Assignment - 2
DSC

① A node can be inserted at various places in a linked list. Write algorithms for inserting a new node in a single linked list i) At the front of the linked list ii) After a given node iii) At the end of the linked list.

A linked list is an ordered collection of finite, homogeneous data elements called nodes where the linear order is maintained by means of links (or) pointers.

There are Various portions where a node can be inserted

(i) Insertion at the front

(ii) Insertion at the End

(iii) Insertion at any other position

Before these insertions a procedure GetNode(NODE) is assumed to get a pointer of a memory block which suits the type NODE.

Procedure GetNode :-

Input :- NODE is the type of the data for which a memory has to be allocated

Output :- Return a message if the allocation fails else the pointer to the memory block allocated.

Steps :-

```
1.  If (AVAIL = NULL)
2.      Return(NULL)
3.      Print "Insufficient memory : Unable to allocate memory".
4.  Else
5.      ptr = AVAIL
6.      While ( Sizeof (ptr) ≠ Sizeof (NODE)) and (ptr→LINK ≠ NULL) do
7.          ptr1 = ptr
8.          ptr = ptr→LINK
9.      EndWhile
10.     If (SizeOf (ptr) = SizeOf (NODE))
11.         ptr1 → LINK = ptr→LINK
12.         Return (ptr)
13.     Else
14.         print "The memory block is too large to fit"
15.         Return (NULL)
16.     EndIf
17. EndIf
18. Stop
```

<i> **Inserting a node at the front :-**

Algorithm InsertFront_SL

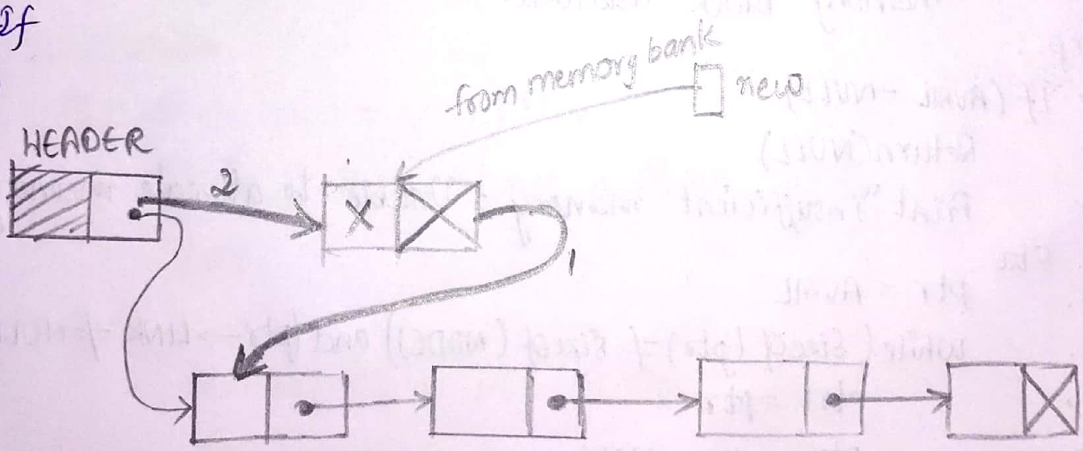
Input : HEADER is the pointer to the header node and X is the data of the node to be inserted.

Output : A single linked list with a newly inserted node at the font of the list

Data Structures : A single linked list whose address of the starting node is known from the HEADER

Steps :-

1. new = GetNode (NODE)
2. If (new = NULL) then
3. print " Memory underflow : No insertion".
4. Exit
5. Else
6. new → LINK = HEADER → LINK
7. new → DATA = X
8. HEADER → LINK = new
9. EndIf
10. Stop



