**Week 1**

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)

#include<iostream>

#include<string.h>

#include<time.h>

#include<iomanip>

using namespace std;

int main(){

int T=10,arr[500000],size,key;

clock\_t t;

string answer="";

answer.append("size\tkey\tPresent / Not Present \n");

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

size=rand()%500000;

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

arr[i]=rand()%500000;

}

key=rand()%5000;

t = clock();

int count=0;

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

if(arr[j]==key){

t = clock() - t;

double time\_taken = ((double)t)/CLOCKS\_PER\_SEC;

answer.append(to\_string(size)+"\t"+to\_string(key)+"\t");

answer.append("Present"+"\t"+to\_string(count+1)+"\t");

answer.append(to\_string(time\_taken)+"\n");

break;

}

count++;

}

if(count==size){

t = clock() - t;

double time\_taken = ((double)t)/CLOCKS\_PER\_SEC;

answer.append(to\_string(size)+"\t"+to\_string(key)+"\t");

answer.append("Not Present"+"\t"+to\_string(count) + "\t");

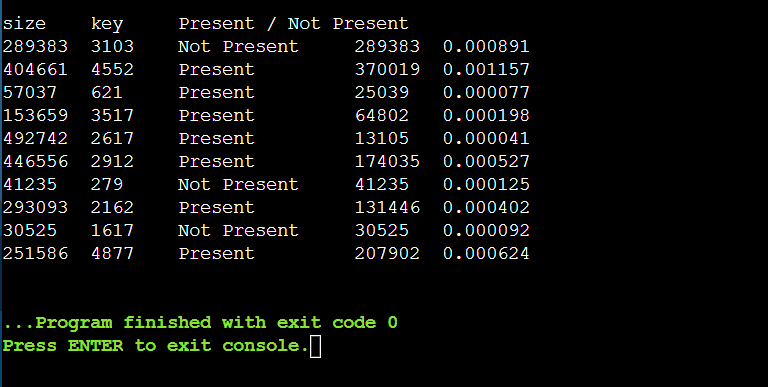
answer.append(to\_string(time\_taken)+"\n");

}

}

cout<<endl<<answer;

}

OUTPUT:

1. 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(log n), where n is the size of input).

#include<iostream>

#include<string.h>

#include<time.h>

#include<iomanip>

#include <bits/stdc++.h>

using namespace std;

int main(){

int T,arr[500000],size,key;

clock\_t t;

string answer="";

answer.append("size\tkey\tPresent / Not Present \n");

T=1000;

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

size=rand()%500000;

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

arr[i]=rand()%500000;

}

key=rand()%500000;

sort(&arr[0],&arr[size]);

t = clock();

int count=0,flag=0;

int start=0,end=size-1,mid;

while(start <=end){

mid=(start+end)/2;

if(arr[mid]==key){

count++;

flag++;

t = clock() - t;

double time\_taken = ((double)t)/CLOCKS\_PER\_SEC;

answer.append(to\_string(size)+"\t"+to\_string(key)+ "\t" +"Present"+"\t");

answer.append(to\_string(count+1)+"\t"+to\_string(time\_taken)+"\n");

break;

}

else if(key<arr[mid]){

end=mid-1;

count++;

}

else if(key>arr[mid]){

count++;

start=mid+1;

} }

if(flag==0){

t = clock() - t;

double time\_taken = ((double)t)/CLOCKS\_PER\_SEC;

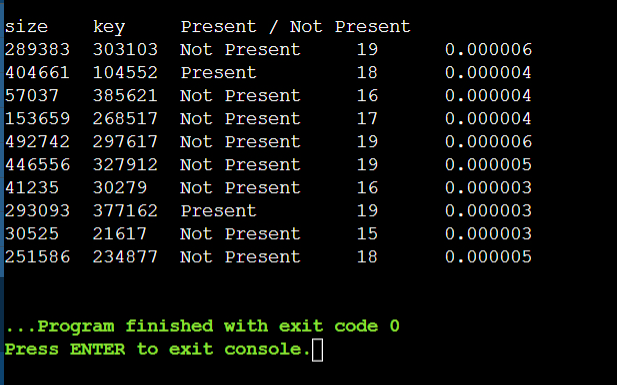
answer.append(to\_string(size)+"\t"+to\_string(key)+"\t"+"NotPresent "+"\t");

answer.append(to\_string(count)+"\t"+to\_string(time\_taken)+"\n");

} }

cout<<endl<<answer;

}

OUTPUT:

1. Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether a given key element is present in the sorted array or not.

