Bansilal Ramnath Agarwal Charitable Trust's

Vishwakarma Institute of Information Technology

Department of Artificial Intelligence and Data Science

Name: Sahil Dilip Sonawane		
Class: TY	Division: C	Roll No: 373057
Semester: V	Academic Year: 2023-2024	
Subject Name & Code: Design and Analysis of Algorithm: ADUA31202		
Title of Assignment: Write a program to perform binary search on an unsorted random list of at least 5000 elements. The key element should be user input. Use the Divide & Conquer method to implement this program.		
Date of Performance: 04-08-2023	Date of Submission: 11-08-2023	

ASSIGNMENT NO. 1

Theory:

Binary search is a fast search algorithm with run-time complexity of O(log n). This search algorithm works on the principle of divide and conquer. For this algorithm to work properly, the data collection should be in the sorted form.
Binary search looks for a particular item by comparing the middle most item of the collection. If a match occurs, then the index of item is returned. If the middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item. Otherwise, the item is searched for in the sub-array to the right of the middle item. This process continues the sub-array as well until the size of the subarray reduces to zero.

How Binary search works?

1) **Divide and Conquer Principle:**

   Binary search is based on the "divide and conquer" approach, which means it breaks down a larger problem into smaller subproblems until a solution is found. In the case of binary search, the algorithm repeatedly divides the search space in half, eliminating half of the remaining elements at each step, thus significantly reducing the number of elements to search through.

2) **Preconditions:**
   1. The collection of data must be sorted in ascending (or descending) order. This is crucial because binary search relies on comparing elements to decide whether to search the left or right half.
   2. The data structure should allow for efficient random access to elements, such as arrays or lists with indexed access.

3) **Searching Process:**
   a. **Initialization:**
      1. Initially, you have the entire sorted collection as your search space.
      2. You maintain two pointers: one pointing to the start of the current search space and the other pointing to the end.
   b. **Comparison and Subdivision:**
      1. Calculate the middle index of the current search space: (start + end) / 2.
      2. Compare the element at the middle index to the target element you're searching for.
      3. If the middle element matches the target, you've found it and can return the index.
      4. If the middle element is greater than the target, this means the target must be in the left half, so you update the 'end' pointer to be the middle index minus one.
      5. If the middle element is less than the target, the target must be in the right half, so you update the 'start' pointer to be the middle index plus one.

   c. **Iteration or Termination:**

1. The process is repeated until the search space is reduced to zero (start becomes greater than end), indicating that the target element is not present in the collection.
2. If the target is found, its index is returned. If not, the algorithm terminates with a signal that the target is not in the collection.

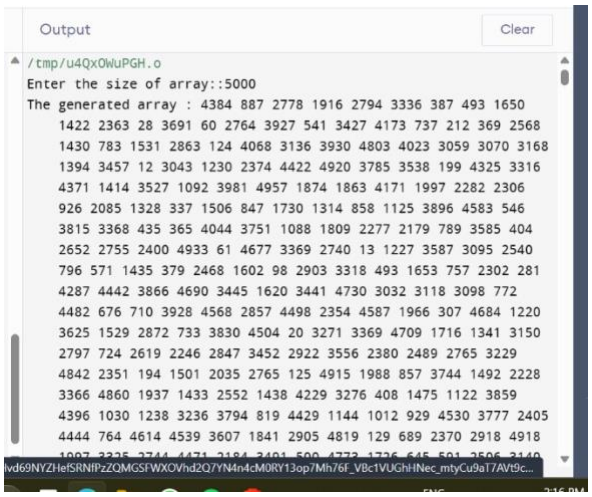
Time Complexity:

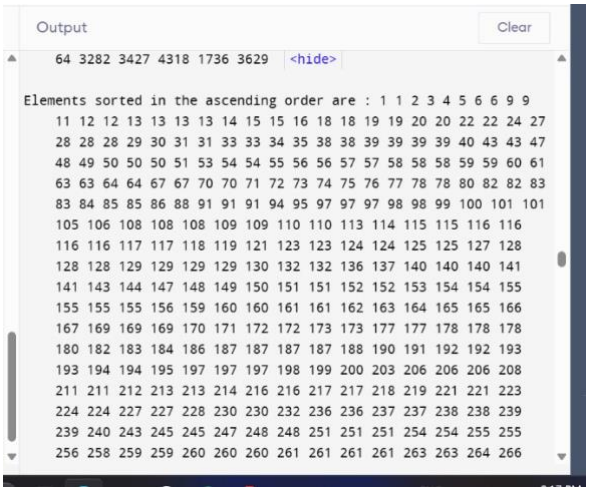
The key reason for binary search's efficiency is its logarithmic runtime complexity, which is O(log n). With each iteration, the search space is effectively halved, leading to rapid convergence even with large datasets. This is in stark contrast to linear search (O(n)), where each step eliminates only one element from consideration.