Inserting a node in the front of a single linked - List

(ii) Inserting a node at any Position in the list :-

Algorithm InsertAny_SL

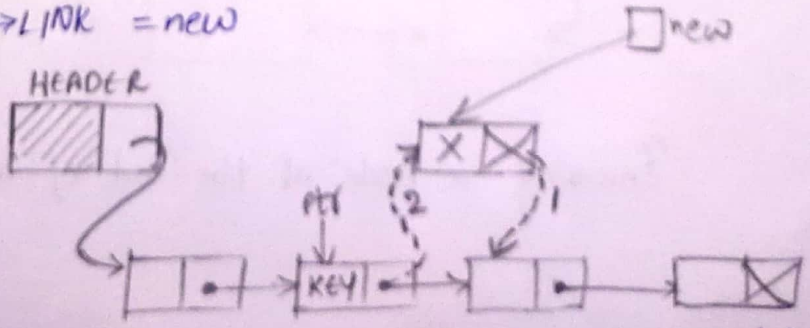
Input : HEADER is the pointer to the header node, X is the data of the node to be inserted and KEY being the data of the keynode after which the node has to be inserted.

Output : A Single linked list enriched with newly inserted node having data X after the node with data KEY

Data Structures :- A Single linked list whose address of the starting node is known from the HEADER

Steps :-

```
1.  new = GetNode (NODE)
2.  If (new = NULL) then
3.        Print "Memory is insufficient: Insertion is not possible"
4.        Exit
5.  Else
6.        ptr = HEADER
7.        while ( ptr→DATA ≠ KEY) and (ptr→LINK ≠ NULL) do
8.              ptr = ptr →LINK
9.        EndWhile
10.       If((ptr→LINK) = NULL) then
11.             print "KEY is not available in the list"
12.             Exit
13.       Else
14.             new→LINK = ptr →LINK
15.             new → DATA = X
16.             ptr →LINK = new
17.       EndIf
18. End If
19. Stop
```



Inserting a node at any position on a Single Linked List.

Scanned with CamScanner

(iii) Inserting a node at the end:-

Algorithm InsertEnd_SL

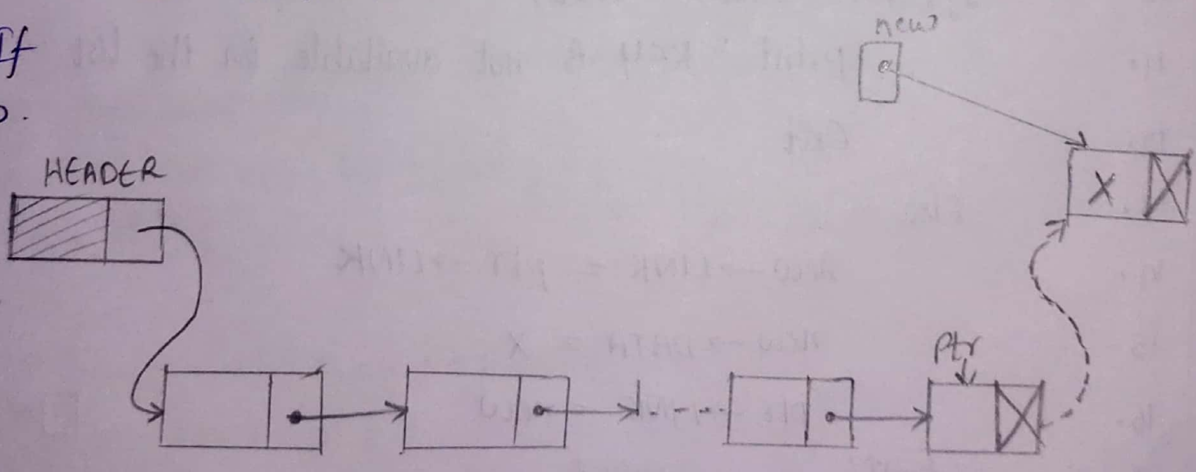
Input : HEADER is the pointer to the header node and X is the data of the node to be inserted.

Output : A Single linked list with a newly inserted node having data X at the end of the list.

