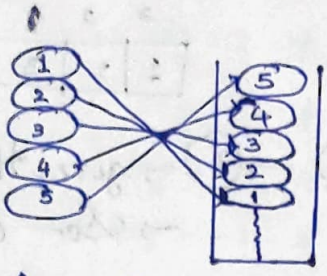
Stack:

→ It is a Last In First Out. (LIFO) data structure.



Reverse an Array Using Stack:

```
int[] array = [1, 2, 3, 4, 5]
Stack<Integer> st = new Stack<>();
for(int i=0; i<array.length; i++)
{
    st.push(array[i]);
}
while(!st.isEmpty()) {
    S.O.pln(st.peek());
    st.pop();
}
```

O/P: 5, 3, 4, 2, 1

Stack Operations:

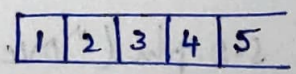
→ push() ⟶ to push / store the element in the stack.

→ pop() ⟶ to remove the element from stack.

→ isEmpty() ⟶ to check if the stack is free (or) not, returns true/false.

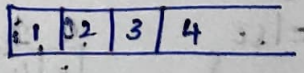
→ peek() ⟶ To get the Top element from stack.

```
S.O.pln(stack.pop());
```



⇓

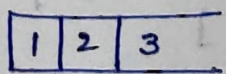
5 ⟶




```
S.O.pln(stack.pop());
```

⇓

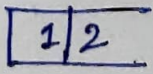
4 ⟶




```
S.O.pln(stack.pop());
```

⇓

3 ⟶

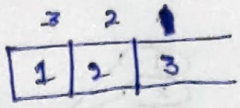


⇓

```
S.O.pln(stack.pop()); ⟶ 2
```

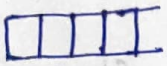
Time Complexity for Operations:

1. Delete (pop()) ⟶ O(1)
2. Insert ⟶ O(1)
3. Search ⟶ st.search(2) → 2 → gives the index → So O(n)
4. Peek ⟶ O(1)



Applications:

→ We use Stack in Recursion.
→ Browser. → when we press back button, all the webpages stored in stack.

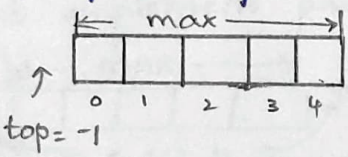


→ Undo & Redo
→ Infix to PostFix

Array Implementation of Stack:

i.e without stack, implement push and pop operations.

Stack[] =



Push:-

At first top = -1, max = 4, max-1 = 3

→ increment top and store the value at stack[top]

Stack[top] = 5

→ top = 0 [0 ≠ 3] increment top, top = 1

stack[1] = 4

→ top = 1 [1 ≠ 3], ++top = 2

stack[2] = 3

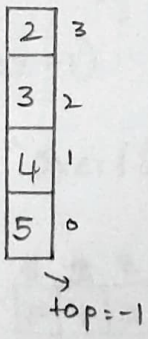
→ top = 2 [2 ≠ 3], ++top = 3

stack[3] = 2

→ top = 3, ~~++top = 4~~

top = max-1 ⇒ 3 = 3

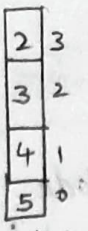
So Overflow



→ check top == max-1, before adding to array.

```
void push (int number)
{
  if (top == max-1)
  {
    S.O.pln ("Overflow");
  }
  else
    {
      Stack[++top] = number;
    }
}
```

Pop():-



→ Top is pointing at index 3.

→ Stack[3] = 2, then decrement the top

→ top = 2 ≠ -1, pop the element. Stack[2] = 3, top--

→ top = 1 ≠ -1, pop the element. Stack[1] = 4, top--

→ top = 0, ≠ -1, " " Stack[0] = 5, top--

→ top = -1 = -1, Underflow.

```
int pop() {
    if (top == -1) {
    s.o.pln ("Underflow");
    }
  else
    {
    int element = stack[top];
       top--;
    }
    return element;
}
```

isFull():

