Step 5: Print the merged array arr3:
arr[i] along with space.

Step 6: End.

Q3) WAP for Linear Search.

Sol- Source Code:

```
# include <stdio.h>
int linear search(int arr[], int key){
  int i,n;
  for(int i=0; i<n-1; i++){
    if (arr[i] == key){
    return i;
    }
  }
  return -1;
}
int main(){
   int i, n, arr[50], key = 5;
   printf("Enter the number of elements: \n");
   scanf("%d", &n);
   printf("The array elements are: \n");
   for(i=0; i<n; i++){
      scanf("%d", &arr[i]);
   }
   int index = linear search (arr, key);
   if (index == -1){
      printf("Number not found");
   else{
      printf("Number found at index: %d, index);
   }
   }
```

INPUT:

Enter the number of elements:
4
The array elements are:
5
6
3
2

OUTPUT:

Number found at index: 0

ALGORITHM:

Step 1: Start

Step 2: Declare variable & initialize i=0
i, n (integer)

Step 3: Repeat the steps while i<n:
→ Check if arr[i] is equal to the key
→ If yes, return the value of i
→ Otherwise, increment i by 1.

Step 4: If the loop completes without finding the key:
→ Return -1 to indicate that the key is not present in the ky.

Step 5: End.

Q3) WAP for Binary Search.

Sol- Source Code:

```
#include <stdio.h>
int binarySearch(int arr[], int ky, int start, int end){
   // start = 0, end = n-1
while (start <= end) {
   int mid = (start+end)/2;
   if(arr[mid] = key){
    return mid;
   }
```

```
    if(arr[mid] < key) {
        start = mid+1;
    }
    else {
        end = mid-1;
    }
}
return -1;
}
int main() {
    int i, n, key = 5, arr[50];
    printf("Enter the number of elements: \n");
    scanf("%d", &n);
    printf("The array elements are: \n");
    for(i=0; i<n; i++) {
        scanf("%d", &arr[i]);
    }
        printf("Key found at index: %d", binarySearch(arr, key, 0, n-1));
}
```

INPUT:

```
Enter the number of elements:
5
The array elements are:
3
6
7
2
9
```

OUTPUT:

```
Key found at index: -1
```

Q) WAP for Bubble Sort.

Sol- SOURCE CODE:

```
#include <stdio.h>
void bubble (int arr[], int n){
    int turn, i, j, temp;
    for(turn=0; turn <n-1; turn ++) {
        for(j=0; j< n-1-turn; j++) {
            if (arr[j] > arr[j+1]) {
                temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }}}
        printf("Array after sorting: \n");
        for(i=0; i<n; i++){
        printf("%d", arr[i]);
        }}
int main(){
    int i, n, arr[50];
    printf("Enter the number of elements: \n");
    scanf("%d", &n);
    printf("The array elements are: \n");
    for(i=0; i<n; i++){
        scanf("%d", &arr[i]);
    }
    bubble(arr,n);
}
```

INPUT:

Enter the number of elements:
5
The array elements are:
3
5
2
1
7

OUTPUT:

Array after sorting:
1 2 3 5 7.

ALGORITHM

Step 1: Start

Step 2: Declare variables & intialize variable turn = 0

~~turn~~, int turn, i, j (loop variables); temp (Integer type temporary variable for swapping); arr (Integer type) and n (Integer type for no. of elements in the array).

Step 3: Outer loop (repeat n-1 times) → Loop goes through each element until the end of the unsorted portion.
Inner loop (repeat n-1-turn times) → Comparing arr[j] with arr[j+1], if arr[j] > arr[j+1], swap the elements as temp = arr[j]
arr[j] = arr[j+1]
arr[j+1] = temp.

Step 4: Print the sorted array: Loop through each element of the array and print the elements followed by a space.

Step 5: End.

Q6) WAP for Selection Sort.

