**Problem Statement:** Design and implement both normal and recursive Binary Search algorithms using the Divide and Conquer method for a given input. Determine the time required to search an element for a large value of *n*. Compare the time taken by Binary Search with the time taken by Linear Search.

**Brief about the Problem:** Searching is a fundamental operation in computer science, used to retrieve information from data structures. Linear Search is a simple method where each element is checked sequentially until the target is found. However, it is inefficient for large datasets. Binary Search is a more efficient approach that utilizes the divide-and-conquer technique, reducing the problem size by half in each iteration. This code aims to compare the execution time of Linear Search, Iterative Binary Search, and Recursive Binary Search for large input sizes.

**Algorithm:**

1. **Linear Search Algorithm:**
   * Traverse through the array sequentially.
   * Compare each element with the target value.
   * If the element matches the target, return its index.
   * If the end of the array is reached without finding the target, return -1.
2. **Binary Search Algorithm (Iterative):**
   * Initialize start as 0 and end as size-1.
   * Compute the middle index: mid = start + (end - start) / 2.
   * If the middle element is the target, return its index.
   * If the target is smaller than the middle element, repeat the search in the left half.
   * If the target is larger, repeat the search in the right half.
   * Continue until start exceeds end, returning -1 if the target is not found.
3. **Binary Search Algorithm (Recursive):**
   * Base Case: If start > end, return -1.
   * Compute the middle index: mid = start + (end - start) / 2.
   * If arr[mid] is the target, return mid.
   * If the target is smaller, recursively search in the left half.
   * If the target is larger, recursively search in the right half.

**Analysis of Algorithm:**

* **Time Complexity:**
* **Linear Search:** O(n) - Each element is checked once in the worst case.
* **Binary Search (Iterative & Recursive):** O(log n) - The search space is halved in each iteration.
* **Space Complexity:**
* **Linear Search:** O(1) - No additional space required.
* **Binary Search (Iterative):** O(1) - Only a few extra variables are used.
* **Binary Search (Recursive):** O(log n) - Due to recursive function calls.

**Program:**

#include<iostream>

#include<cstdlib>

#include<ctime>

#include<algorithm>

#include<chrono>

using namespace std;

using namespace std::chrono;

int LinearSearch(long  arr[] , int size , int target){

    for(int i = 0;i<size;i++){

        if(arr[i]==target){

            return i;

        }

    }

    return -1;

}

int RecursiveBinary(long arr[] , int size , int target, int start , int end){

    while(start<=end){

        long mid = start + (end - start)/2;

        if(target<arr[mid]){

           return RecursiveBinary(arr,size,target,start,mid-1);

        }else if(target>arr[mid]){

           return RecursiveBinary(arr,size,target,mid+1,end);

        }else{

            return mid;

        }

    }return -1;

}

int BinarySearch(long  arr[], int size , int target){

    long start = 0;

    long end = size-1;

    while(start<=end){

        int mid = start + (end - start) /2;

        if(target == arr[mid]){

            return mid;

        }else if(target > arr[mid]){

            start = mid+1;

        }else{

            end = mid -1;

        }

    }

    return -1;

}

int main(){

    srand(time(0));

    long long  size;

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

    cin>>size;

    long arr[size];

    for(int i = 0 ; i < size ; i++){

        arr[i]=rand();

    }

    sort(arr,arr+size);

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

    //     cout<<arr[i]<<" "<<endl;

    // }

    long start = 0;

    long end = size-1;

    long index = rand() % size;

    long target = arr[index];

    cout<<"Target "<<target<<endl;

    auto start1 = high\_resolution\_clock::now();

    int result1 = LinearSearch(arr,size,target);

    auto end1 = high\_resolution\_clock::now();

    auto TimeofLinearSearch = duration\_cast<nanoseconds>(end1-start1);

    auto start2 = high\_resolution\_clock::now();

    int result2=BinarySearch(arr,size,target);

    auto end2 = high\_resolution\_clock::now();

    auto TimeofBinarySearch = duration\_cast<nanoseconds>(end2 - start2);

    auto start3 = high\_resolution\_clock::now();

    int result3 = RecursiveBinary(arr,size,target,start,end);

    auto end3 = high\_resolution\_clock::now();

    auto TimeofRecursiveBS = duration\_cast<nanoseconds> (end3 - start3);

    cout<<"Element found at index "<<result1<<" and time for Linear search is "<<TimeofLinearSearch.count()<<endl;

    cout<<"Element found at index "<<result2<<" and time for Binary search is "<<TimeofBinarySearch.count()<<endl;