Data Structures : A Single linked list whose address of the starting node is known from the HEADER.

Steps:-

```
1.  new = GetNode(NODE)
2.  If (new = NULL) then
3.        print "Memory is insufficient : Insertion is not possible"
4.        Exit.
5.  Else
6.        ptr = HEADER
7.        While (ptr→LINK ≠ NULL) do
8.              ptr = ptr→LINK
9.        EndWhile
10.       ptr→LINK = new
11.       new→DATA = X
12. End If
13. Stop.
```



Inserting a node at the end of a Single Linked List

② Explain quick sort algorithm and simulate it for the following
20, 35, 10, 16, 54, 21, 25

* Quick Sort is a divide-and-conquer algorithm
- Divide Step
  1) Choose an item P (known as pivot) and partition the items of a[i---j] into two parts.
  - Items that are smaller than P
  - Items that are greater than or equal to P
  2) Recursively sort the two parts.
- Conquer step
  Just arrange the elements into a list in same order of last step

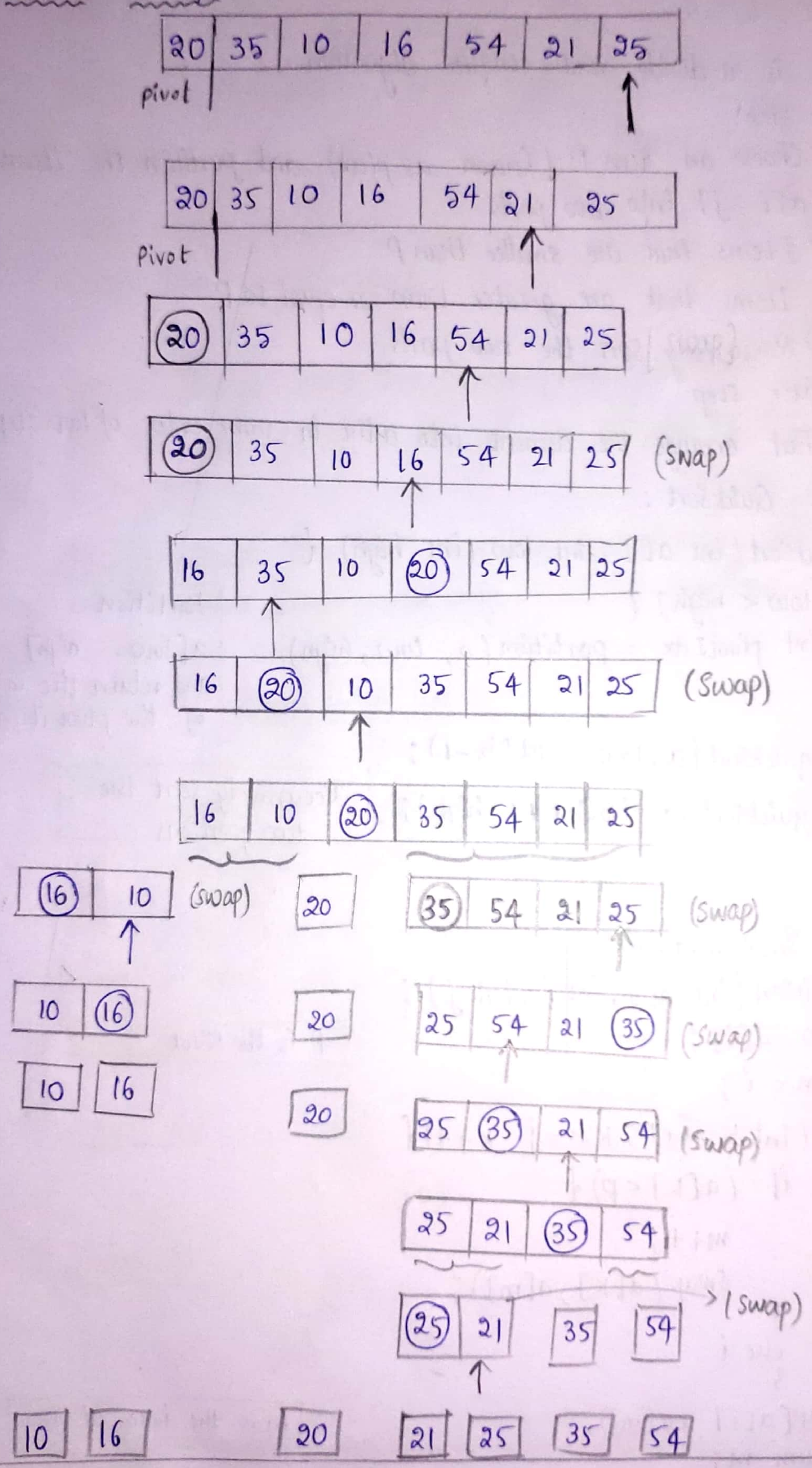
Algorithm QuickSort :

```
void quickSort(int a[], int low, int high) {
    if (low < high) {
        int pivotIdx = partition(a, low, high)      // Partition a[low....high] and return the index of the pivot item

        quickSort(a, low, pivotIdx-1);   }  Recursively sort the
        quickSort(a, pivotIdx+1, high);  }  two portions
    }
}
```

