the value of ‘v’ and ‘e’ from the user.

Step 2: Declare and input the adjacency list representation for the graph.

Step 3: Initialize a color array of size ‘v’ with -1 (uncolored).

Step 4: Iterate through all nodes and perform BFS from an unvisited node:

* Assign the source node a color (0).
* Create a queue and push the source node into it.
* Until the queue is not empty:
* Pop a node from the queue.
* Check all its adjacent nodes:
* If the adjacent node is uncolored, assign it the opposite color and push it to the queue.
* If the adjacent node has the same color as the current node, return false

Step 5: If BFS completes without conflicts, return true

**STOP**

**SOURCE CODE:**

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

bool isBipartite(vector<vector<int>>& graph) {

int n=graph.size();

vector<int> colour(n,-1);

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

if(colour[i]==-1){

queue<int> que;

que.push(i);

colour[i]=0;

while(!que.empty()){

int node=que.front();

que.pop();

for(int it:graph[node]){

if(colour[it]==-1){

colour[it]=!colour[node];

que.push(it);

}

else if(colour[it]==colour[node]){

return false;

}

}

}

}

}

return true;

}

int main(){

int v,e;

cout << "Enter number of vertices and edges: ";

cin >> v >> e;

vector<vector<int>> graph(v);

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

int m,n;

cin >> m >> n;

graph[m].push\_back(n);

graph[n].push\_back(m);

}

if(isBipartite(graph)){

cout << "The Graph is Bipartite..." << endl;

}

else{

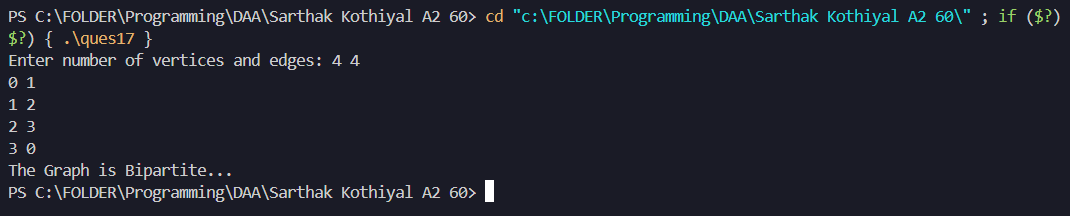
cout << "The Graph is Not Bipartite..." << endl;

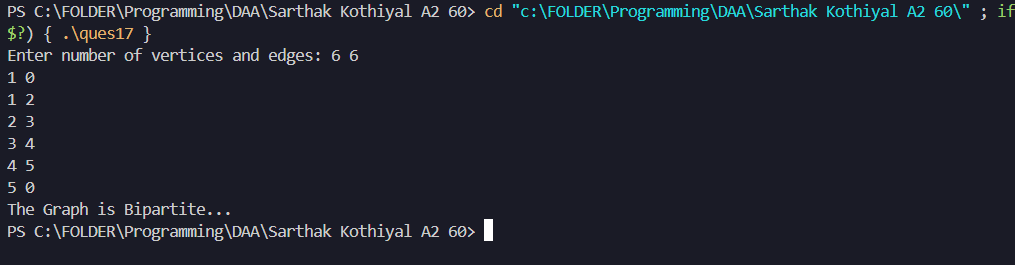
}

return 0;

}

**OUTPUT:**





**PRACTICAL 18**

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

**ALGORITHM:**

**START**

Step 1: Read the value of ‘v’ and ‘e’ from the user.

Step 2: Declare and input the adjacency list representation for the graph.

Step 3: Initialize a visited array of size ‘v’ with value 0.

Step 4: Iterate through all nodes and perform DFS from an unvisited node:

• Mark the node as visited.

• For each adjacent node:

• If the adjacent node is unvisited, recursively call DFS on it.

• If the adjacent node is already visited and is not the parent, return true

Step 5: If DFS completes without detecting a cycle, return false

**STOP**

**SOURCE CODE:**

#include <iostream>

#include <vector>

using namespace std;

bool dfs(int s,vector<vector<int>> &adj,vector<int> &vis,int parent){

vis[s]=1;

for(auto it:adj[s]){

if(vis[it]==0){

if(dfs(it,adj,vis,s)){

return true;

}

}

else if(it!=parent){

return true;

}

}

return false;

}

bool isCycle(vector<vector<int>>& adj) {

vector<int> vis(adj.size(),0);

for(int i=0;i<adj.size();i++){

if(vis[i]==0){

if(dfs(i,adj,vis,-1)){

return true;

}

}

}

return false;

}

int main(){

int v,e;

cout << "Enter number of vertices and edges: ";

cin >> v >> e;

vector<vector<int>> graph(v);

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

int m,n;

cin >> m >> n;

graph[m].push\_back(n);

graph[n].push\_back(m);

}

if(isCycle(graph)){

cout << "The Graph contain cycle..." << endl;

}

else{

cout << "The Graph does Not contain cycle..." << endl;

}

return 0;

return 0;

}

**OUTPUT:**