Sol- SOURCE CODE:

```
#include <stdio.h>
Void selection (int arr[], int n){
int temp, i, j, minPos;
for(i=0; i<n-1; i++){
  minPos = i;
  for(j=i+1; j<n; j++){
    if(arr[j] > arr[minPos]){
    minPos = j;
    }}
    temp = arr[i];
    arr[i] = [minPos];
    arr[minPos] = temp;
    }
printf("Array after sorting: \n");
for(i=0; i<n; i++){
printf("%d", arr[i]);
}}
int main(){
  int i, n, arr[50];
  printf("Enter the number of elements: \n");
  scanf("%d", &n);
  printf("The array elements are: \n");
  for(i=0; i<n; i++){
   scanf("%d", &arr[i]);
  }
  selection(arr, n);
}
```

INPUT:

Enter the number of elements:
5
The array elements are:
3
5
2
1
7

OUTPUT:

Array after sorting:
1 2 3 5 7

ALGORITHM → Step 1: Start

Step 1: Declare variables & intialize minPos = i:
i, j, temp (Integer type loop variables); arr (Integer type ~~for~~ array to be sorted); n (Integer type for no. of elements in the array).

Step 3: Loop through each element of the array (i from 0 to n-1): Loop through the remaining unsorted portion of the array (j from i+1 to n-1): Compare arr[j] with arr[minPos]; if arr[j] > arr[j+1], update minPos to j.

Step 4: Swap the smallest element found with the element at position i i.e temp = arr[i]
arr[i] = arr[minPos]
arr[minPos] = temp

Step 5: Print the sorted array → loop through each element of the array and print the elements followed by a space.

Step 6: End.

Q7) WAP for Insertion Sort.

Sol- SOURCE CODE:

```
#include <stdio.h>
Void insertion (int arr[], int n){
   int curr, i, prev;
   for(i=1; i<n-1; i++){
   curr= arr[i];
   prev= i-1;
   while(prev >=0 && arr[prev] > curr) {
      arr[prev + 1] = arr[prev];
      prev --;
   }
   arr[prev+1] = curr;
   }
   printf("Array after sorting: \n");
   for(i=0 ; i<n; i++){
     printf("%d", arr[i]);
   }
}
 int main(){
   int i,n, arr[50];
   printf("Enter the number of elements : \n");
   scanf("%d", &n);
   printf("The array elements are: \n");
   for(i=0; i<n; i++){
      scanf("%d", &arr[i]);
   }
   insertion (arr,n);
}
```

INPUT :

Enter the number of elements :
5
The array elements are :
3
5
2
1
7

OUTPUT :

Array after sorting :
1 2 3 5 7.

ALGORITHM

Step 1 : Start.

Step 2 : Declare variables :
i, prev, curr : (Integer type loop and temporary variables)
arr (Integer type to be sorted)
n (Integer type for no. of elements in the array)

Step 3 : Loop through each element of the array (i from 1 to n-1):
a) Set curr = arr[i] (the element to be inserted)
b) Set prev = i-1 (the index no. of the previous element)

Step 4 : Compare curr with elements before it :-
While prev is greater than or equal to 0 and arr[prev] > curr :
Shift arr[prev] to the right (arr[prev+1] = arr[prev]), then decrement by 1.

Step 5 : Insert curr at the correct position :
Set arr[prev] = curr

Step 6 : Print the sorted array :
loop through each element of the array and print the elements followed by a space.

Step 7 : End.

Q8) WAP for Merge Sort.

Sol- SOURCE CODE:

```
#include <stdio.h>
void merge (int arr[], int left, int right, int mid) {
int size1, size2, i, j, k, a[100], b[100];
size1 = mid - left + 1;
size2 = right - mid;
a[size1];
b[size2];
for(i=0; i<size1; i++) {
a[i] = arr[left + i];
}
for(j=0; j<size2; j++) {
b[j] = arr[mid + 1 + j];
i=0; j=0; k=1;
while (i<size1 && j<size2) {
if (a[i] > b[j]) {
   arr[k] = b[j];
   j++;
}
else {
   arr[k] = arr[i];
   i++;
}
k++;
}
```

```
while (i<size1){
arr[k] = a[i];
i++;
k++;
}
while (j<size2){
    arr[k] = b[j];
    j++;
    k++;
}

void mergeSort (int arr[], int left, int right){
  int mid1;
  mid1 = left + (right-1)/2;
  if(left < right){
  mergeSort(arr, left, mid1);
  mergeSort(arr, mid1+1, right);
  merge(arr, left, right, mid1);
  }
}

void main(){
   int i, n, arr[100];
   printf("Enter the number of elements: ");
   scanf("%d", &n);
   printf("Enter the %d elements: \n");
   for(i=0; i<n; i++){
     scanf("%d", &arr[i]);
  }
```

```
    mergeSort(arr, 0, n-1);
    printf("The sorted elements are: \n");
    for(i=0; i<n; i++){
        printf("%d \t", arr[i]);
    }
}
```

