Assignment : 1

Ans 1: write the following stack algorithms and explain with example.

1) push
2) pop
3) peep
4) change.

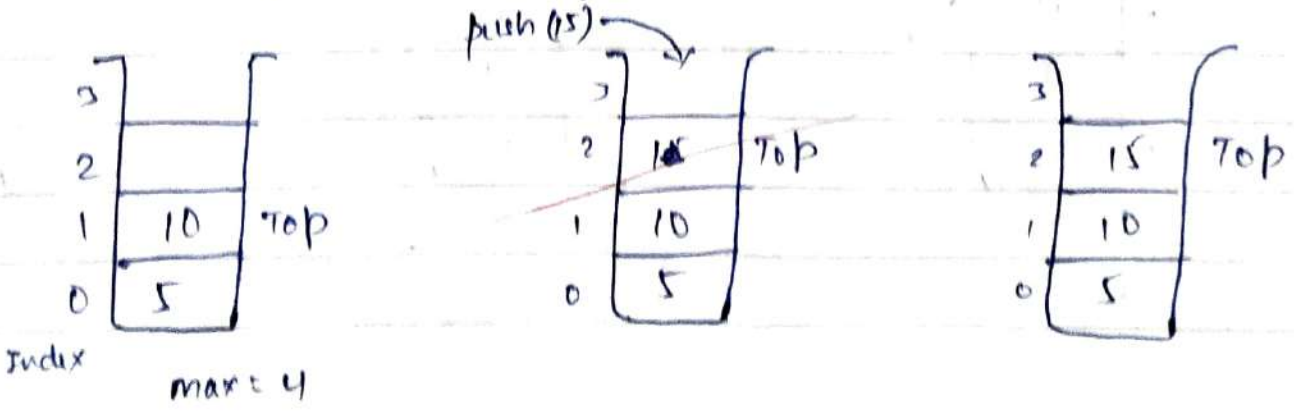
Sol^{n}- Stack: A stack is a linear data structure which can be implemented using an array or a linkedlist.

we have to take two variable in order to perform any operation on stack, which is top (used to store topmost element address) and max (used to store the size of stack)

if top = -1 then stack is empty
if top = max - 1 then stack is full.

i) push operation:

s — stack
max — maximum no. of element in stack
top — indix of topmost element



• push algorithm

Step 1: [check for stack overflow]
if top = max - 1 then
write "stack overflow"
return
[end of if]

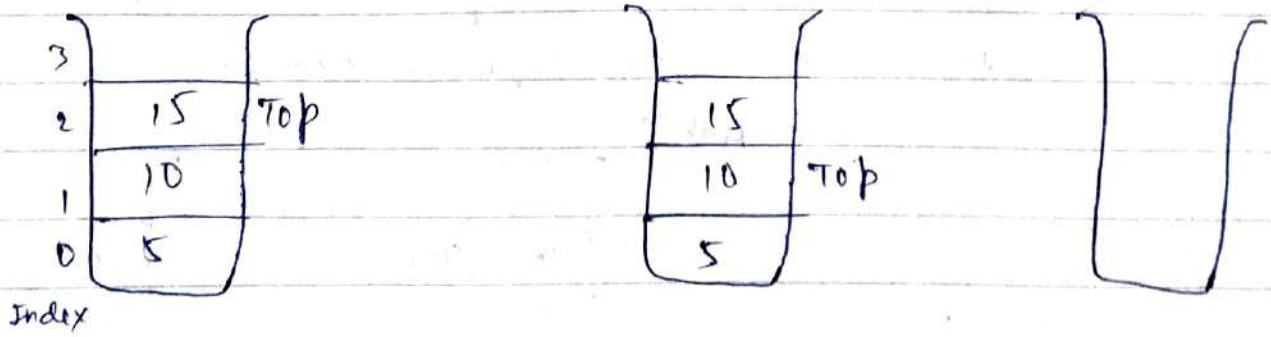
Step 2: [Increment top]
set top = top + 1

Step 3: [insert element]
set S[top] = value

Step 4: [finished]

(ii) pop operation

S - stack
max - size of stack
top - index of topmost element.