→ if top is pointing at max-1 then array is full. return true.

```
boolean isFull() {
    if (top == max-1) {
     return true;
    }
    return false;
}
```

isEmpty():-

→ If the top is pointing at -1, then return true.

```
boolean isEmpty() {
    if (top == -1) {
      return true;
    }
    return false;
}
```

top():-

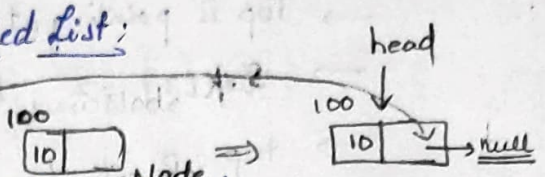
first check if the array is empty(or) not. and then array[top]

```
int top() {
 if (isEmpty() {
     return arr[top];
 }
    return -1;
}
```

Implementation of Stack Using Linked List:

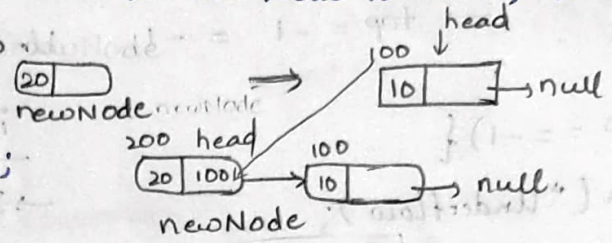
Push():

① head = null

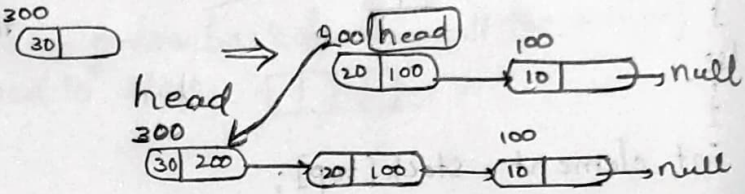


→ At first head is pointing to null
→ Push an element by creating a newNode
→ point next of newNode to head, and head to newNode

②



③



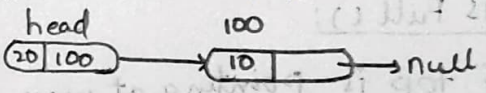

```
Void push (int element){
Node newNode = new Node(element);
newNode.next= head
//newNode point to head
   head = newNode;
}
```

Pop():-

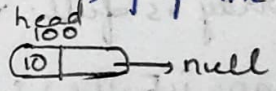
→ removing the element. i.e break the connection of the top element. i.e move the head to head.next

```
void pop() {
   if(head! =null) {
      head= head.next;
   }
}
```

① head is at 30, pop the element.



② head is at 20, pop the element.



top:-

→ get the data of the head pointing Node.

```
int top() {
 if (head !=null) {
   return head.data;
  }
 return -1;
}
```

```
Boolean isEmpty() {
   if (head ==null) {
      return true;
   }
   return false;
}
```

```
int Size(){
 Node current = head;
 length =0;
 while (current! =null) {
 current = current.next;
 length ++;
 return length; }
}
```

Infix → between operands 4+5

Prefix → before " +45

PostFix → after " 45+

Infix	Prefix	Postfix
a*b	*ab	ab*
a-b	-ab	ab-

1) Infix → x-y*z use BODMAS.

Step 1:- Parenthesize the expression based on presedence.

* has more presedence than -

x-(y*z)

Step 2: go to inner most bracket and compute the result:

Prefix: x-[*yz] | Postfix: x-[yz*]

-x*yz | xyz*-

2) $P - q - \frac{x}{a}$

Prefix: $(P-q) - (\frac{x}{a})$

$P-q - /xa$ (A = Pq, B = /xa)

$-Pq - (/xa)$ → A-B → -AB

$--Pq/xa$

Post Fix:

$(P-q) - (\frac{x}{a})$

$P-q - (xa/)$

$(Pq-) - (xa/)$

$Pq - xa/-$

3) (m-n) * (p+q)

Prefix:

[-mn] * [+pq]

* -mn+pq

Postfix:-

(mn-) * (pq+)

mn-pq+*

Infix to Postfix: ⟶

Postfix abc+*df

523+*4*

5(2+3)*4*

55*4*

(5*5)4*

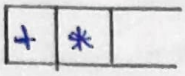
(25*4) = 100

Note:- It is easier for the compiler to compute Postfix Expression. It converts Infix to Postfix and then it Evaluates.

Note: Bigger Person can sit on Smaller Person.

① I/P:- a+b*c.

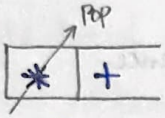
→ If operand simply add to o/p:



O/P: abc*+

→ If the incoming operator has a higher precedence than the stack peek then push it on the stack.

