**Part II. Data Structure and Algorithms**

**Question 1. Assume that you have an SORTED array of records. Assume that the length of the array (n) is known. Give TWO different methods to SEARCH for a specific value in this array. You can use English or pseudo-code for your algorithm. What is the time complexity for each algorithm and why?**

Let’s assume that we have a sorted array of records of array length 11 [0-10]

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

Here, we can perform two methods of searches 1) Linear search 2) Binary search

**Method 1: Linear Search.**

Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.

Let’s say that we want to search an element with value ‘6’ in the above given array. Then the following steps of an algorithm should be followed.

Step 1: Find the length of the given array. The length of the array in the above example is 11.

Step 2: Start with the 0th index and find if the value is equal to ‘6’. (i=6)

Step 3: If the value is equal to ‘6’, return the position of the index.

Step 4: If the value of the index is not ‘6’, then iterate to the next index.(i+1=6)

Step 5: Repeat steps 3 and 4 till we find an index with value ‘6’. The above iteration will repeat until we reach the end of the array (maximum of 11 iterations – the length of the array).

**Time-Complexity:** O(n). The reason being, in the above array, let us say we want to search for an element with value 10 or a value that is not present in the given array, we iterate n times with n being the length of the array.

**Method 2: Binary Search**

Binary search is a fast search algorithm. For a binary search to work, it is mandatory for the target array to be sorted. 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 on the sub-array as well until the size of the sub array reduces to zero.

Step 1: In binary search, first of all we know that it works on sorted arrays.

Step 2: keep three pointers for three variables that will contain the indices of three different places i.e lower mid and upper limit of the array

Step 3 : First determine which element is to be found. Then take the midpoint of the array. If the array is even then [(0+upper limit)/2]=floating number (eg 7.5,then it will take 7 as the midpoint).

Step 4: Check if the value to be searched and the value in middle index are equal. If the values are equal, return the index.

Step 5: If the value to be searched (here its ‘8’) is greater than the value in the middle index, then find the middle index between 6 and 10 and compare the value to the value to be searched.

Step 6: If the value to be searched is lesser than the value in the middle index, then find the middle index between 0 and 4 and compare the value to the value to be searched.

Step 7 : Repeat steps 5 and 6 till we find an index with value ‘8’.

(In step 5/step 6 , initialize the mid-point as new lower limit. Then mid =[(new low + high)/2].again compare the index value of new mid-point with the target. If the values are not equal, again we will slice off the array by neglecting the redundant elements. This step will keep on until we come down to the target is found)

**Time-Complexity:** O(logn). The reason being, in the above array, we are reducing the search range by half for every iteration.

**Question 2. Assume that you have a linked list of records. Assume that you have a head, a current, and a tail pointer. Write an algorithm that swaps the data in the current node and the node after it. You can use pseudo-code, English or drawing to describe your solution.**

Let’s assume to have head, current and a tail pointer.

A-------> B------->C------->D------->E. Here “A” be the head , “C” be the current and “E” be the tail node.

Step 1: Create a temporary node (F)

Step 2: store the value of current node in the temporary node (C to F)

Step 3: Assign the value of the next node to the current node (D to C)

Step 4: Assign the value of the next node from the temp variable.(F to D)

Pseudo code:

var temp = current.value;

current.value = current.next.value;

current.next.value = temp;

The modified linked list will look like A-------->B-------->D-------->C---------->E

**Question 3. Assume that you have a linked list of records. Assume that you have a head, a current, and a tail pointer. Write an algorithm that DELETES the node BEFORE the current node. You can use pseudo-code, English or drawing to describe your solution. ( this was, and remains to be, a popular technical interview question)**

 Consider a linked list as below.

A-------->B-------->D-------->C---------->E-------->F with A as head, D as current node and F as the tail node.

Pseudo-code :

1)       If head == current, then return

2)       If head. next value == current.value

               Then head = head.next

                then return

3)       Set start = head

4)        If start.next.next.value == current.value

                Then start.next = start.next.next

          Else

                Start = start.next

5)       Repeat step 4 till the tail is reached