max: 4

- pop algorithm

Step 1: [check for stack underflow]
if top = -1
write "stack underflow"
return

Step 2: [print/return topmost element]
write s[top]

Step 3: [Decrement top by 1]
set top = top - 1

Step 4: [finished]

(iii) peep/peek operation:
s - stack
max - size of stack
top - index of topmost element

Step 1: [check for stack underflow]
if top = -1
write "stack underflow"
return

Step 2: [print/return topmost element]
write s[top]

Step 3: [finished]

(iv) Change element operation

s- stack
max- size of stack
top = index of topmost element.
pos- position at which element changed.

step 1: [check for position underflow]
if top - pos + 1 < 0
write "stack underflow for position"
return

step 2: [change the element]
s[top - pos + 1] = new_value

step 3: [Finished]

Qno2. Convert the following infix expression in postfix & prefix using stack.

1) (A+B)/(C*D^E)-F

2) A+(B-C^(D*E))/F

sol^n- (i) (A+B)/(C*D^E)-F

infix to postfix

symbols	stack	expression
(	(	
A	(	A
+	(+	A
B	(+	AB
)		AB+
/	/	AB+
(	/(	AB+
C	/(	AB+C
*	/(*	AB+C
D	/(*	AB+CD
^	/(*^	AB+CD
E	/(*^	AB+CDE
)	/	AB+CDE^*
-	/-	AB+CDE^*
F	/-	AB+CDE^*F/
		AB+CDE^*/F-

- (A+B) / (C * D ^ E) - F

infix to postfix

Step 1: Reverse the infix string

Step 2: Convert the infix to postfix

Step 3: Reverse the postfix

1 - F - (E ^ D * C) / (B + A)

Symbols	Stack	expression
F		F
-	-	F
(	-(	F
E	-(	FE
^	-(^	FE
D	-(^	FED
*	-(*	FED^
C	-(*	FED^C
)	-	FED^C*
/	-/	FED^C*
(	-/(	FED^C*
B	-/(	FED^C*B
+	-/(+	FED^C*B
A	-/(+	FED^C*BA
)	-/	FED^C*BA+
		FED^C*BA+/-
		-/+AB*C^DEF

(ii) A + (B - C ^ (D * E)) / F

infix to postfix

Symbol	Stack	expression
A		A
+	+	A
(	+ (	A
B	+ (	AB
-	+ (-	AB
C	+ (-	ABC
^	+ (- ^	ABC
(	+ (- ^ (	ABC
D	+ (- ^ (	ABCD
*	+ (- ^ (*	ABCD
E	+ (- ^ (*	ABCDE
)	+ (- ^	ABCDE *
)	+	ABCDE * ^ -
/	+ /	ABCDE * ^ -
F	+ /	ABCDE * ^ - F
		ABCDE * ^ - F / +

- A + (B - C ^ (D * E)) / F
  infix to postfix

Step 1: Reverse the infix string
Step 2: Convert infix to postfix
Step 3: Reverse the postfix string

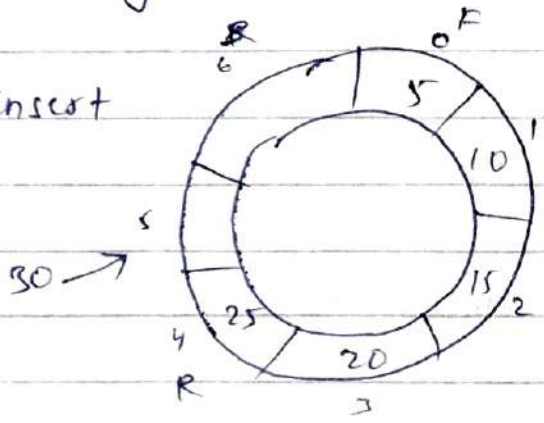
→ A + (B - C ^ (D * E)) / F
F / ((D * E) ^ (- B) + A

