**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)

**Algorithm**

* Initialize a counter called "comparisons" to 0 and a flag "found" to false.
* Loop through each element of the array.
* For each element, increment the "comparisons" counter by 1.
* Check if the current element is equal to the key:
* If found set "found" to true and exit the loop immediately.
* After the loop, check whether the key was found and display the total number of comparisons.

**Code**

#include <iostream>

#include <vector>

using namespace std;

bool search(const vector<int>& arr, int key, int &comparisons) {

comparisons = 0;

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

comparisons++;

if (arr[i] == key)

return true;

}

return false;

}

int main() {

int n, key, comparisons;

cout << "Enter number of elements"<<endl;

cin >> n;

vector<int> arr(n);

cout << "Enter " << n << " numbers "<<endl;

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

cin >> arr[i];

}

cout << "Enter key to search"<<endl;

cin >> key;

bool found = search(arr, key, comparisons);

if (found)

cout << "Key " << key << " is present in the array" << endl;

else

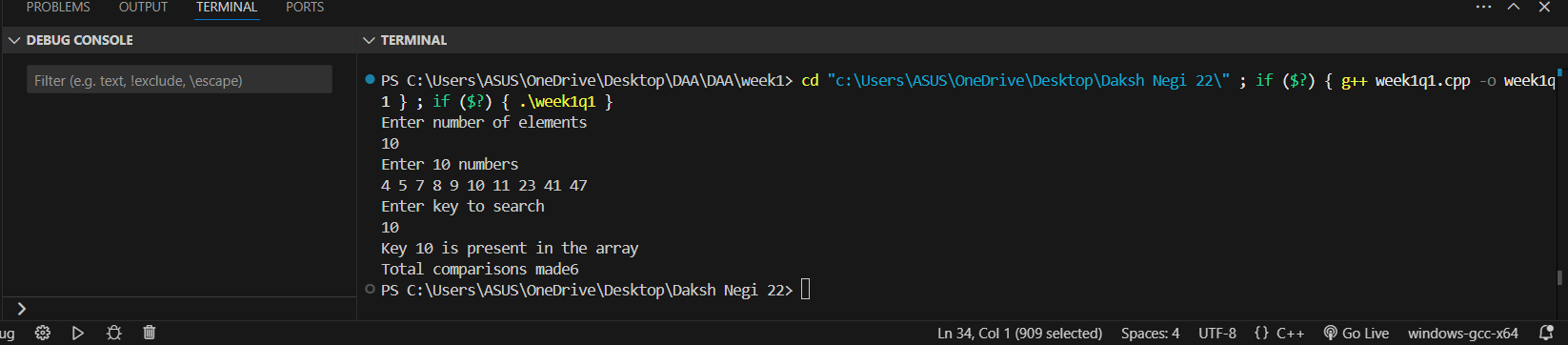
cout << "Key " << key << " is not present in the array" << endl;

cout << "Total comparisons made" << comparisons << endl;

return 0;

}

Output



II. 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**

* Initialize a counter comparisons to 0.
* Set left to 0 and right to n-1 (where n is the array size).
* While left is less than or equal to right:

Increment comparisons by 1.

Calculate mid as (left + right) / 2.

If the element at mid equals the key, return true.

Otherwise, if the key is less than the element at mid, update right to mid - 1.

Otherwise, update left to mid + 1.

* If the loop ends without finding the key, return false.
* Check whether the key was found and display the total number of comparisons made.

**Code**

#include <iostream>

#include <vector>

using namespace std;

bool binarySearch(vector<int>& arr, int key, int &comparisons) {

int left = 0, right = arr.size() - 1;

comparisons = 0;

while (left <= right) {

comparisons++;

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

if (arr[mid] == key)

return true;

else if (key < arr[mid])

right = mid - 1;

else

left = mid + 1;

}

return false;

}

int main() {

int n, key, comparisons;

cout << "Enter number of elements: ";

cin >> n;

vector<int> arr(n);

cout << "Enter " << n << " sorted positive integers: ";

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

cin >> arr[i];

}

cout << "Enter key to search: ";

cin >> key;

bool found = binarySearch(arr, key, comparisons);

if (found)

cout << "Key " << key << " is present in the array." << endl;

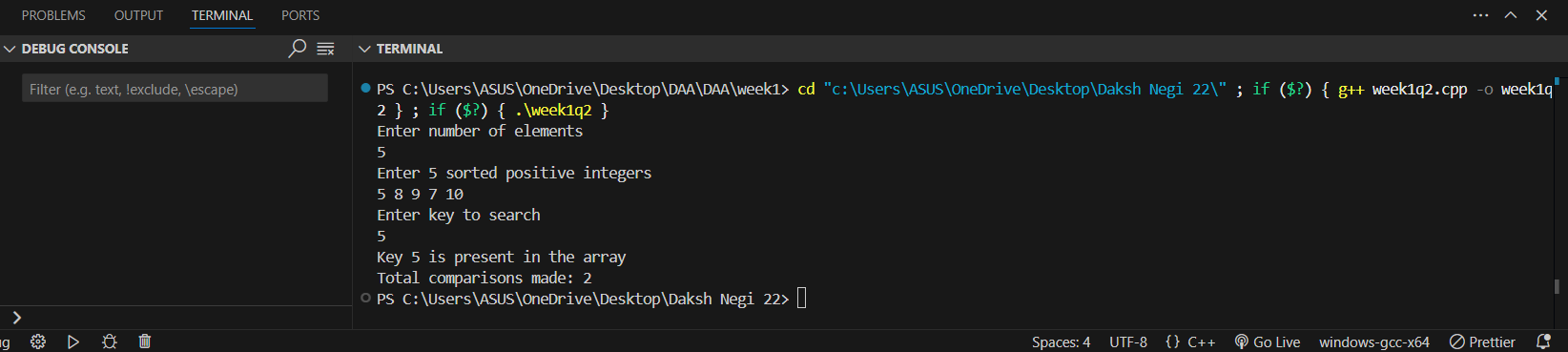
else

cout << "Key " << key << " is not present in the array." << endl;

cout << "Total comparisons made: " << comparisons << endl;

return 0;

}



III.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

**Algorithm**

* Start by initializing a counter comparisons to 0.
* Let n be the size of the sorted array.
* Set an index prev to 0 and define a fixed jump size of 2.
* While prev is less than n and the element at index prev is less than the key:

Increment comparisons.

Check if prev + jump is beyond the array bounds or if the element at index prev + jump is greater than or equal to the key:

If so, break out of the loop; this defines the interval [prev, min(prev + jump, n)] where the key could be.

Otherwise, update prev by adding the jump size.

* Perform a linear search starting from index prev up to the smaller of prev + jump or n:

For each element in this interval, increment comparisons.

If the element equals the key, return that the key is found.

* If the key is not found in the interval, return that the key is not present.
* The total comparisons count records the number of comparisons made during both the jump and linear search phases.

Code