② a*b+c → O/P: ab*c+



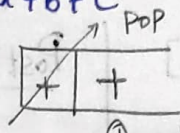
Note: If the incoming operator has a precedence lower than stack peek then pop the stack until the incoming precedence is greater.

3. ((a+b)+c)

A B

ab+ +c

I/P: a+b+c



O/P: ab+c+

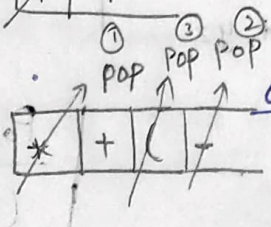
4. a*b+(c-d)

(a*b)+(c-d)

A B

(ab*)+(cd)-

ab*cd-+



O/P: ab*cd-+

Algorithm:

```
char ch = str.charAt(i);
```

→ If ch is operand

```
ans+ = ch;
```

→ If ch is '('

```
st.push(ch);
```

→ If ch is ')'

```
Pop until st.peek() == '('
```

→ If ch is operator.

```
if (stack is Empty){
    Push ch onto the stack }
else {
    //Bigger person can sit on Smaller person ⇒ Incoming Precedence
    has a higher precedence than stack's peek.
        st.push(ch);
} //Incoming operator has lower person. pop until the stack peek is
                                                    high.
```

Next Greater Element to the RHS:

I/P: | 18 | 7 | 6 | 12 | 15 | → O/P: | -1 | 12 | 12 | 15 | -1 |

Brute Force:

```
for (int i = 0; i < arr.length; i++) {
    for (int j = 0; j < arr.length; j++) {
        if (arr[i] < arr[j]) {
            arr[i] = arr[j]
        }
    }
}
```

O(n²)

Using Stack:

| 18 | 7 | 6 | 12 | 15 |

ansArray

| -1 | -1 | -1 | -1 | -1 |

→ Initially the stack is empty, push the index of first element

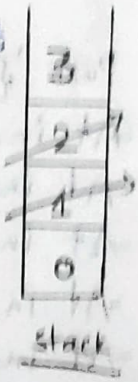
→ 7 < 18 (Peek element)
So push the index of 7 to stack.

→ 6 < 7 (Peek element)
So push the index of 6 to stack.

→ 12 ≠ 6 (Peek element)
Pop the indexs from the stack until the peek element > 12.
and update the array indices with 12.

→ 15 ≠ 12 (Peek element)

→ 15 < 18, So update the array index of 15 to -1.



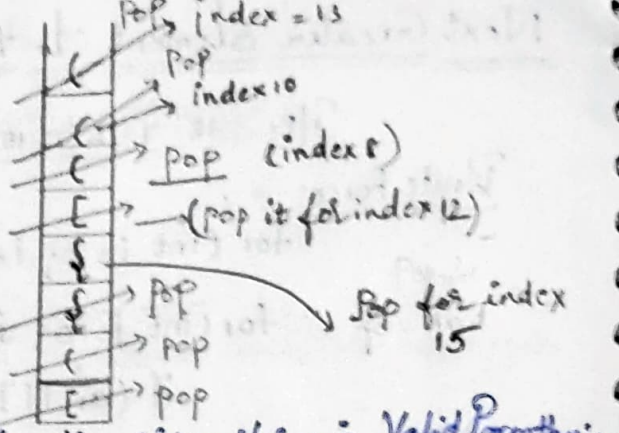
So, finally O/P: | -1 | 12 | 12 | 15 | -1 |

```
Stack<Integer> st = new Stack<>();
int[] ansArray = new int[A.length];
Arrays.fill(ansArray, -1);
for(int i = 0; i < A.length; i++){
    if(st.isEmpty()){
        st.push(i);
    }
    else{
        while(!st.isEmpty() && A[i] > A[st.peek()]){
            int index = st.pop();
            ansArray[index] = A[i];
        }
        st.push(i);
    }
}
return ansArray;
```

Valid Parenthesis:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

[( )] { } { [[( ) ( )] ( )]



→ When charAt(i) is '[' (or) '(' (or) '{' push it to stack.

→ When st.peek is having the corresponding open bracket then pop the bracket.

→ At last, check if the stack is empty then the given string is Valid Parenthesis

---

→ In the above string, moving from left to right, push [, (to stack.

→ We see ~~opening~~ closing bracket.

st.peek → (and char =)

So pop the element.

