Team Members

Kabilan Senapathy ks879

Manoj Nagarajan mn393

Srijan Deo sd759

Sirisha Bojjireddy sb2423

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?

Linear Search

1) Traverse the array using a for loop.
2) In every iteration, compare the target value with the current value of the array.
    a) If the values match, return the current index of the array.
    b) If the values do not match, move on to the next array element.
3) If no match is found, return -1

Time Complexity
Best Case - $\Theta(1)$
The number of operations in the best case is constant (not dependent on n). So time complexity in the best case would be $\Theta(1)$

Worst Case - $\Theta(n)$.
The worst case happens when the element to be searched (target value) is not present in the array. When x is not present, search function compares it with all the elements of array one by one. Therefore, the worst case time complexity of linear search would be $\Theta(n)$.

Binary Search

1) Start with the middle element:
    a) If the target value is equal to the middle element of the array, then return the index of the middle element.
    b) If not, then compare the middle element with the target value.
    c) If the target value is greater than the number in the middle index, then pick the elements to the right of the middle index, and start with Step 1.
    d) If the target value is less than the number in the middle index, then pick the elements to the left of the middle index, and start with Step 1.
2) When a match is found, return the index of the element matched.
3) If no match is found, then return -1

Time Complexity
Best Case - O(1)
The element is found at the beginning of the loop, where middle element = target value.

Worst Case - O(log n)
Since binary search is a divide and conquer method to find a number, log N represents the number of times the array is being divided.

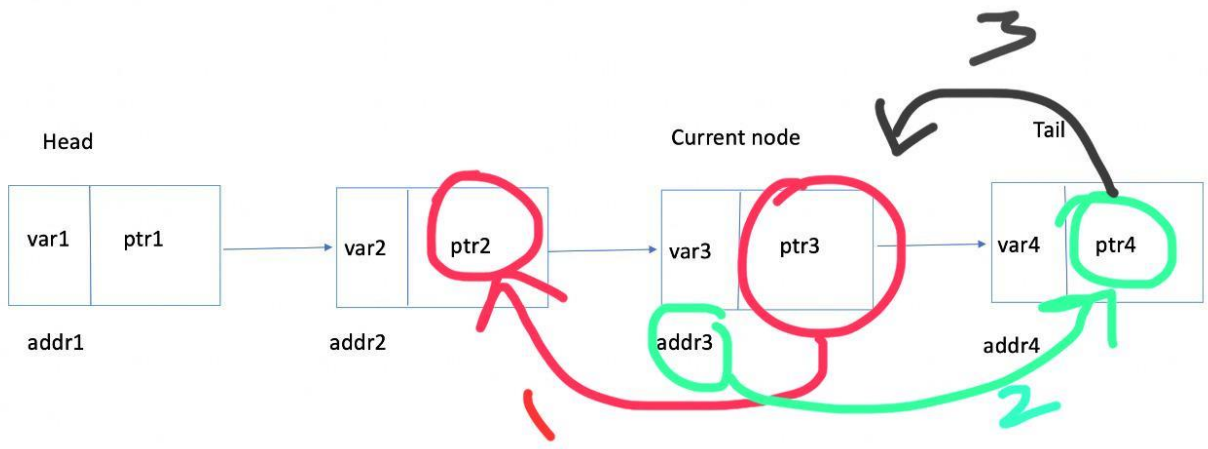
Question 2. Assume that you have a linked list of records. Assume that you have a **head**, a **current**, and a **tail** pointer. Write an algorithm that **swaps the data in the current node and the node after it**. You can use pseudo-code, English or drawing to describe your solution.

Consider next as a pointer pointing to the address of the next node.
Step 1 -> Find the next node

a) Consider a temp node
b) While (temp IS NOT NULL && temp->next = current)
   {
   temp=temp->next
   }

Step 2 -> Copy the pointer value of the current node to the previous node's pointer value.
Step 3 -> Copy the pointer of the next node to the pointer value of the current node
Step 4 -> Change the pointer value of the next node with the address of the current node.

For example,



step 1 -> ptr2=ptr3
step 2 -> ptr4=addr3
step 3 -> ptr3=ptr4

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)

Deletion in linked list simply means just unlink the node from the list.

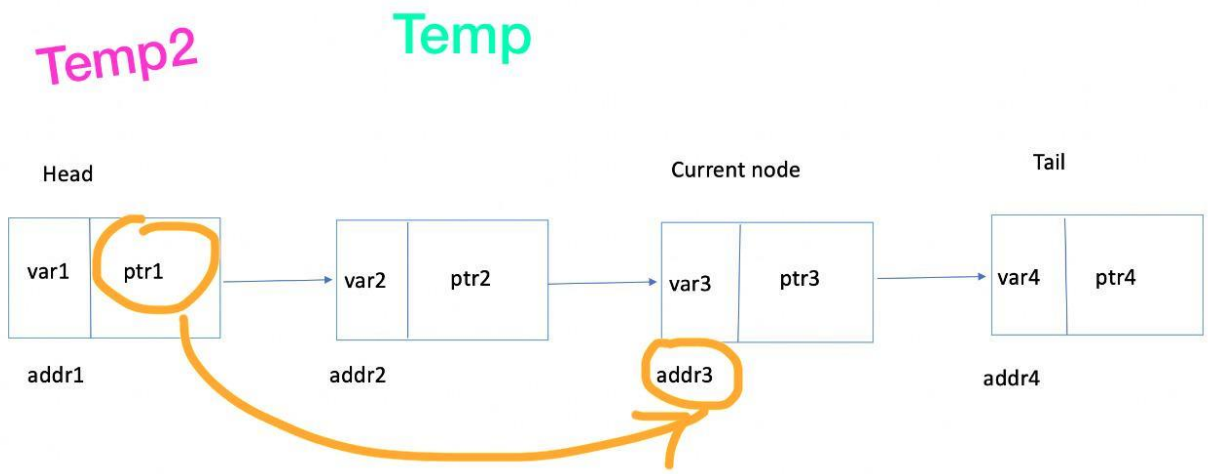
Step 1 -> Find the previous node.

a) Consider a temp node
b) //Finding previous node of current node
   While (temp != NULL && temp->next = current)
   {
   temp=temp->next
   }
c) Finding previous node of the previous node
   While (temp2 != NULL && temp2->next != current)
   {
   temp2=temp2->next
   }

/* link the previous node's previous node with the current node so that the temp is unlinked and the other two nodes (previous node of previous node and current node) are linked thus deleting the previous node*/

Step 2 -> Copy the address value of the current node to the pointer value of previous node of previous node.

For example,



step 1 -> find previous node TEMP
step 2 -> find previous of previous node TEMP2
step 3 -> ptr = addr3 (current node address)
Unlink TEMP between TEMP2 and current => previous node is deleted.