# Assignment No. 1

## 1 Title:

Using Divide and Conquer Strategies design a function for Binary Search using C++/Java/Python/Scala.

## 2 Theory:

### 2.1 Divide and Conquer:

- Given a function to compute on an inputs the divide and conquer strategy suggest splitting the input into k distinct input sets ,such that 1¡k¡n yielding k sub problems.

- These sub problems must be solve and then method must be found to combine the sub solution into a solution of a whole.

- If the sub problems are relatively large then the divide and conquer strategy can be possibly reapplied.

- Often the sub problems are of the same type as the same original problem for which the reapplication of divide and conquer principle is naturally expressed by a recursive algorithm.

### 2.2 Binary Search:

- Binary search can be implemented using a binary decision tree.

- Example:Consider the following-
  Sorted array: L = [1, 3, 4, 6, 8, 9, 11]
  Target value: X = 4
  Compare X to 6. X is smaller. Repeat with L = [1, 3, 4].
  Compare X to 3. X is larger. Repeat with L = [4].

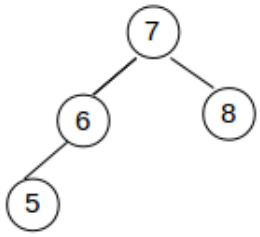
Compare X to 4. X equals 4, so the position is returned.

## 2.3 Applying divide and conquer approach on binary search tree:

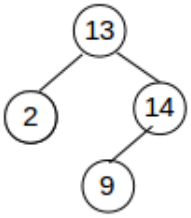
- If divide and conquer approach is use for generating the binary search tree, the input element set is to be divided into subset such that BST can be generated for each subset.

- The problem occurs when these BST's are generated from the subset are to be merged into a single BST consisting of all the input elements.

- Consider the following
  Let the input set
  I=7,6,5,8,13,2,14,9
  If we divide the input set into two parts then
  I1=7,6,5,8 I2=13,2,14,9
  then respective BST's are




BST1

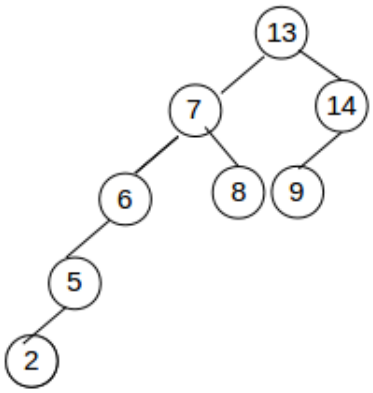



BST2

## 2.4 Proposal :

- In the proposed approach we divide the input into two parts such that all the elements smaller than the first element are in the first sub part and all those greater than the first element are in the second sub part.
  I=7,6,5,8,13,2,14,9

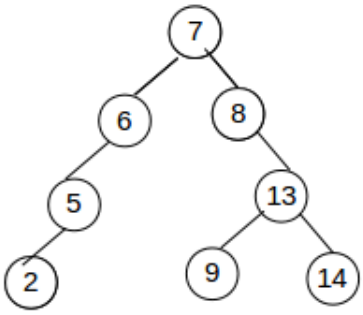
  I1=7,6,5,2 I2=7,8,13,14,9
  Also preserving the order in which they appear in I1.




BST

BST's thus formed from I1 and I2 will be now as follow


- Now we can easily merge the two BSTs as follow



# 3 Mathematical model:

- Inputa. Elements in ascending order
  b. Element to be searched

- Output  Element location

- Formula  :a. Mid= lower+upper/2
  b. a[mid+1]

- Algorithm-
  1. Start
  2. Binary-search (A, first, last)
  3. if first ¿ last // (i.e., nothing to search)
  4. return false
  5. otherwise
  6. middle ¡=(first+ last)
  7. if A[middle]matches key
  8. return true.
  9. otherwise.
  10. if A[middle]¿ key.
  11. return.
  12. Bin-search(A, first,middle -1, key).
  13. otherwise.
  14. return
  15. Bin-search(A, middle +1, last, key).
  16. Post: returns true if key is in A [first...last].
  17. Stop

# 4 Testing:

## 4.1 Blackbox Testing:

- Black-box testing is a method of software testing that examines the function- ability of an application without peering into its internal structures or workings.

