Technical Assignment 1

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?

Answer :

Binary Search Method

Binary search method is a fast method of searching. It works on divide and conquer technique and has a runtime complexity of **O(log n).**

The array should be a sorted array is the prerequisite for Binary Search. By cutting the problem set in half at every step, we can say that to get from n elements to single element is O(log n) steps.

- We are given a sorted Array whose length is n. We assume the sorted array to be in ascending order.
- Now, to find a particular element or a specific value from the given sorted array, we will consider X to be the element or the value to be searched.
- We consider the lower bound to be 1 and upper bound to be 'n' as the size of the array is given to be 'n'. For simplicity, we will assign the terms 'high' and 'low' for upper bound and lower bound respectively.

$$low = 1$$

$$high = n$$

- First we check if the $high < low$. If that is the condition, then X would not exist and the program would terminate there.
- But if the above condition is false, then further steps are followed.
- We then find the mid by using the following formula :

$$mid = low + (high - low)/2$$

- The position of the array element would be used and not the actual value of the data element in place of $high$ and low in the above formula.
- Now, once we get the mid, we will compare it with the element to be found X.
- We check the condition whether
  - $X > mid$. If so, then we set $low = mid + 1$.
  - $X < mid$. If so, then we set $high = mid - 1$.
  - $X = mid$. If so, then we come to the conclusion that X is found at the mid.
- We follow the same procedure till we arrive at X.

We would consider an example to find a value in an sorted array using Binary Search Method.

Consider the following Sorted Array A.

Array A	10	15	20	25	30	35	40	45	50	55
Position	0	1	2	3	4	5	6	7	8	9

- Say that we have to find 35. Hence $X = 35$.
- Here, $low = 0$ and $high = 9$.
- According to the explanation above, we find the mid.

$$mid = low + (high - low)/2$$

$$mid = 0 + (9 - 0)/2$$

$$mid = 4.5$$

- So we consider element at position 4 to be the mid of the array. It means position number 4. Element at location 4 is 30.

Array A	10	15	20	25	**30**	**35**	40	45	50	55
Position	0	1	2	3	4	5	6	7	8	9

- We find that element at position 4 is not equal to X. As the value of element at position 4 is less than X, we set $low = mid + 1$. Hence, new $low = 4 + 1 = 5$.
- Now again using the formula

$$mid = low + (high - low)/2$$
$$mid = 5 + (9 - 5)/2$$
$$mid = 7$$

- We now consider element at position 7 to be the mid of the array.

Array A	10	15	20	25	30	**35**	40	**45**	50	55
Position	0	1	2	3	4	5	6	7	8	9

- We can see that the element at position 7 is not yet equal to X. As the value of element at position 7 is greater than X, we set $high = mid - 1$. Hence, new $high = 7 - 1 = 6$.
- Now again using the formula

$$mid = low + (high - low)/2$$
$$mid = 5 + (6 - 5)/2$$
$$mid = 5.5$$

- Now this time the mid is at 5. And this is the same position for which we were searching. It means that $X = mid$.

Array A	10	15	20	25	30	**35**	40	45	50	55
Position	0	1	2	3	4	5	6	7	8	9

- We conclude here that X is found at mid. Hence the searching terminates here as we have reached the element we wanted to find.

Linear Search Method

Linear Search is the most simple search algorithm which searches for the element to be searched sequentially. The time complexity is **O(n).**

A linear search works by looking at each element in a list of data until it either finds the target or reaches the end. This results in O(n) performance on a given list.

- We are given a sorted Array whose length is n. We assume the sorted array to be in ascending order.
- Now, to find a particular element or a specific value from the given sorted array, we will consider X to be the element or the value to be searched.

Consider the following sorted Array A.

