TUTORIAL-4

SEQUENTIAL SEARCH:

If the input array is unsorted only linear sequential search can be used.

For a sorted array the search is more efficient because the search can be terminated when a record with a key is greater than the search key.

All cases the complexity of the algorithm is O(n).

JUMP SEARCH:

Instead of searching every element, this algorithm jumps of length k where k =vn.

Input array must be sorted.

All cases the complexity of the algorithm is O(vn).

BINARY SEARCH:

If the set of records are large but sorted in ascending order of key values, then total search time can be reduced significantly using a Binary search.

Binary search implements the “Divide and Conquer” method.

All cases the complexity of the algorithm is O(log n).

INTERPOLATION SEARCH:

It is a modified algorithm of Binary search reducing the complexity.

Best/Average case complexity is O(log log n)) which will occur when elements of the array are fairly uniformly distributed over their interval eg. . [3,5,8,10,13,15]

Worst case complexity is O(n) which will occur when the elements are not fairly uniformly distributed over their interval eg. [3,5,8,10,13,100]

BINARY SEARCH TREE:

Binary Tree Search is dynamic which allows for inserting and deleting data items.

In Binary Search, binary trees are used only to describe the sequence of comparisons and to facilitate performance analysis but in Binary Tree Search, this data structure is actually constructed and used for the search.

The cost of each operation is proportional to the number of nodes accessed during the operation.

Best case search time (find operation) is O(log n) which occurs when the tree is perfectly balanced.

Worst case search time (find operation) is O(n) which occurs when the tree is heavily unbalanced (all the nodes are on the path to the deepest node).

Average case search time (find operation) is O(log n) which occurs when the tree is created by random insert sequences but with no remove operations.

Example: Search for the key 20 in the following array using Binary Search, Interpolation Search and Jump Search.

int[] a = {0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30}

Binary Search:

low=0, high=15, middle=(0+15)/2=7

a[middle] = 14 which is less than key(20).

low=middle+1=7+1=8, high=15, middle = (8+15)/2 = 11

a[middle] = 22 which is greater than key.

low = 8, high = middle-1=11-1=10, middle= (8+10)=9

a[middle]=18 which is less than key.

low= middle+1=9+1=10, high=10, middle = (10+10)/2 = 10

a[middle] = 20 which is equal to key(20).

So key found at index at 10.

Interpolation Search:

middle = low + (high - low)*((key – a[low]) / (a[high] – a[low]))

middle = 0 + ceil(15 – 0)* ((20 – 0) / (30 – 0)) = 0 +10 =10

a[middle] = 20 which is equal to key.

So key found at index 10.

Jump Search:

Here k =vn =v16 = 4 (n = length of array)
This algorithm will look for 4^{th} element, 8^{th} element, 12^{th} element and so on…..
4^{th} element = 6 which is less than key (20).
8^{th} element = 14 which is less than key.
12^{th} element = 22 which is greater than key.

Now we are sure that key is in between index 8 and 11. So applying sequential search starting from index-8 we can find out key is at index 10.

Example: Proof that the average number of comparisons to find a key in the array [0,1,2,3,……..,n-2,n-1] by using interpolation search is O(1).

Let us assume, key = i.

middle = low + (high - low)*((key – a[low]) / (a[high] – a[low]))
= 0 +((n-1)-0)*((i – 0) / ((n-1) – 0)) = i.

a[middle] = a[i] = i which means that the key can be found in one step ie. if key = 5 then the key can be found at index 5.
Hence the complexity is O(1).