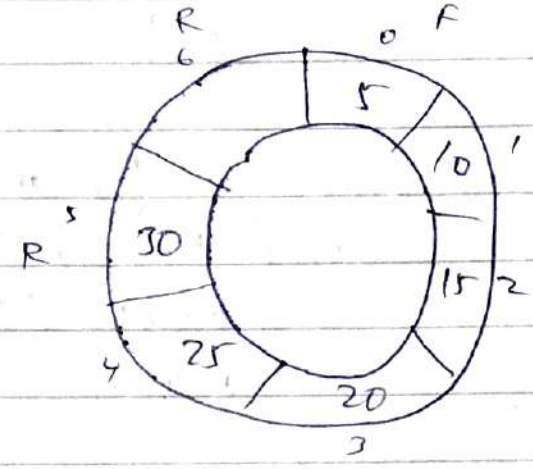
Symbol	Stack	expression
F		F
/	/	F
(	/(	F
(	/((	F
E	/((	FE
*	/((*	FE
D	/((*	FED
)	/(	FED*
^	/(^	FED*
(	/(^(	FED*(
-	/(^(-	FED*(^
B	/(^(-	FED*(^B
)	/	FED*(^B-
+	+	FED*(^B-/
A	+	FED*(^B-/A
		FED*(^B-/A+
		+A/-B^C*DEF

Qno 5. write the following circular queue algorithm and explain with example.

1) Insert
2) Delete
3) Peep
4) Change

Soln - 1) Insert





step 1: [checking for overflow]
if (F = 0 and R = max-1) or (R = F-1) then
write "Queue overflow"
return

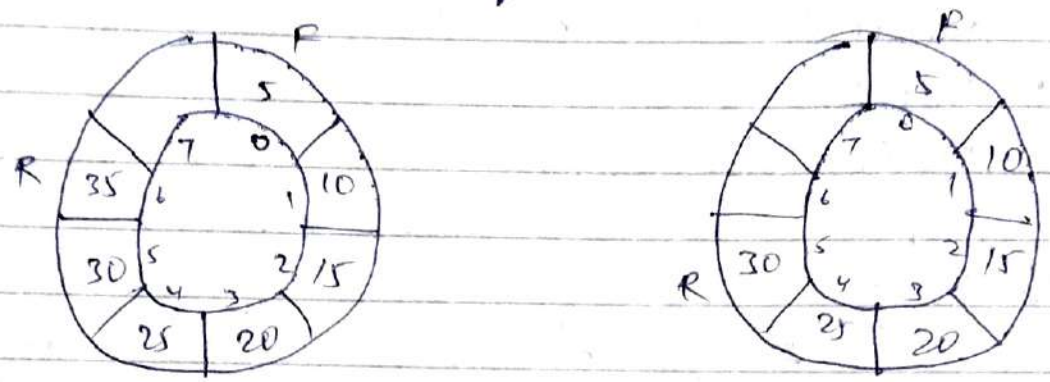
step 2: [reset Rear pointer]
set R = (R + 1) % max

step 3: [insert an element]
set Q[R] = Y

step 4: [checking queue is empty then set front pointer]
if F = -1 then
F = 0

step 5: [Finished]

(ii) Delete from circular queue.



Step 1: [checking for underflow]
if (F = -1) then
write "Queue underflow"
return

Step 2: [print/return deleted element]
write Q[F]

Step 3: [increment front pointer]
[Only one element in Queue]
if F = R then
F = R = -1
else
F = (F + 1) % max
return

Step 4: [Finished]

(iii) peep operation.

```
step1: [checking for underflow]
       if (F = -1) then
          write "Queue underflow"
          return

step2: [print/return recently inserted element in queue]
          write Q[(R+1) % max]
          return

step 3: [finished]
```

