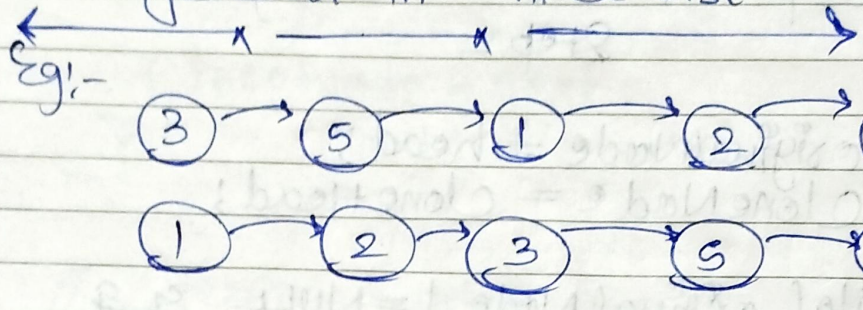
Inp (**)

Page No.
Date

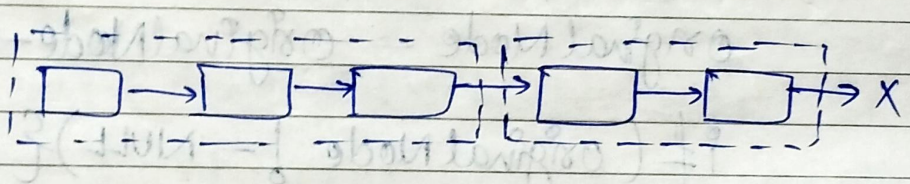
Q Merge Sort in Linked List.

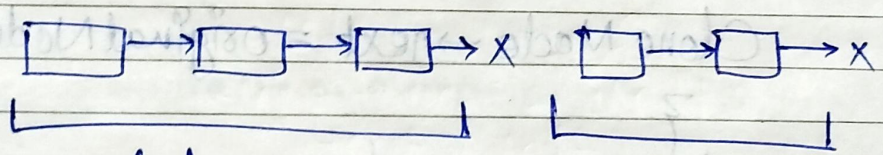


Eg:-

Without replacing the data.
Change links.

Approach : Same as array merge sort





left Part Sort

Right Part Sort

Merge.

T.C = O(n.logn)

code :

```
Class Solution {
Private :
Node * findMid ( Node * head){

    Node * slow = head;
    Node * fast = head->next;

while ( fast != NULL && fast->next != NULL)
    {
      fast = fast->next->next;
      slow = slow->next;
    }
    return slow;
}

Node * Merge ( Node *left , Node * right){
    if ( left == NULL)
            return right;
    if ( right == NULL)
            return left;

// Creating a dummy node
    Node * ans = new Node(-1);
    Node * temp = ans;
```

```
while( left != NULL && right != NULL){
  if( left->data < right->data){
        temp->next = left;
        temp = left;
        left = left->next;
    }
  else {
        temp -> next = right;
        temp = right;
        right = right->next;
      }
}

  while( left != NULL){
        temp->next = left;
        temp = left;
        left = left->next;
    }
  while( right != NULL){
        temp-> next = right;
        temp = right;
        right = right->next;
    }
  ans = ans -> next;
  return ans;  }
```

Page No.
Date

```
public:
Node *mergeSort(Node *head){
    // base case
    if(head==NULL || head->next==NULL)
    {
        return head;
    }
    // break linked list into two half's
    Node *mid = findMid(head);

    Node *left = head;
    Node *right = mid->next;
    mid->next = NULL;

    // Recursive call
    left = mergeSort(left);
    right = mergeSort(right);

    // merge left & right half's
    Node *result = Merge(left, right);

    return result;
}
};
```

→ Refers GFG for more detail.

Page No.
Date

Smart Pointers

As we know unconsciously not deallocating a pointer causes a memory leak that may lead to the crash of the program. Languages like C#, Java has Garabage collection mechanism to smartly deallocate unsed memory to be used again.

The programmer doesn't have to worry about any memory leak. C++ comes up with its own mechanism that the smart pointer.

When the object is destroyed it frees the memory as well. So, we don't need to delete it as smart pointer does will handle this.