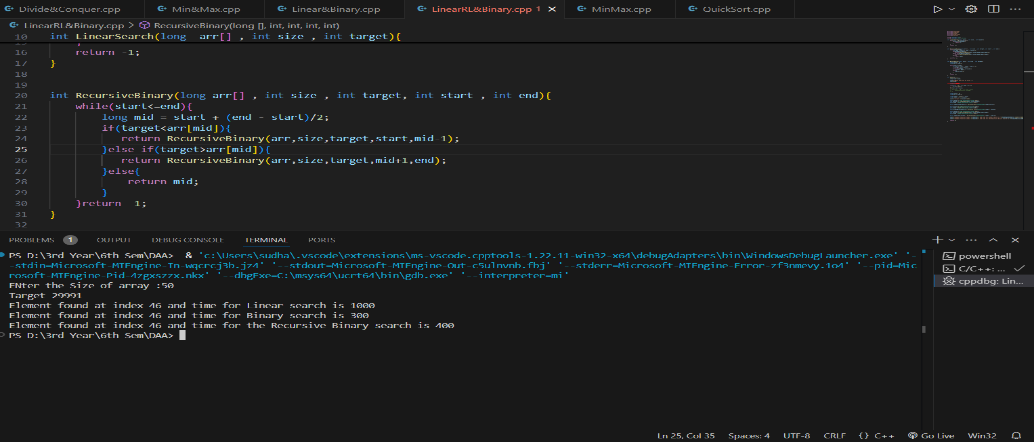
    cout<<"Element found at index "<<result3<<" and time for the Recursive Binary search is "<<TimeofRecursiveBS.count()<<endl;

    return 0;