For an array arr[n], search at the indexes arr [0], arr [2], arr [4], …. ,arr [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

#include<iostream>

#include<string.h>

#include<time.h>

#include<iomanip>

#include <bits/stdc++.h>

using namespace std;

int main(){

int T,arr[500000],size,key;

clock\_t t;

string answer="";

answer.append("size\tkey\tPresent / Not Present \n");

T=10;

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

size=rand()%500000;

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

arr[i]=rand()%500000;

}

key=rand()%500000;

sort(&arr[0],&arr[size]);

t = clock();

int flag=0,index=0,count=0;

for (int i = 0; i < size; i=i+2){

if(arr[i]==key){ count++;

t = clock() - t;

double time\_taken = ((double)t)/CLOCKS\_PER\_SEC;

answer.append(to\_string(size)+"\t"+to\_string(key)+"\t");

answer.append("Present"+"\t"+to\_string(count+1)+"\t");

answer.append(to\_string(time\_taken)+"\n");

index=i;

flag++;

break;

}

if(arr[i]>key){

count++;

index=i-2;

break;

}

count++;

}

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

if(arr[i]==key){

count++;

t = clock() - t;

double time\_taken = ((double)t)/CLOCKS\_PER\_SEC;

answer.append(to\_string(size)+"\t"+to\_string(key)+"\t");

answer.append("Present"+"\t"+to\_string(count+1)+"\t");

answer.append(to\_string(time\_taken)+"\n");

flag++;

break;

}

count++;

}

if(flag==0){

count++;

t = clock() - t;

double time\_taken = ((double)t)/CLOCKS\_PER\_SEC;

answer.append(to\_string(size)+"\t"+to\_string(key)+"\t");

answer.append("Present"+"\t"+to\_string(count)+"\t");

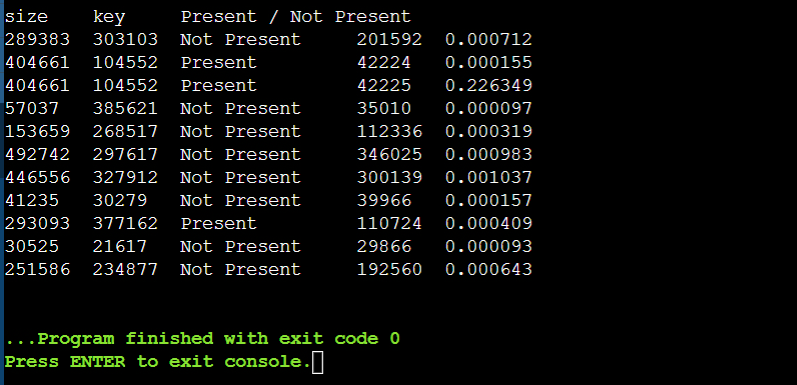
answer.append(to\_string(time\_taken)+"\n");

}

}

cout<<endl<<answer;

}

OUTPUT:

**Week 2**

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(log n))

#include<iostream>

#include<string.h>

#include<time.h>

#include<iomanip>

#include <bits/stdc++.h>

using namespace std;

int main(){

int T,arr[500000],size,key;

clock\_t t;

string answer="";

answer.append("size\tkey\tPresent/NP iteraions\ttime \toccurance\n");

T=1000;

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

size=rand()%500000;

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

arr[i]=rand()%500000;

}

key=rand()%500000;

sort(&arr[0],&arr[size]);

t = clock();

int count=0,flag=0;

int start=0,end=size-1,mid,countii=0;

while(start <=end){

mid=(start+end)/2;

if(arr[mid]==key){

int temp= mid-1;

flag++;

countii++;

while (arr[temp]==key){

count++;

countii++;

temp--;

}

while (arr[mid+1]==key){

count++;

countii++;

mid++;

}

t = clock() - t;

double time\_taken = ((double)t)/CLOCKS\_PER\_SEC;

answer.append(to\_string(size)+"\t"+to\_string(key)+"\t"+"Present "+"\t");

answer.append(to\_string(count+1) +"\t"+to\_string(time\_taken)+"\t");

answer.append(to\_string(countii)+"\n");

break;

}

else if(key<arr[mid]){

end=mid-1;

count++;

}

else if(key>arr[mid]){

count++;

start=mid+1;