(*) C++ provides the absolute flexibility with memory management

- (•) Allocation,
- (•) Deallocation
- (•) Lifetime management

(*) Potentially Serious problem - raw Pointer

(•) uninitialized (wild) Pointer
(•) Memory leaks
(•) Dangling Pointer
(•) Not exception safe.

Q RAII principle ?
= (RAII : Resource Acquisition is Initialization)

(*) C++ Smart Pointers
(•) unique Pointer (unique_ptr)
(•) shared Pointer (shared_ptr)
(•) weak Pointer (weak_ptr)
(•) Auto Pointer (auto_ptr)

→ we have to include the memory header file
#include <memory>

→ Pointer arithemetic in Smart Pointer like ++, --.. is not supported.

Q understand the ownership of the pointers ?

(#) Stack :- (LIFO : Last in first out)

A linear data structure

(•) Push → used to place the values in stack

(•) Pop ~~remove~~ → used to ~~pop~~ remove the values from stack

(•) peek → for the top element

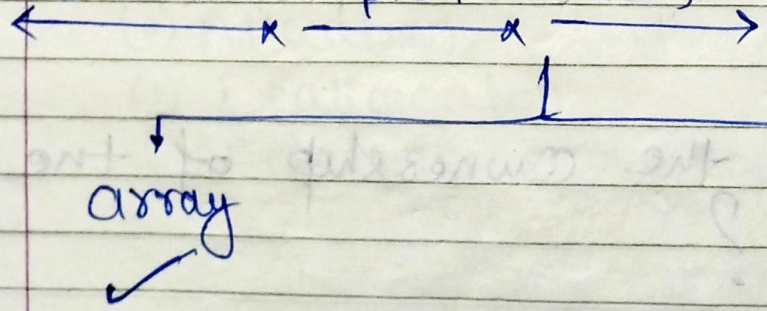
(•) empty → To check for empty

Eg:- By using STL we can create stack

```
Stack<int> S;

S.push(2);
S.pop(5);
S.top();
... etc
```

(*) Stack Implementation



H/W

Implementation using array.

Page No.
Date

Code:

```
#include<iostream>
using namespace std;

class Stack{

Public:
    // data members
    int *arr;
    int top;
    int size;

    // constructor
    Stack(int size){

        this->size = size;
        top = -1;
        arr = new int[size];
    }
    // Push operation

    void push(int element){
    // size available or not

        if(size->top > 1){
            top++;
            arr[top] = element;
        }
        else{
        cout<<" Stack overflow ";
```

```
    }
}
    // Pop operation
    void Pop() {
        // element exists or not
        if (top >= 0) {
            top--;
        }
        else {
            cout<<" Stack underflow";
        }
    }
    int peek() {
        // element exists or not
        if (top >= 0) {
            return arr[top];
        }
        else {
            cout<<" Stack is empty";
        }
    }
    bool isEmpty() {
        if (top == -1) {
            return true;
        }
```

```
else{
    return false;
    }
}
};
int main (){
    Stack st(5);

    st.push(10);
    st.push(20);
    st.push(30);
cout<<"pop element is:"<<st.peek();

st.Pop();

cout<<"pop element is:"<<st.peek();

if( is.Empty()){
        cout<<"stack is empty";
    }
    else {
        cout<<"not empty";
    }
return 0;  }
```

for more detail visit CPH.

Page No.
Date

Exception handling

(•) Exception handling:

(1) Dealing with extraordinary Situations
(2) Indicates that an extraordinary Situation has been detected or has occured
(3) Program can deal with that situation.

(•) What causes exceptions?

1. Insufficient resources
2. Missing resources
3. Invalid operations
4. Range violation
5. underflow or overflow
6. ... etc.

(•) Three keywords for Exception handling

(1) throw
   (•) throws an exception
   (•) followed by an argument

(2) try { code that may throw an exception }
   (•) you place code that may throw an exception
   (•) If the code throws an exceptions the try block is exited

- the throw exception is handled by a catch handler
- if no catch handler exists the program terminates.

(3) catch (Exception ex){ code to handle the exception }

- Code that handles the exception
- can have multiple catch handler
- may or may not cause the prog to terminates

Q Exception classes ?