- This method of test can be applied to virtually every level of software testing:unit, integration, system and acceptance

- Input:Sorted array of N elements, Key to be searched

- output:Position of key after searching Example shown in the above table

Table 1: black box testing

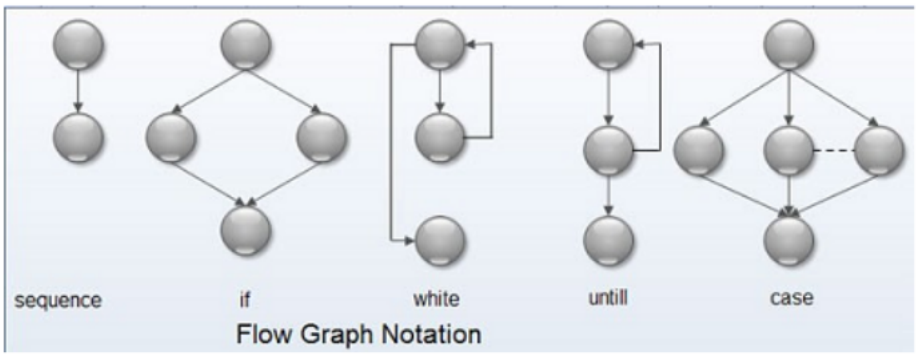
| Input array(T) | Key(key) | Output(Found,L) |
|---|---|---|
| 17 | 17 | true,1 |
| 17 | 0 | false,?? |
| 17,21,23,29 | 17 | true,1 |
| 9,16,18,30,31,41,45 | 45 | true,7 |
| 17,18,21,23,29,38,41 | 23 | true,4 |
| 17,18,21,23,29,33,38 | 21 | true,3 |
| 17,18,21,23,32 | 23 | true,4 |

## 4.2 White Box Testing:

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality(i.e. black-box testing). While developing test cases for white box testing it is understood that complete testing is impossible. In White Box testing we checkup to which extent the code is being executed, i.e. Covered. There are different kinds of coverage like, statement coverage, path coverage, etc. We will use one of the most popular technique i.e. Statement coverage. Statement coverage is a white box testing technique, which involves the execution of all the statements at least once in the source code. It is a metric, which is used to calculate and measure the number of statements in the source code which have been executed. For this, we will use Flow Graphs. Flow graphs are, Syntactic abstraction of source code Resembling to classical flow charts Forms the basis for white box test case generation principles.Conventions of flow graph notation, Sample Input: For integer array int A[] = 1,2,3,4,5,6,7,8,9,10; Function is passed following arguments: Binary Search(A, 1, 10, 7);
Output obtained: Entered second Entered third Entered first 6
The underlined nodes are the ones being tested. The above output shows that every test region is covered for given input.

## 4.3 Positive/Negative Testing:

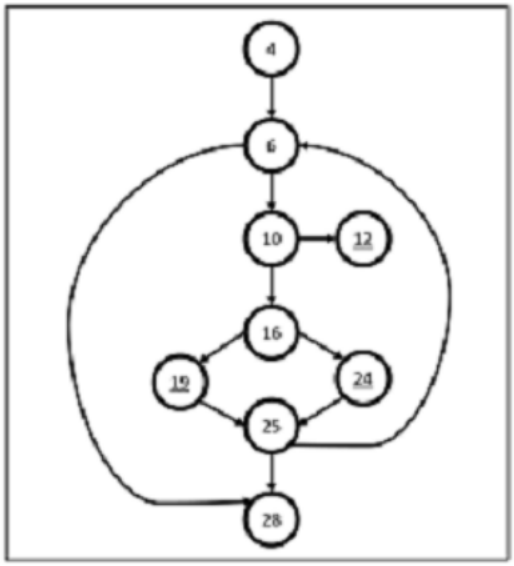
- Position Testing :The element entered in ascending order then correct location of key will be found



Flow Graph Notation

```
1  // Returns location of key, or -1 if not found
2  int BinarySearch(int A[], int l, int r, int key)
3  {
4      int m;
5
6      while( l <= r )
7      {
8          m = l + (r-l)/2;
9
10         if( A[m] == key ) // first
11             {
12                 cout<<"Entered first"<<endl;
13                 return m;
14             }
15
16         if( A[m] < key ) // second
17             {
18                 cout<<"Entered first"<<endl;
19                 l = m + 1;
20             }
21         else
22             {
23                 cout<<"Entered first"<<endl;
24                 r = m - 1;   // third
25             }
26     }
27
28     return -1;
29 }
```

Function



Flow Graph Notation

Example- Input sequence(17) key (17) output(true ,1)

- Negative Testing :The element not entered in ascending order then wrong location of key will be found
  Example- Input sequence(17) key (0) output(false,??)

# 5 Conclusion:

From this experiment, we have successfully ,implemented binary search using divide and conquer strategies.

| Roll No. | Name of Student | Date of Performance | Date of Submission | Sign. |
|---|---|---|---|---|
| BECOC357 | Sunny Shah | 22 / 06 / 2017 | 29 / 06 / 2017 | |