→ We see closing bracket.

st.peek → [and char =], so pop the element

→ Push '{' when index is at 4

→ At index 5, char is } and peek is '{' so, pop '{'

→ At index 6, char is {, push it to stack.

→ At index 7, char is [" ,

→ At index 8, char is (" ,

→ At index 9, we see closing bracket,

st.peek = '(' and char = ')' so pop it.

→ At index 10, we have opening bracket, push to stack.

→ At index 11, we see closing bracket

st.peek = '(' and char = ')', so pop it

→ At index 12, closing bracket, ']'

st.peek = '[', so pop it

→ At index 13, opening bracket, (, push it

→ At index 14, ')', st.peek = '(', so pop()

→ At index 15, closing bracket, '}', st.peek = '{'

→ At last, The stack is empty.

So This is a Valid Parenthesis.

Max Element:

10	1	2	25	4

→ Approach:

Store the max so far.

10	10	10	25	25

return the peek element

O/P: 25

1) 1 < 10
   So st.push(10)
2) 2 < 10
   So st.push(10)
3) 25 ≠ 10
   So st.push(25)
4) 4 < 25
   So st.push(25)

```
Stack<Integer> st = new Stack<>();
for (int i=0; i<arr.length; i++){
    if(st.isEmpty()){
        st.push(arr[i]);
    }
    else {
        if(st.peek()>arr[i]){
            st.push(st.peek());
        }
        else{
            st.push(arr[i]);
        }
    }
    return st.peek();
```

Stock Span Problem:-

arr[] = {100, 80, 60, 70, 60, 75, 85}
0 1 2 3 4 5 6

Span: max no. of days in the past such that price at i^{th} day > Price in past

→ max no. of days in the past such that price at 5th day > price at past days

75 > 60, 75 > 70, 75 > 60, 75 ≯ 80

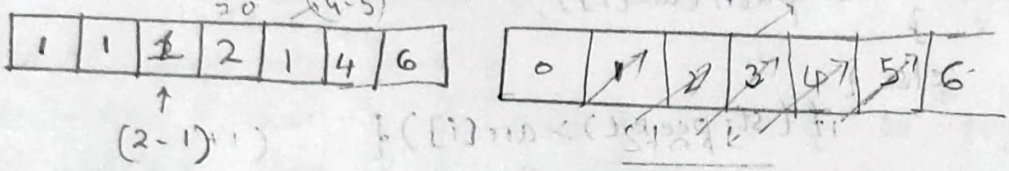
Span = 1 + (3) = 4

Span at 6th day:

85 > 75, 85 > 60, 85 > 70, 85 > 60, 85 > 80, 85 ≯ 100

Span = 1 + (5) = 6

O/P: | 1 | 1 | 1 | 2 | 1 | 4 | 6 |



Algo:

1. array[0] = 1, push(0)
2. If arr[i] > st.peek() then pop it until the condition is valid.
3. Calculate the Span by i - stack.peek() → if stack is not empty
   i + 1 → if stack is empty
4. stack.push(i)

Duplicate Parenthesis
Simplify path
Remove duplicate from string

1) Push '0' to stack, arr[0] = 1 | 0 |

2) At index: 1 arr[1] = 80 < 100, Span = 1 - 0 = 1, push(1) | 0 | 1 |

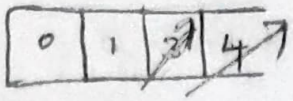
3) " : 2 arr[2] = 60 < 80, Span = 2 - 1 = 1, push(2) | 0 | 1 | 2 |

4) " : 3 arr[3] = 70 ✗ arr[st.peek]
= 70 ✗ arr[2] ⇒ 70 > 80, So pop(2) | 0 | 1 |
Span = 3 - 1 = 2, push(3) | 0 | 1 | 3 |

5) " : 4 arr[4] = 60, arr[3] ⇒ 60 < 70
Span = 4 - 3 = 1, push(4) | 0 | 1 | 3 | 4 |

6) " : 5 arr[5] = 75, arr[4] ⇒ 75 > 60

pop the stack until 75 > arr[peek]



So 75 > arr[4]

75 > 60, So pop → 4

Span = 5 - 1 = 4

75 > arr[3]

75 > 70, So pop → 3

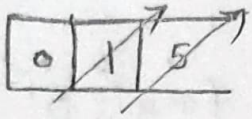
Push (5)

75 ≯ arr[1]

75 ≯ 80.