INPUT:

Enter the number of elements: 4
Enter the 4 elements:
2
8
1
5

OUTPUT

The sorted elements are:
1 2 5 8

ALGORITHM

Step 1: Start

Step 2: Input size and elements of the array.

Step 3: First, divide the two arrays into two equal halves.

Step 4: These subarrays are further divided into two halves. The process is continued till the subarrays become of unit length.

Step 5: These subarrays are merged together based on the sorting conditions.

Step 6: The merging process is continued until the sorted array is generated from the subarrays.

Step 7: Display the array elements.

Step 8: End.

LINKED LIST

Normal memory allocation ⟹ int arr[30]

Dynamic memory allocation ⟹ int *ptr (int*)malloc (n*sizeof(int))

⇓

we can use this heap memory also

Advantages over array : Fixed size ka nhi banana padta hai.

Contigous memory allocation nhi krna padta hai

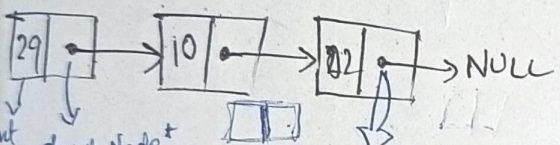
Aaraam se kbhi bhi extend krskte hai.

Insertion easy hojata (shift krne a zarurat nhi hai)

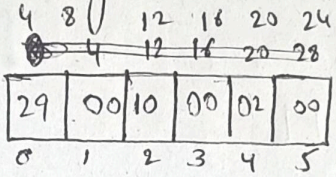
Array mein element ko access krna easy hojata hai

Linked list mein pura traverse krna pdega.

LL mein insertion/deletion easy // array mein hard hojata hai



yeh pointer hai jo address store kr rahe hai



For 4th elements accessing,

4 + n x 4

⇒ 4 + 4 x 4 = 20

⇒ Har baar extra memory banana pd rahe hai pointer ke liye

** Structure is a user-defined data type where we can store different data-type under a single type

INSERTION

Beginning O(1)

```
ptr -> next = head;
head = ptr;
return head;
```

In between O(n)

```
ptr -> next = p -> next
p -> next = ptr
```

at the end O(n)

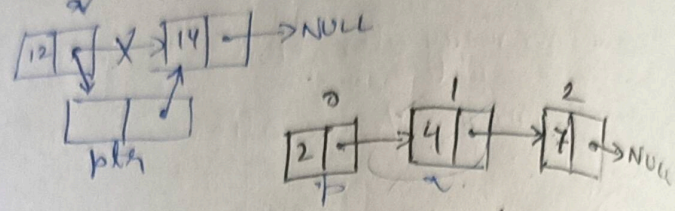
```
ptr -> next = NULL;
p -> next = ptr
```

p is a ptr which is traversed from begining

after a Node (q ptr given) O(1) → Because Traverse krne ki zaroorat nhi hai, ptr given hai

ptr->next = q->next

q->next = ptr



DELETION

First Node O(1)

struct *Node *ptr = head

head = head -> next

free(ptr)

Node in Between O(n)

struct Node *p = head (Traverse krne ka liye)

while (p != index - 1){ p = p -> next; }

struct Node *q = p->next

p->next = q->next

free(q)

Last Node

p aur q do ptr chlao aur jaise hi q ka next NULL hojata hai q ko free krdo aur

p->next = NULL.

