Unit 8: Searching

Introduction- Searching

- Searching is defined as a process of trying to find the required data in any list or collection of data.
- It is not necessary that the data item we are searching for must be present in the list.
- If the searched item is present in the list then the searching algorithm (or program) can find that data item, in which case we say that the search is successful, but if the searched item is not present in the list, then it cannot be found and we say that the search is unsuccessful.

Sequential Search (Linear Search)

- A sequential search algorithm, checks every value of a data list one at a time in sequence until a match is found.
- It is the simplest technique to find out an element in an unordered list.
- In this technique, the value of key is compared with the first element of the list, if it gets matched then the search is terminated with success (and the corresponding location is returned). Otherwise next element from the list is fetched and compared with the key. This process is continued till the key is found or the list is completely traversed.
- If the key is not found in entire list then the search is unsuccessful.

C Code for Sequential Search

- The following C function shows the sequential search operation in an array of integers:

```
int linearSearch(int arr[ ], int n, int key)    //arr is array of items & n is number of items in list
{                                               //key is the element to be searched
    int i, location;
    for(i=0; i<n; i++)
    {
            if(arr[i]==key)                     //key found in ith location
            {
                    printf("Search is successful\n");
                    location = i+1;
                    return location;
            }
    }                                           //whole list traversal is completed here
    printf("Key not found\n");
    location=-1;
    return location;
}
```

Complexity Analysis

- If there are ***n*** items in our collection (whether it is stored as an array or as a linked list), then in the worst case, when there is no item in the collection with the desired key, then ***n*** comparisons of the key with items in the collection will have to be made.
- In the case of searching, the dominant operation is the comparison operation. Since the search requires ***n*** comparisons in the worst case, we say that, this is a ***O(n)*** algorithm.
- In the best case the first comparison returns a match hence requires a single comparison and is ***O(1)***.

Model Question (2008)

- Illustrate the sequential search with suitable example.

Binary Search

- When the size of the list is very large, searching an item by linear search becomes very tedious and time consuming.
- In such case we need a searching algorithm which is more efficient and fast. Binary search helps to find out an element in the list faster and efficiently by reducing the searching domain by half in each pass.
- Binary search is a technique for searching an ordered list in which we first check the middle item and based on that comparison discard half the data. The same procedure is then applied to the remaining half until a match is found or there are no more items left.

Binary Search…

- In this method to search an element we compare it with the element present at the centre of the list, if it matches then search is successful otherwise the list is divided into two halves.
- One half contains elements from first position to center position and another half from (center+1)th position to last position.
- For a list in ascending order the first half contains the elements smaller than element in middle position and the second half contains the elements greater than element in middle position (and vice-versa if the list is in descending order).
- The search is continued with one of the halves depending on whether the search key is greater than or smaller than the center element.
- The binary search can be implemented in two ways:
  - Iterative Implementation
  - Recursive Implementation

Iterative Implementation of Binary Search

```
int binarySearch(int arr[ ], int low, int high, int key)
{
    int mid;
    while ( high >= low)
     {
      mid = (low+high)/2;
             if (key == arr[mid])
             {
                           printf(“Search is successful\n”);
                           return mid+1;
             }
             else if(key < arr[mid])
                           high = mid-1;
             else
                           low = mid+1;
     }
    printf(“Item not found\n”);
    return -1;
}
```

Recursive Implementation of Binary Search

```
int binarySearch( int arr[ ], int low, int high, int key)
{
     int mid;
     if(  low>high)
     {
     printf("Search is unsuccessful\n");
     return -1;
     }
     mid = (high+low)/2;
     if (key>arr[mid])
               binarySearch(arr,mid+1,high,key);
     else if (key<arr[mid])
               binarySearch(arr,low,mid-1,key);
     else
     {
               printf("Search is successful\n");
               return mid+1;
     }

}
```

Illustration

- Suppose that we are trying to find an element 33 in the list given below.

[Note that to perform binary search the list must be in sorted order]

5	12	25	33	37	48	57	86	92	98
0	1	2	3	4	5	6	7	8	9

- Initially the lower limit of array is 0 and higher limit is 9, hence according to the binary search algorithm the value of *mid = (0 + 9)/2 = 4.5*. Since only the integer value is taken as mid, so *mid=4*. Now, in the array index 4, there is value 37. The key value given is 33, so in this case key is less than 37. Thus the value of high is set as *mid-1* i.e. *high = (4-1) = 3*.

Illustration…

- Next time, the value of *mid = (3+0)/2 = 1*, so value of item at mid position is 12.
- The key value 33 is greater than 12. Thus the value of low is set as *mid + 1* i.e. *low* = 2 and the process is repeated again.
- This process is repeated until we get the desired key in the *mid* position or the *low* gets greater than *high*.

Iteration No.	low	high	mid	Remarks
0	0	9	4	key<arr[mid]
1	0	3	1	key>arr[mid]
2	2	3	2	key>arr[mid]
3	3	3	3	key=a[mid]

- Here in this case the searching terminates with success after finding the key in fourth position of the list.

Complexity Analysis

- Let there are $\mathbf{n=2^m}$ items in the list (hence $\mathbf{m = \log_2 n}$).
- In the best case if the required element is in the middle position of the list, the first comparison returns the key resulting complexity of ***O(1)***.
- In the worst case if the desired key is not in the list we have to keep on dividing the list by half until there is only one item present in the list and we make only one comparison in each sub-list. Thus,

$$\begin{aligned} \text{Total no. of comparisons} &= 1+1+1+\ldots\ldots\ldots\ldots \text{ up to } \mathbf{m} \text{ terms} \\ &= m \\ &= \log_2 n \end{aligned}$$

- So the complexity of binary search in the worst case is ***O(logn).***

TU Exam Question (2065)

- Explain the binary searching.

TU Exam Question (2066)

- Trace Binary Search algorithm for the data:

  21, 36, 56, 79, 101, 123, 142, 203

  and search for the values 123 and 153.

Hashing

Model Question (2008)

- What is hashing? Explain the terms hash collision.

TU Exam Question (2066)

- Write a short note on hash function.