(iv) change operation

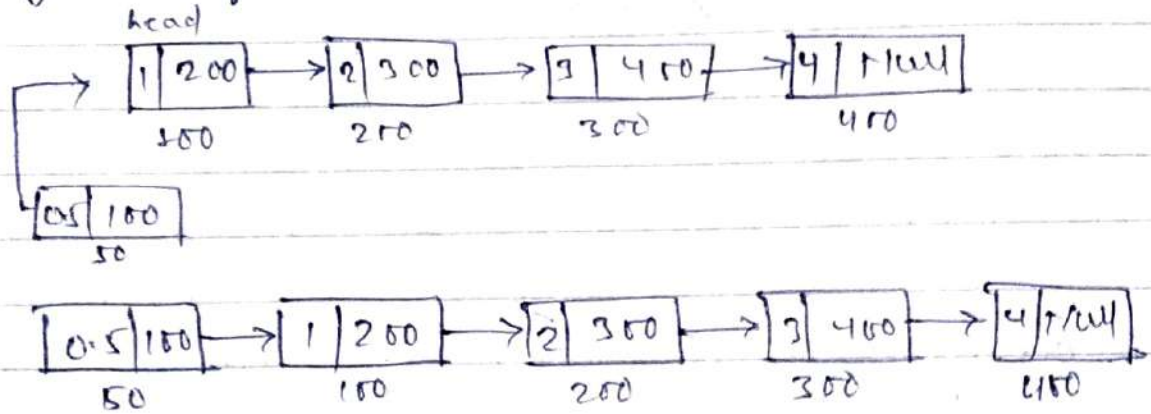
Q - queue, Found = 0, O - old element, Y - new element

```
step1: Repeat this stp till all Queue traversal.
          if (Q[i] = O) then
             set Q[i] = Y
             set Found = 1
             write "element changed"
             return
          if (i = R)
             return
          set i = (i+1) % max
step2: if (Found = 0) then
          write "element not found"
          return
step 3. [finished]
```

Qno 4. write the following singly linkedlist algorithm and explain with example.

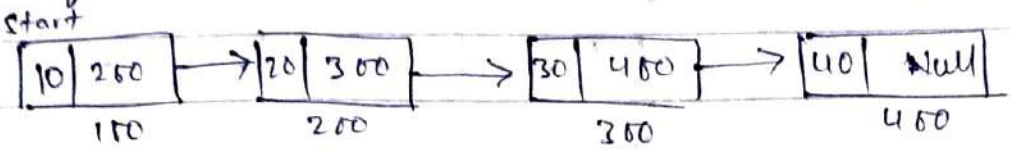
(i) Insertion at beginning.
(ii) Insest after given node.
(iii) Delete at end.
(iv) Traverse in a linked list.

sol^n - (i) Algorithm for new node insertion at beginning.



step 1: If new_node = Null
write "memory overflow"
Go to step 5 /return
step 2: set new_node -> data = val
step 3: set new_node -> next = head
step 4: set head = new_node
step 5: Exit.

(ii) Algorithm for new node is inserted after the given node.

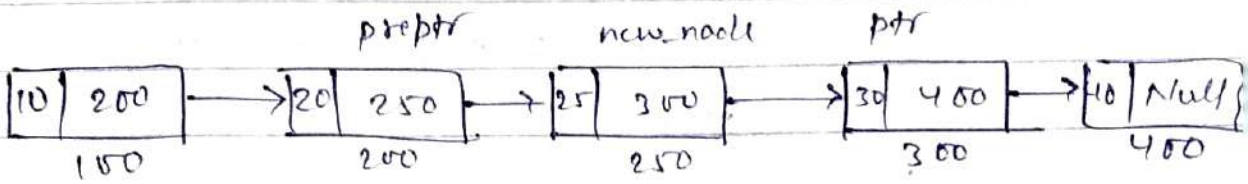


Given node - 20

Take two extra pointer which is ptr and preptr.

new_node value = 25

address = 250



Step 1: [check for memory overflow]

if new_node = Null

write "memory overflow"

return

2: set new_node -> data = val

~~set new_node -> next =~~

3: set ptr = head

4: ~~Repeat~~ set preptr = ptr

5: Repeat while preptr -> data != Num

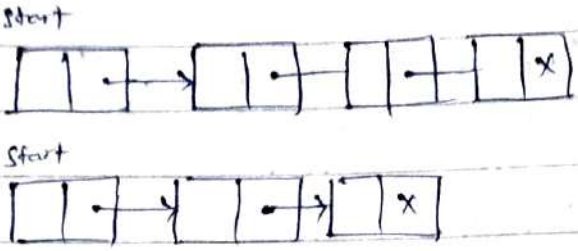
set ptr = ptr -> next > set preptr = ptr

6: preptr -> next = new_node

7: set new_node -> next = ptr

8: Exit.