Partition Implementation :-

```
int partition(int a[], int i, int j) {
    int p = a[i];        // P is the Pivot
    int m = i;

    for (int k = i+1; k <= j; k++) {
        if (a[k] < p) {
            m++;
            swap(a[k], a[m]);
        }
        else {
        }
    }
    swap(a[i], a[m]);    // m is the index of pivot
    return m;
}
```

Partition Example :-



After Sorting Using Quick Sort Algorithm :-

10	16	20	21	25	35	54

Time Complexity is $O(n)$

Wort Case $\rightarrow O(n^2)$

Best Case $\rightarrow O(n \log n)$

③ Construct a Binary Search Tree for the following data and do-in order, Preorder & Post-order traversal of the tree. 50, 60, 25, 40, 30, 70, 35, 10, 55, 65, 5.

Binary Search Tree (BST) is a special kind of binary Tree in which every node contains smaller values only in the left subtree and only larger values in its right subtree.

Construction of Binary search Tree :-

Question -

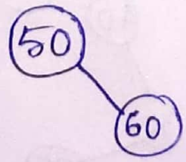
Construct a BST for the following sequence of numbers 50, 60, 25, 40, 30, 70, 35, 10, 55, 65, 5

Solution -

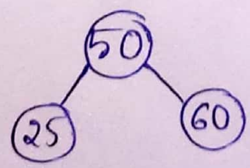
When elements are given in a sequence, we consider the first element as the root node.