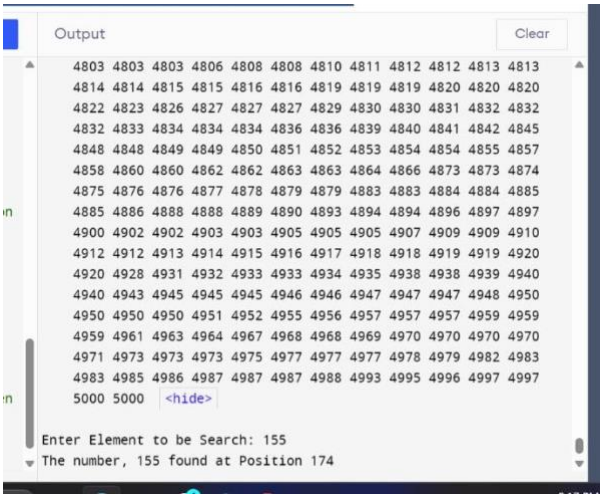
OUTPUT:

```
#include<iostream> using
namespace std; int main()
{
    int i, num, first, last, middle;
int x, y,size, temp;     int sz;

    cout<<"Enter the size of array::";
cin>>sz;     int randArray[sz];
int arr[5000];     for(int
i=0;i<sz;i++)
        randArray[i]= 1 + rand()%5000; //Generate number between 0 to
5999

    cout<<"The generated array : ";
for(int i=0;i<sz;i++)
cout<<randArray[i]<<" ";

    for(int x=0;x<sz;x++)
arr[x]=randArray[x];

    //Ascending order     for (x =
0; x < sz; x++){         for (y =
x; y < sz; y++){             if
(arr[x] > arr[y+1]){
temp = arr[x];
arr[x] = arr[y+1];
arr[y+1] = temp;
            }
        }
```

```
    }

    //Output
    cout <<"\n\nElements sorted in the ascending order are :
";     for (x = 1; x <= sz; x++){         cout << arr[x]<<" ";
    }

    cout<<"\n\nEnter Element to be Search:
";     cin>>num;     first = 0;     last =
sz-1;
    middle = (first+last)/2;
while(first <= last)
    {
if(arr[middle]<num)
first = middle+1;         else
if(arr[middle]==num)
        {
            cout<<"\nThe number, "<<num<<" found at Position
"<<middle;
            break;
        }
else
            last = middle-1;
middle = (first+last)/2;
    }
    if(first>last)
        cout<<"\nThe number, "<<num<<" is not found in given Array";
cout<<endl;     return 0;
}
```






Output
Clear
4803 4803 4803 4806 4808 4808 4810 4811 4812 4812 4813 4813
4814 4814 4815 4815 4816 4816 4819 4819 4819 4820 4820 4820
4822 4823 4826 4827 4827 4827 4829 4830 4830 4831 4832 4832
4832 4833 4834 4834 4834 4836 4836 4839 4840 4841 4842 4845
4848 4848 4849 4849 4850 4851 4852 4853 4854 4854 4855 4857
4858 4860 4860 4862 4862 4863 4863 4864 4866 4873 4873 4874
4875 4876 4876 4877 4878 4879 4879 4883 4883 4884 4884 4885
4885 4886 4888 4888 4889 4890 4893 4894 4894 4896 4897 4897
4900 4902 4902 4903 4903 4905 4905 4905 4907 4909 4909 4910
4912 4912 4913 4914 4915 4916 4917 4918 4918 4919 4919 4920
4920 4928 4931 4932 4933 4933 4934 4935 4938 4938 4939 4940
4940 4943 4945 4945 4945 4946 4946 4947 4947 4947 4948 4950
4950 4950 4950 4951 4952 4955 4956 4957 4957 4957 4959 4959
4959 4961 4963 4964 4967 4968 4968 4969 4970 4970 4970 4970
4971 4973 4973 4973 4975 4977 4977 4977 4978 4979 4982 4983
4983 4985 4986 4987 4987 4987 4988 4993 4995 4996 4997 4997
5000 5000 <hide>
Enter Element to be Search: 155
The number, 155 found at Position 174