(iii) Delete at end



step 1: [check for linkedlist underflow]
if head = null
write "Linkedlist underflow"
return

step 2: set ptr = head

3: Repeat while ptr -> next != Null
set preptr = ptr
set ptr = ptr -> next

4: set preptr -> next = null

5: free ptr

6: [Finished]

(iv) Traversing in a singly linked list

or

step 1: set ptr = start/head

step 2: Repeat while ptr != Null
write ptr -> data
set ptr = ptr -> next

step 3: [Finished]

Qno 5. 1) Perform merge sort and quick sort for given below data 91, 15, 73, 28, 11, 37, 50

ans- Quicksort – It is based on divide and conquer strategy to divide a list into two sub lists.

→ The steps are:

- Pick an element, called a pivot, from a list
- Reorder the element so that all element which are less than the pivot element come before the pivot and all the greater element come after the pivot. After this partitioning, the pivot is in its final position. This is called the partition operation.
- Recursively sort the sub list of lesser element and the sub list of greater element.

Eg- 91, 15, 73, 28, 11, 37, 50

↓
Pivot

Pass I 15, 91, 73, 28, 11, 37, 50

15, 73, 91, 28, 11, 37, 50

15, 73, 28, 91, 11, 37, 50

15, 73, 28, 11, 91, 37, 50

15, 73, 28, 11, 37, 91, 50

15, 73, 28, 11, 37, 50, 91

less than pivot — greater than Pivot

After 1st pass pivot is in its right position.

Pass-II

Pivot

(15) 73, 28, 11, 37, 50, 91

15, 73, 28, 11, 37, 50, 91

15, 73, 28, 11, 37, 50, 91

11, 15, 73, 28, 37, 50, 91

greater than pivot

Pass-III

Pivot

11, 15, (73), 28, 37, 50, 91

11, 15, 28, 73, 37, 50, 91

11, 15, 28, 37, 73, 50, 91

11, 15, 28, 37, 50, 73, 91

less than pivot

Pass-IV

Pivot

11, 15, (28), 37, 50, 73, 91

11, 15, 28, 37, 50, 73, 91

Algorithm

Step 1: Repeatedly increase the pointer down by one position until a[down] > pivot

Step 2: Repeatedly decrease the pointer up by one position until a[up] <= pivot.

step 3: if down < up, interchange a[down] and a[up]