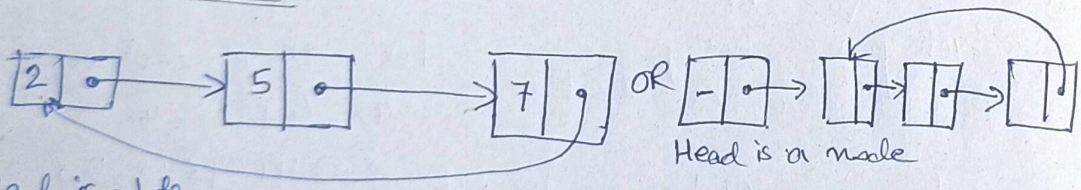
free(q).

Node with given value

struct *deleteGivenNode (*head, *toDeleteNode)

p aur q ptr banayegein aur jaise hi q toDelete Node ke barabar hojata hai waise hi q ko free krdegein aur p->next = q->next.

CIRCULAR LL



Head is a ptr

// BEST WAY

Traverse

while(ptr->next != head){

print(ptr->data)

ptr = ptr->next }

print(p->data)

do {

print(ptr->data) }

while(ptr->next != head);

~~ptr = ptr->prev~~

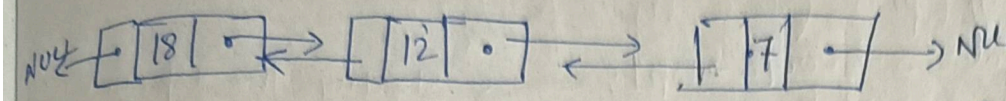
~~//~~ Insertion between & after a Node pura same hoga singly LL ke tarah

Empty Singly LL

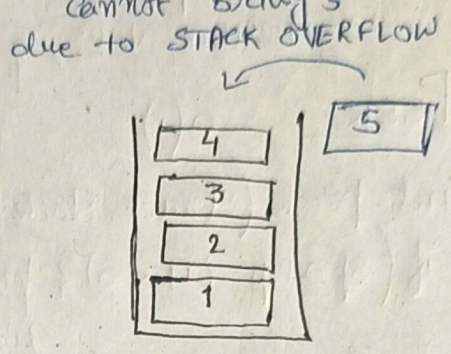
p -> NULL

Empty Circular LL

DOUBLY LL



STACK



LIFO Data Structure

Applications: Functions calls
Infix to Postfix expression
Parenthesis matching

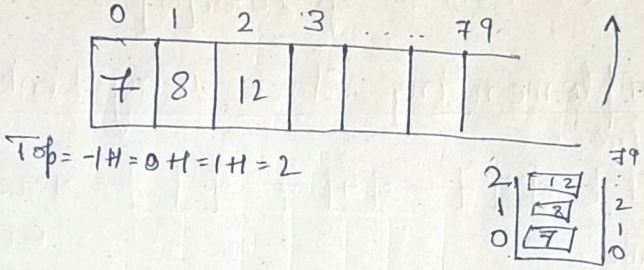
Stack ADT ⟹ One pointer pointing the topmost element.

Implementing Stack using Arrays

-> Fixed size array creation

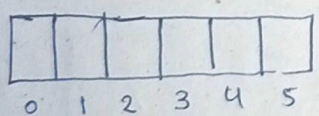
-> Top element

```
struct stack{
    int size;
    int top;
    int *arr; }
struct stack s;
s.size = 80;
s.top = -1; [stack is currently empty]
s.arr = (int *) malloc (s.size * sizeof (int));
```




```
struct stack *s;
or s -> size = 80;
   s -> top = -1;
   s -> arr = (int *) malloc (s->size * sizeof(int));
```

Push, Pop another other operations.