At index 6:

85 ≯ arr[5]



So pop

85 > arr[1]

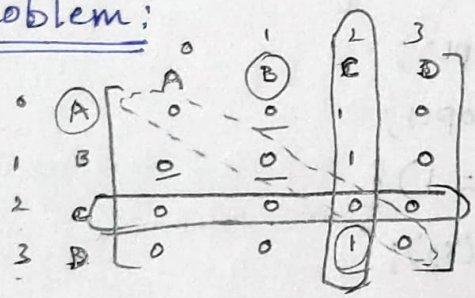
85 > 80, So pop

85 , arr[0]

85 < 100

So span = 6 - 0 = 6

Celebrity Problem:



A doesn't B

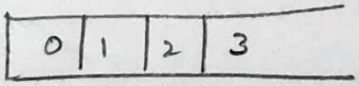
A knows C

B knows C

D knows C

→ A celebrity is a person who doesn't know anyone & who is known by all.

→ In the above I/P: 'c' doesn't know anyone, but all others know 'c'.



Stack

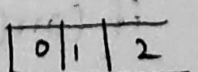
1) Pop 2 elements, $x_1 = 3$, $x_2 = 2$

arr[3][2] = 1

'3' knows '2' ⇒ So 3 is not celebrity

'2' might be a celebrity,

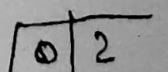
Push '2' into the stack



2) $x_1 = 2$, $x_2 = 1$

arr[2][1] = 0 ⇒ '2' can be Celebrity

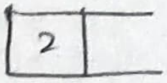
push 2 to stack



3) $x_1 = 2$, $x_2 = 0$ ⇒ arr[2][0] = 0

→ 2 does not know 0

So push '2'



0, 2
1, 2 ⇒ 1
3, 2

2, 0
2, 1 ⇒ 0
2, 3

break if any is '0'

break if any is '1'

```
→ for(int i = 0; i<arr.length; i++){
        stack.push(i);
  }

while(stack.size()>=2){
  int x1 = stack.pop();
      x2 = stack.pop();
  if(arr[x1][x2] == 1){
      stack.push(x2);
    }
  else
    {
     stack.push(x1);
    }
}
```

```
int flag = 0;
int ans = stack.pop();  → 2
for(int i=0; i<arr.length; i++){
  if(i != ans){
   if(arr[i][ans] == 0 || arr[ans][i]==1)
    {  flag = 1;
       break;
    }
}}
```

if (flag == 0)
ans it is Celebrity

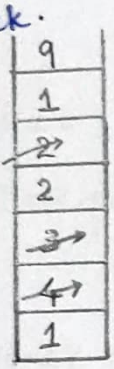
Remove K Digits:

Given string num representing a non-negative integer num, and an integer k, return the smallest possible integer after removing k digits

1. I/P: 1432219, k=3:
   O/P: 1219
2. I/P: 10200, k=1
   O/P: 200
3. I/P: 10, k=2, O/P: 0

→ Using Stack:

→ check the prev element (top element) is greater, if yes remove from stack.



stack

k=3, num="1432219"

→ At index 0: Stack is Empty
push '1' to stack.

→ At index 1: 4 > 1
So push 4 to stack

→ At index 2: 3 < 4 (top element)
So pop 4 and k--; k=2, and push '3'

→ At index 3: 2 < 3
So pop 3 and k--; k=1 and push '2'

→ At index 4: 2 = 2
So push 2 to stack

→ At index 5: 1 < 2
So pop 2 and k--
k=0 and push 1

→ At index 6: 9 > 1
So push 9

→ So the number is 1219

```
if(k >= num.length()) return "0"
Stack<Integer> st = new Stack<>();
for(int i=0 ; i<num.length(); i++){
 while(k >0 && !st.isEmpty() && st.peek() >num.charAt(i))
   { st.pop(); k--; }
    st.push (num.charAt(i));
}
while( k>0 && !st.isEmpty()){
   st.pop(); k--;
}
```

```
StringBuilder sb = new StringBuilder();
  while (!st.isEmpty()){
    sb.insert(0, st.pop());
  }
// Remove leading zeros.
  int i=0;
while( i<sb.length() && sb.charAt(i)=='0')
  { i++;
  }
return ans.subString(i);
        (sb)
```