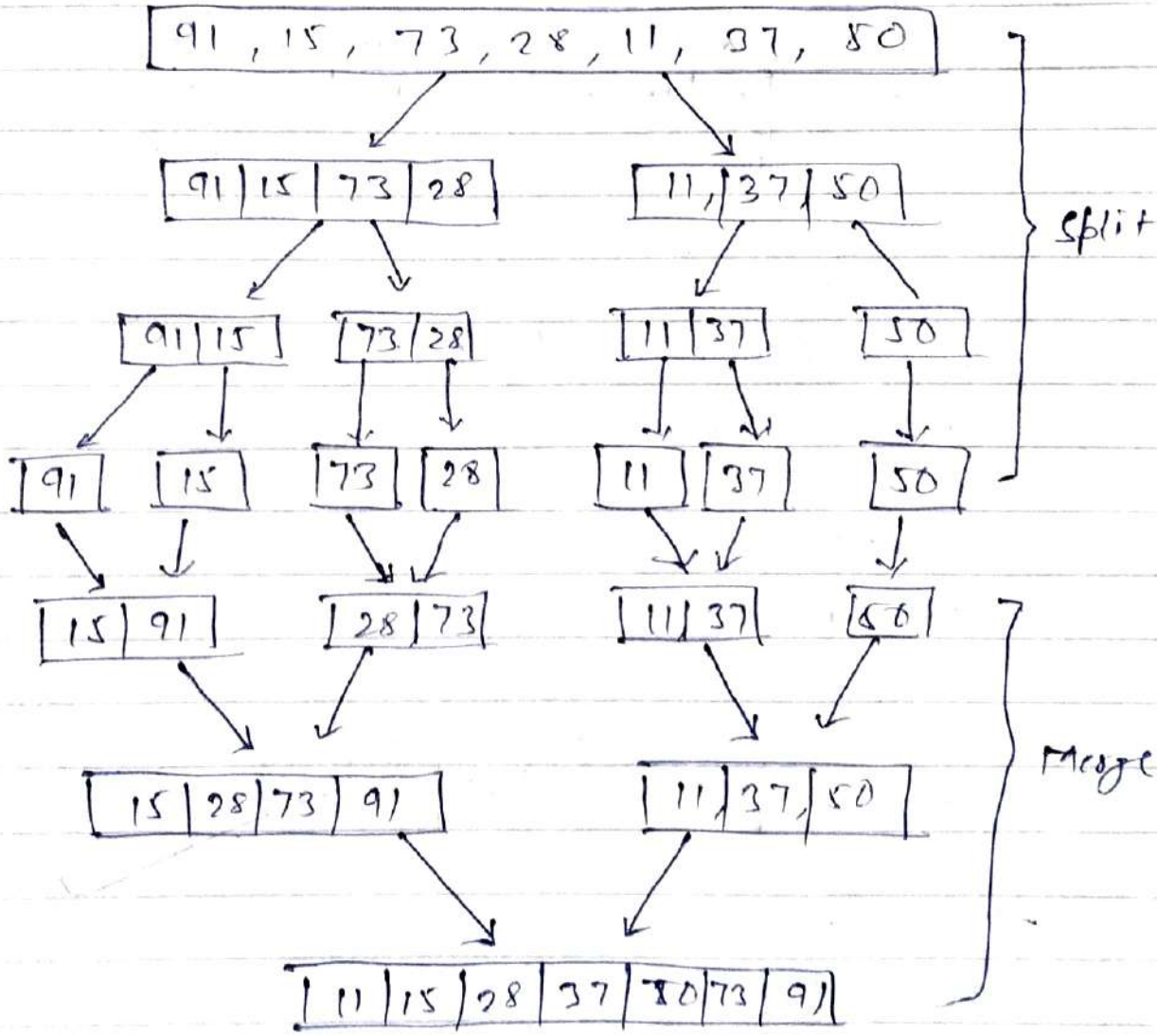
steps 1, 2 and 3 are repeated until step 3 fails

if up <= down, interchange pivot and a[up]

- Mergesort : It is also based on divide & conquer strategy

The steps are :—

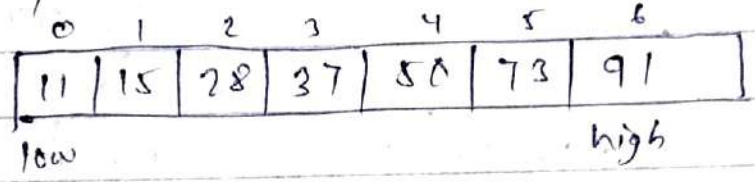
- Divide the unsorted list into two sub lists of about half size.
- Sort each sub list recursively by re-applying merge sort, till you reach a single element array.
- Merge the sub lists back into one sorted list.



2) Perform the binary search for given below data (73)
11, 15, 28, 37, 50, 73, 91

Sol"- Binary search : If array or linkedlist sorted binary search is more efficient algorithm. It also based on divide and conquer algorithm.

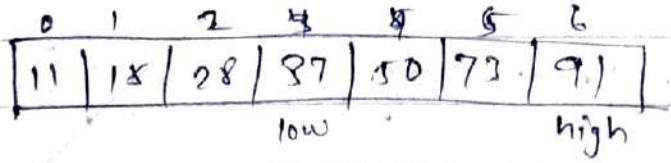
Search Key = 73
Given array = 11, 15, 28, 37, 50, 73, 91



low = 0
high = 6
mid = (low + high)/2 = 3
a[mid] = 37

37 < 73
low = mid + 1
= 3 + 1 = 4

Now,



low = 4
high = 6
mid = (low + high)/2 = 5
a[mid] = 73 // found