}

}

if(flag==0){

t = clock() - t;

double time\_taken = ((double)t)/CLOCKS\_PER\_SEC;

answer.append(to\_string(size)+"\t"+to\_string(key)+"\t"+"Not Present ");

answer.append("\t"+to\_string(count) +"\t"+to\_string(time\_taken)+"\t");

answer.append(to\_string(countii)+"\n");

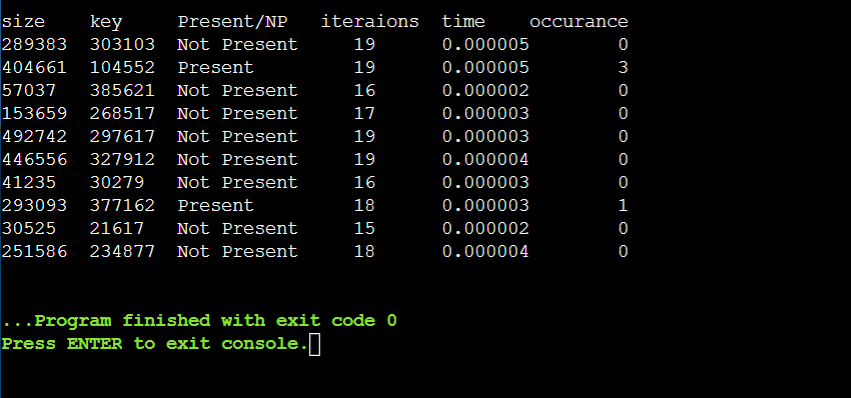
}

}

cout<<endl<<answer;

}

OUTPUT:



1. 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].

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

bool findTriplet(int arr[], int n){

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

int i = 0, j = k - 1;

while (i < j) {

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

cout<<"Triplet found: " << arr[i] << "\t+\t" << arr[j];

cout<<"\t=\t" << arr[k] << "\n";

cout<<"at indices : " << i << "\t\t" << j << "\t\t"

cout<< << k << "\n";

return true;

}

else if (arr[i] + arr[j] < arr[k])

i++;

else

j--;

} } return false;

}

int main(){

int arr[500000];

int n=500000;

cout<<"size => "<<n<<endl;

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

arr[i]=rand()%500000;

}

bool tripletFound = findTriplet(arr, n);

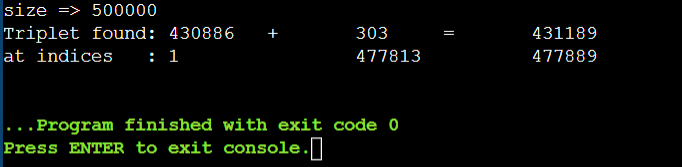
if (!tripletFound)

cout << "No triplet found\n";

return 0;

}

OUTPUT:



1. Given an array of non-negative 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.

#include <iostream>

#include <algorithm>

using namespace std;

int count\_pairs(int arr[], int n, int K) {

sort(arr, arr + n);

int i = 0, j = 1, count = 0;

while (j < n) {

int diff = arr[j] - arr[i];

if (diff == K) {

count++;

i++;

j++;

}

else if (diff < K) { j++; }

else { i++; }

} return count;

}

int main() {

int arr[500000];

int n=500000;

cout<<"size : "<<n<<endl;

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

arr[i]=rand()%500000;

}

int K = rand()%500000;

int result = count\_pairs(arr, n, K);

cout << "Key : " << K << endl;

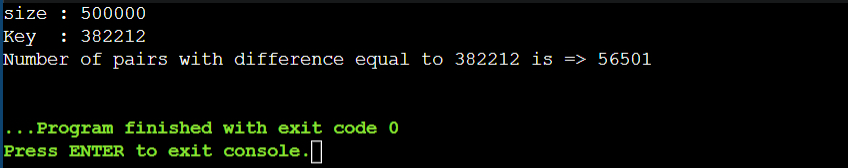
cout << "Number of pairs with difference equal to "<< K <<" is => ";

cout << result << endl;

return 0;

}

OUTPUT:



**Week 3**

1. 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 required for sorting the array.