Array A	10	15	20	25	30	**35**	40	45	50	55

- We set iterator i for iterations. Initially we set $i = 1$.
- First we check whether the value of $i > n$. If that is so, then the program terminates because it means that the number of elements in the array n are less than the value of i.
- But if the above condition is false, then another condition is checked that is $A[i] = X$. This condition checks whether the value of the element where the iterator i points is equal to the value of the element to be searched X. If this condition is true then directly the value of X is printed and the program terminates.
- But if $A[i]$ is not equal to X then the iterator i increments by 1 and points to the next element in the array. This step repeats until the condition $A[i] = X$ is satisfied.

Step 1 :

Array A	**10**	15	20	25	30	**35**	40	45	50	55
Iterator i	i									

Step 2 :

Array A	10	**15**	20	25	30	**35**	40	45	50	55
Iterator i		i								

Step 3 :

Array A	10	15	**20**	25	30	**35**	40	45	50	55
Iterator i			i							

Step 4 :

Array A	10	15	20	**25**	30	**35**	40	45	50	55
Iterator i				i						

Step 5 :

Array A	10	15	20	25	**30**	**35**	40	45	50	55
Iterator i					i					

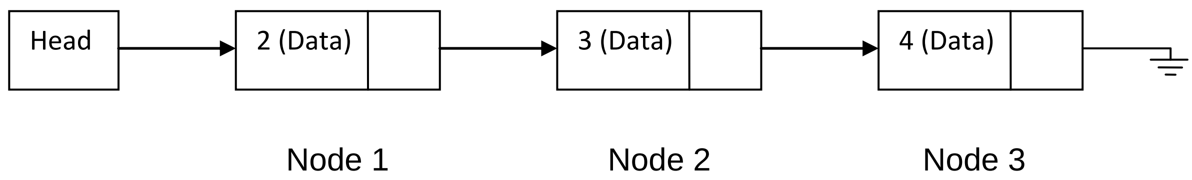
Step 6 :

Array A	10	15	20	25	30	**35**	40	45	50	55
Iterator i						i				

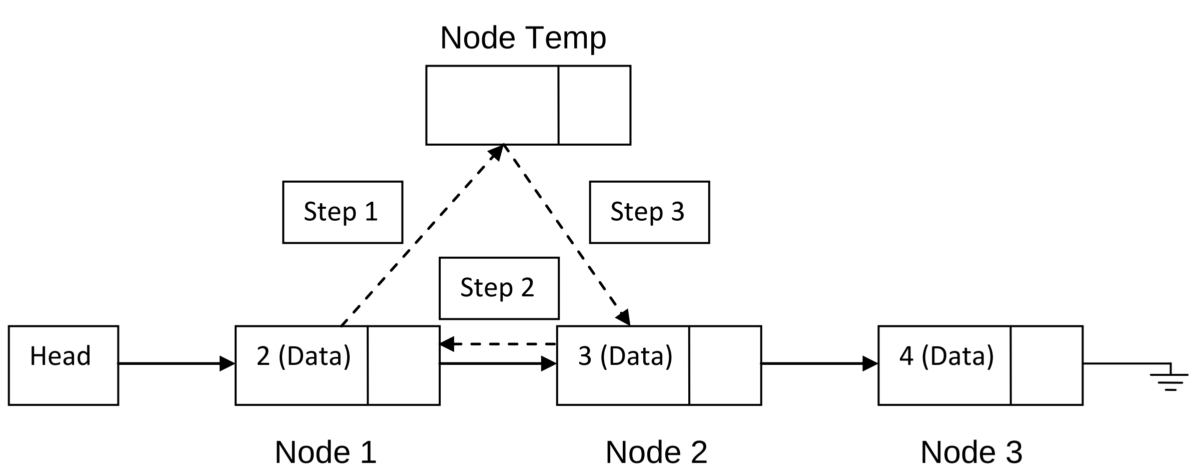
Element found.

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.

