Experiment No. 10: Binary Search Method

Name : Vaibhav Tatkare
Roll No. / Div : 52 / Comps3

Aim : Implementation of Binary Search Method

Objective: 1) Understand how to implement Binary Search algorithm.

Theory:

The improvement to searching method to reduce the amount of work can be done using binary searching. Binary searching is more efficient than linear searching if an array to be searched is in sorted manner.

Here an key item to be searched is compared with the item at middle of array. If they are equal search is completed. If the middle element is greater than key item searching proceeds with left sub array. Similarly, if middle element is less than key item than searching proceeds with right sub array and so on till the element is found.

For large arrays, this method is superior to sequential searching.

Algorithm

```
Algorithm : FIND(arr, x,  first,  last)
  if (first > last)then
    return -1
  End if
  mid = (first + last) / 2
  if (arr[mid] = x)
    return mid
  End if
  if (arr[mid] < x)
    return find(arr, x, mid+1, last)
  End if
```

```
  return find(arr, x, first, mid-1)
```

Code:

```
#include <stdio.h>
#include <stdlib.h>
struct node {
  int data;
  struct node *leftChild, *rightChild;
};
struct node *root = NULL;
struct node *newNode(int item){
  struct node *temp = (struct node *)malloc(sizeof(struct node));
  temp->data = item;
  temp->leftChild = temp->rightChild = NULL;
  return temp;
}
void insert(int data){
  struct node *tempNode = (struct node*) malloc(sizeof(struct node));
  struct node *current;
  struct node *parent;
  tempNode->data = data;
  tempNode->leftChild = NULL;
  tempNode->rightChild = NULL;
```

```
//if tree is empty
if(root == NULL) {
   root = tempNode;
} else {
   current = root;
   parent = NULL;
   while(1) {
      parent = current;

      //go to left of the tree
      if(data < parent->data) {
         current = current->leftChild;

         //insert to the left
         if(current == NULL) {
            parent->leftChild = tempNode;
            return;
         }
      }//go to right of the tree
      else {
         current = current->rightChild;
```

```
            //insert to the right
            if(current == NULL) {
               parent->rightChild = tempNode;
               return;
            }
         }
      }
   }
}
struct node* search(int data){
   struct node *current = root;
   printf("\nVisiting elements: ");
   while(current->data != data) {
      if(current != NULL) {
         printf("%d ",current->data);

         //go to left tree
         if(current->data > data) {
            current = current->leftChild;
         }//else go to right tree
         else {
            current = current->rightChild;
         }
```

```
      //not found
      if(current == NULL) {
        return NULL;
      }
    }
  }
  return current;
}
void printTree(struct node* Node){
  if(Node == NULL)
    return;
  printTree(Node->leftChild);
  printf(" --%d", Node->data);
  printTree(Node->rightChild);
}
int main(){
  insert(10);
  insert(14);
  insert(19);
  insert(26);
  insert(27);
  insert(31);
```

```
  insert(33);
  insert(35);
  insert(42);
  insert(44);
  printf("Insertion done\n");
  printTree(root);
  struct node* k;
  k = search(35);
  if(k != NULL)
    printf("\nElement %d found", k->data);
  else
    printf("\nElement not found");
  return 0;
}
```

Output:

```
Insertion done
 --10 --14 --19 --26 --27 --31 --33 --35 --42 --44
Visiting elements: 10 14 19 26 27 31 33
Element 35 found
--------------------------------
Process exited after 0.008435 seconds with return value 0
Press any key to continue . . .
```

Conclusion:

- Binary Search is a search algorithm that allows us to find elements in a sorted list.
- Binary Search is efficient because it continually divides the search space in half, until it finds the element or only one element remains in the list to be tested.
- Binary Search has a time complexity of O(1) in the best case, and O(log N) in the worst case.
- Binary Search has a space complexity of O(1) in the iterative method, while it is O(log N) in the case of the recursive method.