}

**Output Screenshot:**

For 50 Element



For 500 Element

A screenshot of a computer program

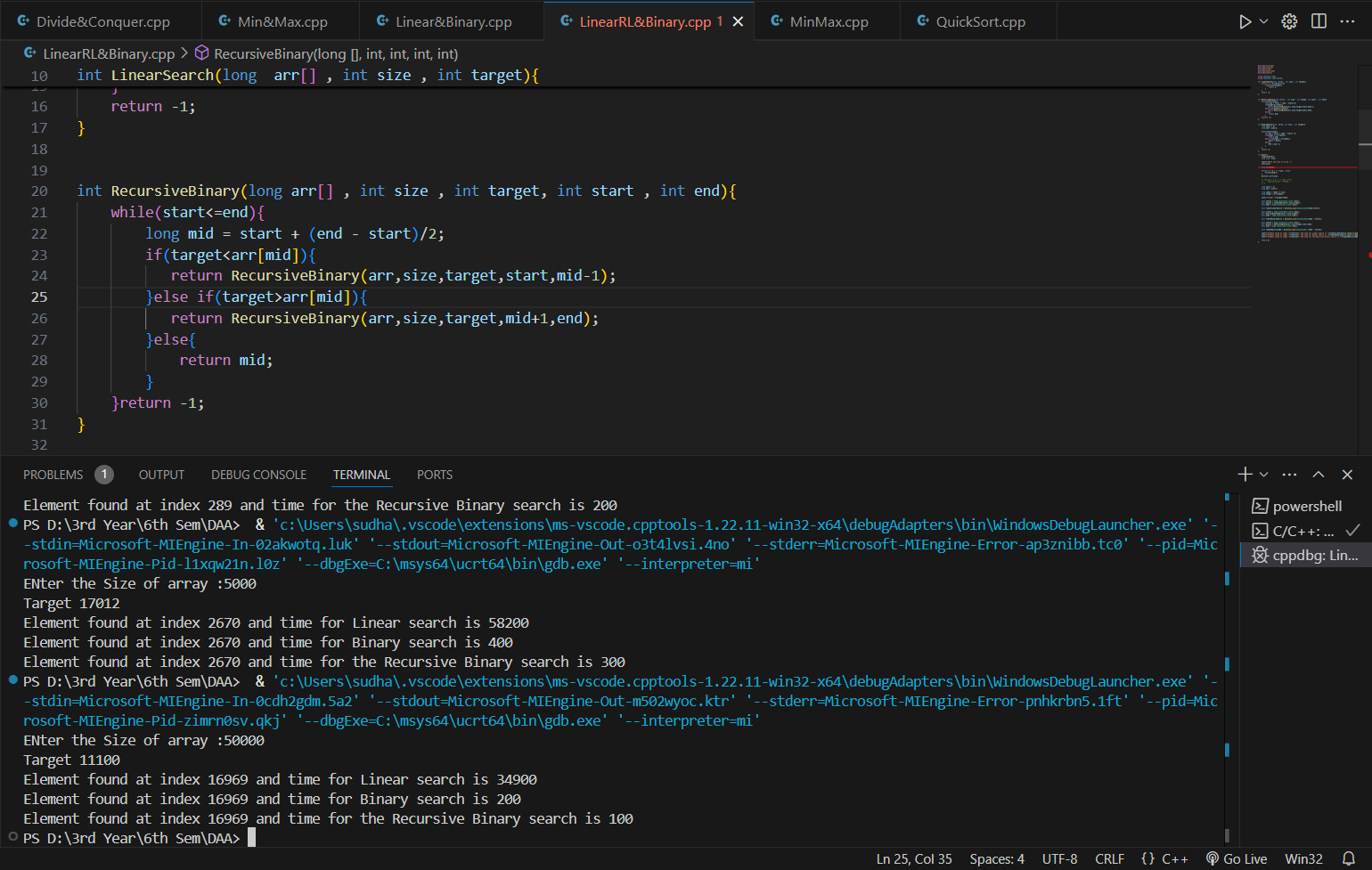
Description automatically generated

For 5000 Element

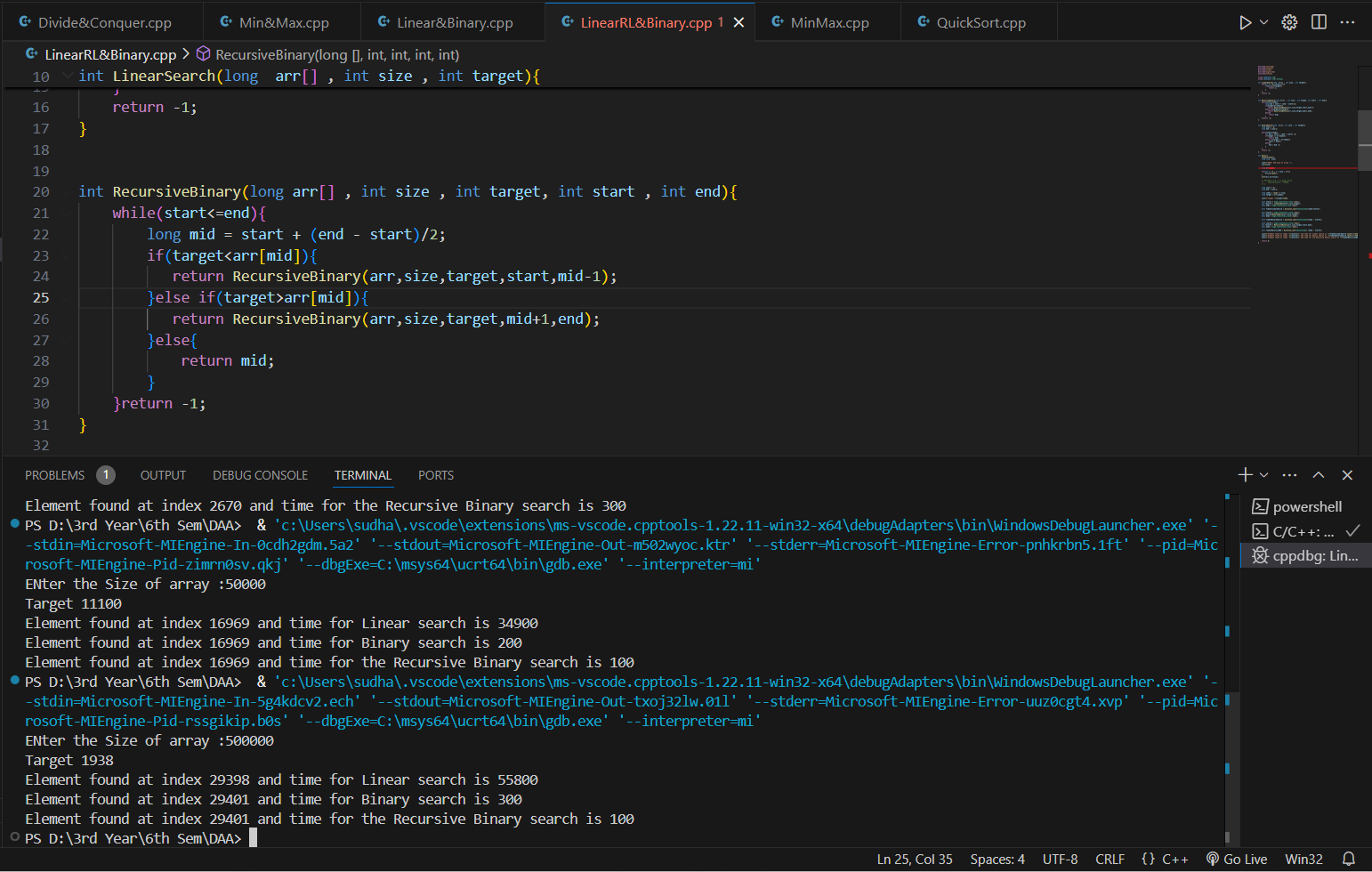
A screenshot of a computer program

Description automatically generated

For 50000 Element



For 500000 Element



**Tables:**

|  |  |  |
| --- | --- | --- |
| **Search Algorithm** | **Time Complexity** | **Space Complexity** |
| Linear Search | O(n) | O(1) |
| Binary Search | O(log n) | O(1) |
| Recursive Binary Search | O(log n) | |  | | --- | |  |  |  | | --- | | O(log n) | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Elements** | | | | |
| **Search Algorithm** | **50** | **500** | **5000** | **50000** | **500000** |
| Linear Search | 1000 | 6000 | 58200 | 34900 | 55800 |
| Binary Search | 300 | 200 | 400 | 200 | 300 |
| Recursive Binary Search | 400 | 200 | 300 | 100 | 100 |

**Time (Nanosecond)**

**Graph of Comparison :**