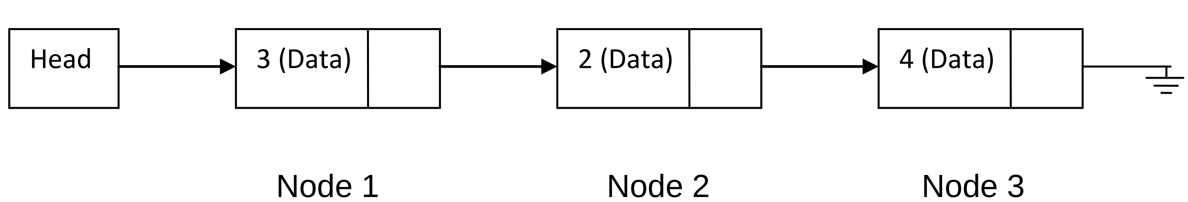
Answer :



- The figure above shows a simple Linked List. We have three nodes in total - Node 1, Node 2 and Node 3.
- The Head points to the Node 1.
- Consider Node 1 to be the current node. So swapping of data has to be done between Node 1 and Node 2.
- We now take into consideration another node called a Node Temp which would be a temporary node used for swapping data.
- Data can be swapped in the way described below.
    - Data from Node 1 is put into Node Temp.
    - Data from Node 2 is put into Node 1.
    - Data from Node Temp is put into Node 2.
- The swapping logic is shown in the figure below.

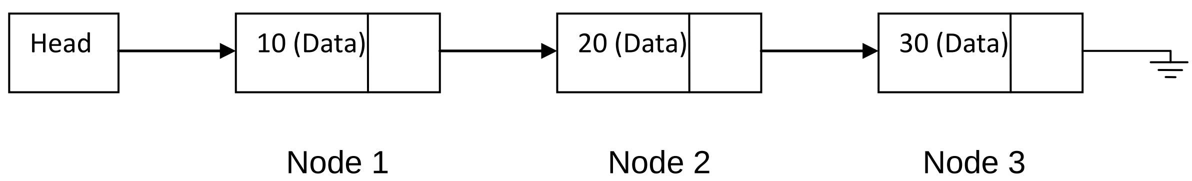


- After swapping the linked list looks like the one shown below.

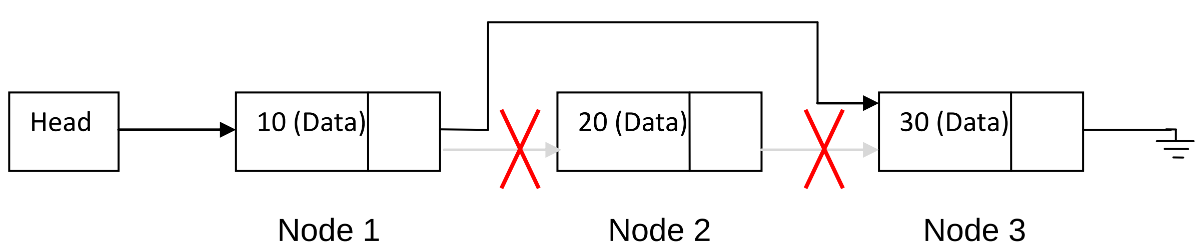


<u>Question 3.</u> Assume that you have a linked list of records. Assume that you have a **head**, a **current**, and a **tail** pointer. Write an algorithm that **DELETES the node BEFORE the current node**. You can use pseudo-code, English or drawing to describe your solution.(this was, and remains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

<u>Answer :</u>



- The figure above shows a simple Linked List. We have three nodes in total - Node 1, Node 2 and Node 3.
- The Head points to the Node 1.
- Consider Node 3 to be the current node.
- According to the given condition, we have to delete the node before the current node.
- So in the linked list shown above, we have to delete Node 2.
- Deleting Node 2 means storing the reference of Node 3 in Node 1 instead of Node 2 and removing the link or the relation between Node 2 and Node 3.
- This can be shown as follows.



- After the deletion of the Node 2, the linked list looks as follows.