1) Insert 50 -

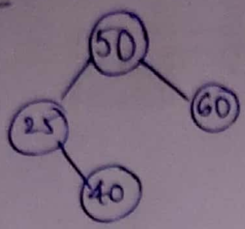
2) Insert 60 -



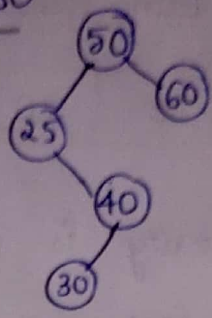
3) Insert 25 -



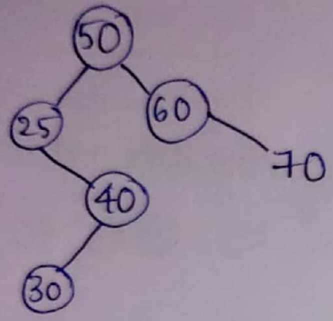
4) Insert 40



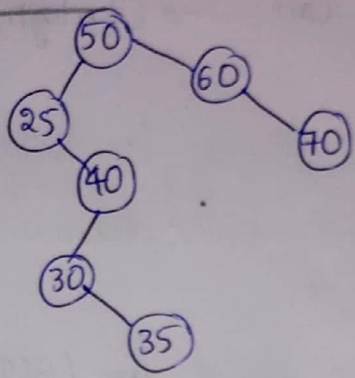
5) Insert 30



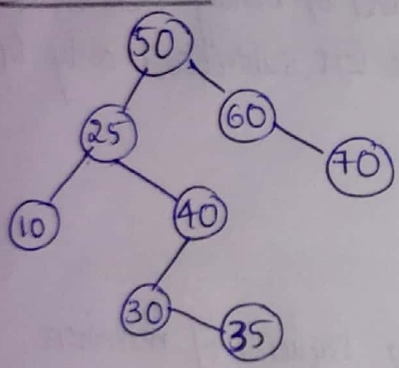
6) Insert 70



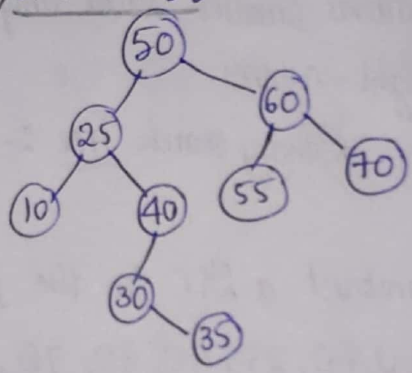
7) Insert 35



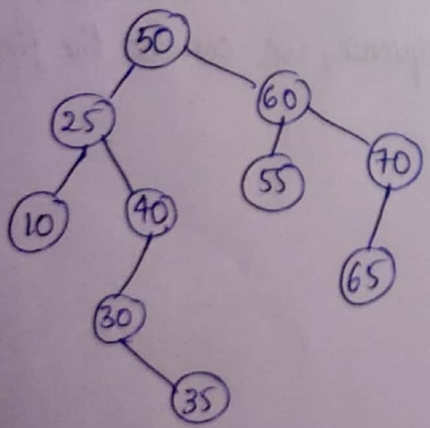
8) Insert 10



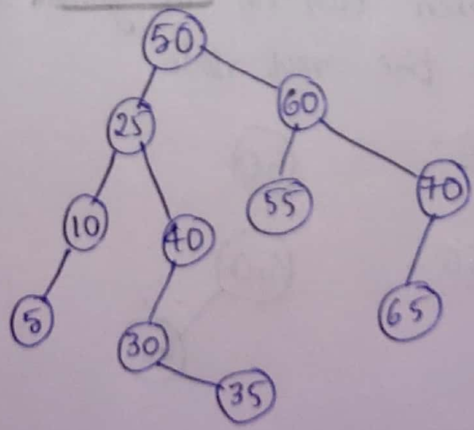
9) Insert 55



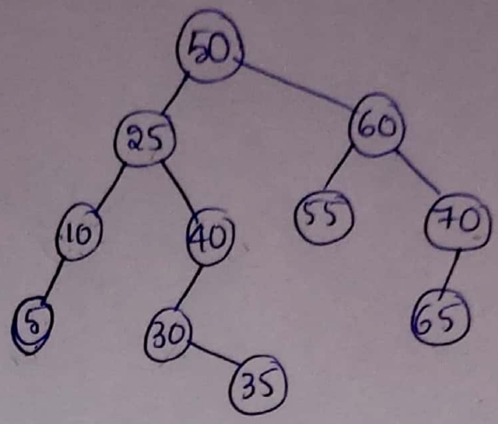
10) Insert 65



11) Insert 5



The Required Binary Search Tree is



(1) Pre-Order Traversal :-

Root → Left → Right

50 25 10 5 40 30 35 60 55 70 65

(2) In-Order Traversal :-

Left → Root → Right

5 10 25 30 35 40 50 55 60 65 70

(3) Post Order Traversal :-

Left → Right → Root

5 10 35 30 40 25 55 65 70 60 50

Scanned with CamScanner