⟹ Cannot Push if full
Cannot Pop if empty

PUSH O(1)

```
struct stack *s;{
s -> size = 8;
s -> top = -1;
s -> arr = (int *) malloc (s->size * sizeof (int));
if (isFull (s)) { print (stack overflow) ; }
else { s -> top ++; s -> arr [s -> top] = value; }
```

```
int isFull (struct stack *ptr){
if (ptr -> top == ptr -> size - 1){
return 1; } else {
return 0; } }
```
O(1)

```
int isEmpty (struct stack *ptr){
if (ptr -> top == -1){
return 1; }
else { return 0; }
```
O(1)

Pop (Nothing is returned) O(1)

```
if(isEmpty(s)){ print(Stack Underflow); return -1; }
else{ int val = s->arr[s->top];   ⟹ Storing the topmost value
      s->top = s->top-1;
      return val; }
```

PEEK O(1)

Position (i)		Array Index
1	29	2 (Top-i+1)
2	10	1
3	2002	0

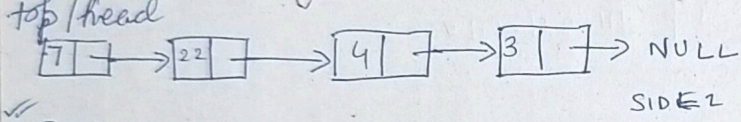
```
int peek(struct Stack *sp, int i){
 if(sp->top-i+1<0){ print(Not valid);
                    return -1; }
 else{ return ~~sp->top~~ sp->arr[sp->top-i+1]; }
```

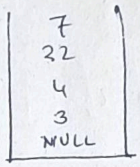
StackTop ⟹ return sp->arr[sp->top];

StackBottom ⟹ sp->arr[0]

[T.C: ~~O(n)~~ O(1)]

Implementing Stack using Linked List





SIDE 1 (for push & pop)

⟹ Stack empty condition → top == NULL

⟹ Stack full condition → when heap memory is exhausted.

⟹ You can always set a custom size.

① int ~~void~~ isEmpty
```
if(top == NULL)
{return 1;}else{return 0;}
```

② int ~~void~~ isFull
```
struct Node* n = (stru..)malloc(())
if(n == NULL){return 1;}
else{return 0;}
```

③ PUSH (Inserting Node at index 0)
```
struct Node *n = (struct Node*) malloc(sizeof(struct Node));
if(isFull(sp)){ print(Stack Overflow!!); }  OR  if(n==NULL){pr...}
else{ n->data = x;
      n->next = top;
      top = n; }
```

POP

```
if (isEmpty(sp)) { print(Stack Underflow) }
else { struct Node *n = top;
       top = top->next;
       free(n);
       return x; }
```

→ int x = n->data;

} Global variable mein top ko declare krna pdega

PEEK

pos	n times movement
1	0
2	1
3	2

```
peek(int pos){ struct Node *ptr = top;
for(i=0; (i<pos-1 && ptr!=NULL); i++) { ptr = ptr->next; }
if (ptr!=NULL) return ptr->data; else return -1;
```

Stack Top ⇒ return top->data

Stack Bottom ⇒
```
struct Node *ptr = top;
while ptr!=NULL
ptr = ptr->next
return ptr->data;
```

operator	precedence
( )	3
*/	2
+-	1

+ / % ±

Infix, Prefix & Postfix

Infix
<operand1> <optor> <operand2>

a+b p/q
a-b p*q

Prefix (Not Impt)
<optor> <oprnd1> <oprnd2>

+ab /pq
-ab *pq

Postfix
(for fast evaluation in computer)

<oprnd1> <oprnd2> <operator>

ab+ pq/
ab- pq*

IN ⇒ A *(B+C) * D

PST ⇒ ABC+*D*

A (B+C)*D*

A*(B+C)D*

A*(B+C)*D

Q1) x-y*z

S1 (Prefix) Parenthesize the expn

(x-(y*z))

(x-[*yz])

-x*yz

P-Q-R/a

((P-Q)-(R/a))

L→R ⇒ (P-Q)-/Ra

⇒ -PQ-/Ra

⇒ --PQ/Ra

S2 (Postfix)

(x-(y*z))

(x-[yz*])

xyz*-

((P-Q)-(R/a))

⇒ ((P-Q)-(Ra/))

⇒ (PQ-)-(Ra/)

⇒ PQ-Ra/-

BODMAS

	Precedence	Assoc.
( )	3 (Highest)	
/*	2	
+-	1 (Lowest)	