( shifts - total number of times the array elements are shifted from their place)

#include <iostream>

using namespace std;

void insertionSort(int array[], int size) {

int shift=0,comp=0;

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

int key = array[i];

int j = i - 1;

while (key < array[j] && j >= 0) {

array[j + 1] = array[j];

shift++; comp++; --j;

}

array[j + 1] = key; shift++;

}

cout <<"Sorted array : ";

for (int i = 0; i < size; i++) { cout << array[i] << " "; }

cout << endl;

cout<<"comparision : "<<comp<<endl;

cout<<"swaps : "<<shift<<endl;

}

int main() {

int data[50]; int size=25;

cout<<endl<<"size : "<<size<<endl;

for (int i = 0; i < size; i++) { data[i]=rand()%50; }

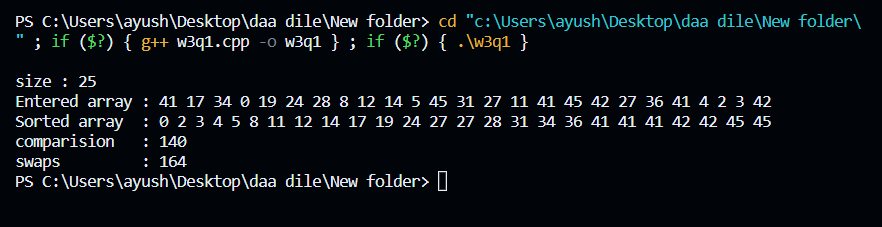
cout <<"Entered array : ";

for (int i = 0; i < size; i++) { cout << data[i] << " "; }

cout << endl;

insertionSort(data, size);

}

OUTPUT:

1. 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.

#include<iostream>

using namespace std;

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

int i, j, minimum,swap=0,comp=0;

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

minimum = i;

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

if (arr[j] < arr[minimum]){ minimum = j; }

comp++;

}

if (minimum!=i){

int temp = arr[minimum];

arr[minimum] = arr[i];

arr[i] = temp;

} swap++;

}

cout<<"comparision : "<<comp<<endl<<"swaps : "<<swap<<endl;

}

int main(){

int arr[] = {645, 652, 31, 324, 22, 553, 12, 54, 65, 86, 46, 325};

int n = 8;

cout <<"Entered array : ";

for (int i=0; i < n; i++){ cout << arr[i] << " "; }

cout << endl;

selectionSort(arr, n);

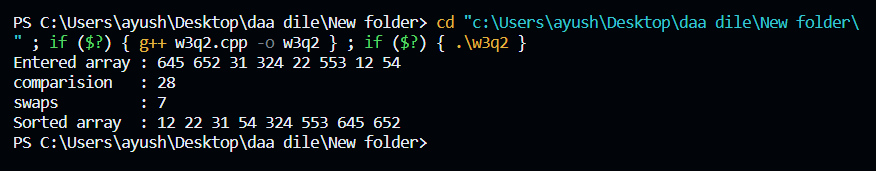
cout << "Sorted array : ";

for (int i=0; i < n; i++){ cout << arr[i] << " "; }

cout << endl;

return 0;

}

OUTPUT:

1. 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))

#include <iostream>

#include <algorithm>

using namespace std;

bool hasDuplicates(int arr[], int n) {

sort(arr, arr + n);

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

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

return true;

}

return false;

}

int main() {

int n;

cout << "Enter the number of elements: ";

cin >> n;

int arr[n];

cout << "Enter " << n << " elements:\n";

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

cin >> arr[i];

if (hasDuplicates(arr, n))

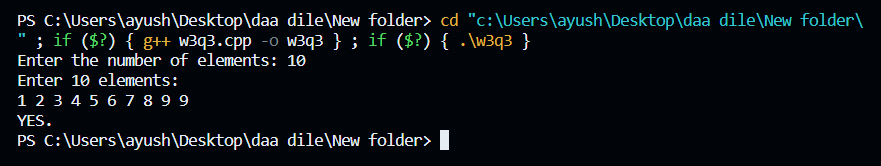
cout << "YES."<<endl;

else

cout << "NO."<<endl;

return 0;

}

OUTPUT:

**Week 4**

1. 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.

#include<iostream>

#include<string.h>

#include<time.h>

#include<iomanip>

using namespace std;

long long merge(int arr[], int left, int mid, int right) {

int i, j, k;

long long inv\_count = 0;

int n1 = mid - left + 1;

int n2 = right - mid;

int L[n1], R[n2];

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

L[i] = arr[left + i];

for (j = 0; j < n2; j++)

R[j] = arr[mid + 1 + j];

i = 0;

j = 0;

k = left;

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

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

inv\_count += (mid - i) + 1;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

return inv\_count;

}

long long mergeSort(int arr[], int left, int right) {

long long inv\_count = 0;

if (left < right) {

int mid = left + (right - left) / 2;

inv\_count += mergeSort(arr, left, mid);

inv\_count += mergeSort(arr, mid + 1, right);

inv\_count += merge(arr, left, mid, right);

}

return inv\_count;

}

int main() {

int n;

cout << "Enter the number of elements: ";

cin >> n;

int arr[n];

cout << "Enter " << n << " elements:\n";

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

cin >> arr[i];

long long inv\_count = mergeSort(arr, 0, n - 1);

cout << "\nSorted array: ";

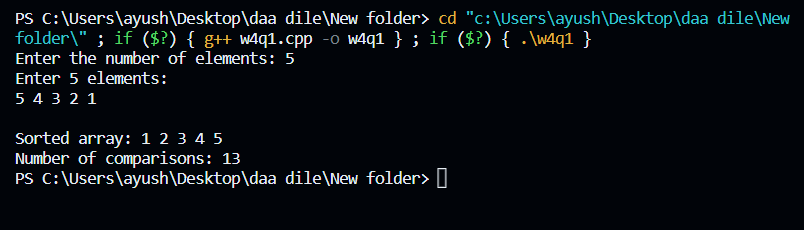
for (int i = 0; i < n; i++)

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

cout << "\nNumber of comparisons: " << inv\_count << endl;

return 0;

}

OUTPUT:

1. 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.

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

int partition(int arr[],int left,int right,long long &comp,long long &swaps){

int pivotIndex = left + rand() % (right - left + 1);

int pivot = arr[pivotIndex];

swaps++;

swap(arr[pivotIndex], arr[right]);

int i = left - 1;

for (int j = left; j < right; j++) {

comp++;

if (arr[j] < pivot) {

i++;

swaps++;

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

}

}

swaps++;

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

return i + 1;

}

void quickSort(int arr[],int left,int right,long long &comp,long long &swaps) {

if (left < right) {

int pivotIndex = partition(arr, left, right, comp, swaps);

quickSort(arr, left, pivotIndex - 1, comp, swaps);

quickSort(arr, pivotIndex + 1, right, comp, swaps);

}

}

int main() {

int n;

cout << "Enter the number of elements: ";

cin >> n;

int arr[n];

cout << "Enter " << n << " elements:\n";

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

cin >> arr[i];

srand(time(NULL));

long long comparisons = 0;

long long swaps = 0;

quickSort(arr, 0, n - 1, comparisons, swaps);

cout << "\nSorted array: ";

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

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

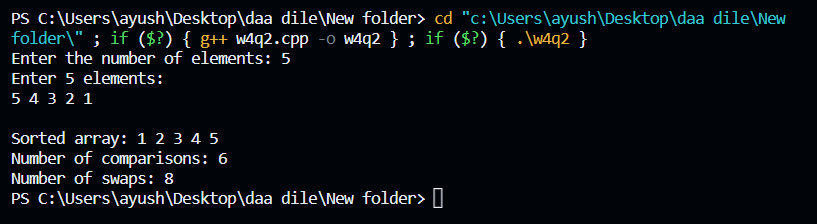
cout << "\nNumber of comparisons: " << comparisons << endl;

cout << "Number of swaps: " << swaps << endl;

return 0;

}

OUTPUT:



1. 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))

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

int partition(int arr[], int left, int right) {

int pivotIndex = left + rand() % (right - left + 1);

int pivot = arr[pivotIndex];

swap(arr[pivotIndex], arr[right]);

int i = left - 1;

for (int j = left; j < right; j++) {

if (arr[j] < pivot) {

i++;

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

}

}

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

return i + 1;

}

int quickSelect(int arr[], int left, int right, int k) {

if (left == right)

return arr[left];

int pivotIndex = partition(arr, left, right);

if (k == pivotIndex)

return arr[k];

else if (k < pivotIndex)

return quickSelect(arr, left, pivotIndex - 1, k);

else

return quickSelect(arr, pivotIndex + 1, right, k);

}

int main() {

int n;

cout << "Enter the number of elements: ";

cin >> n;

int arr[n];

cout << "Enter " << n << " elements:\n";

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

cin >> arr[i];

int k;

cout << "Enter the value of K: ";

cin >> k;

srand(time(NULL));

int kthSmallest = quickSelect(arr, 0, n - 1, k - 1);

cout << "Kth smallest element is: " << kthSmallest << endl;

return 0;

}

OUTPUT:

