Walk in the Park

Ended | Edit

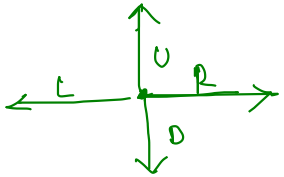
Description

Given a 2d array with 'U', 'D', 'L', 'R' representing up, down, left, right => count the number of steps you have taken before reaching an already visited cell or going outside the 2d array.

Note: The first step is always taken to cell located at (0,0).

Input

The first line of the input contains T, the number of test cases.



The first line of each test case contains N, and M, the number of rows, and columns in the 2D Array.

The next lines of the test case, contains M characters each, denoting the values in the 2D array.

Constraints

1 <= T <= 10

1 <= N,M <= 100

Output

Print a single integer, denoting the number of steps, you take before stepping out of the 2D array, or visiting an already visited cell, on a new line.

Output

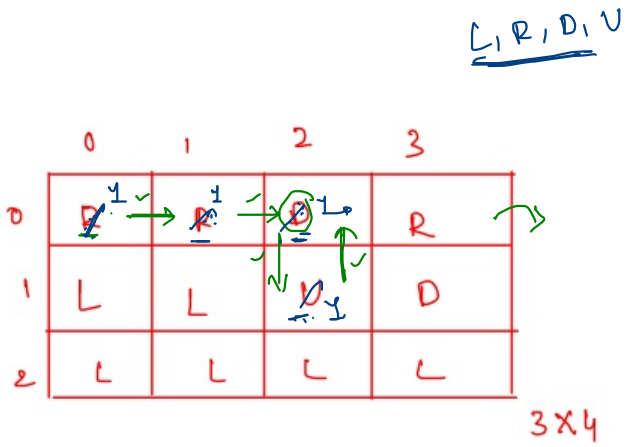
Print a single integer, denoting the number of steps, you take before stepping out of the 2D array, or visiting an already visited cell, on a new line.

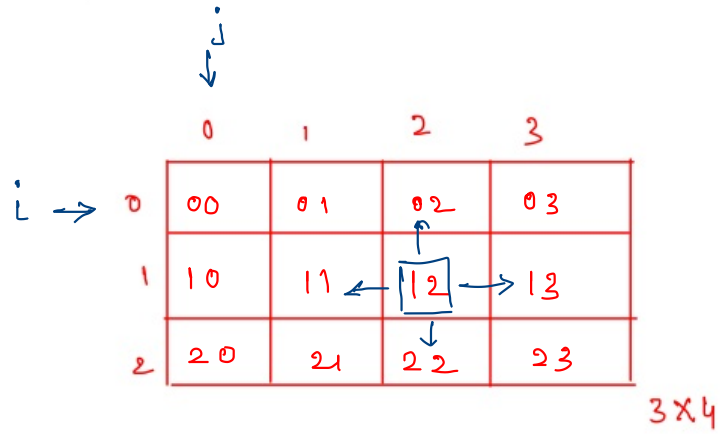
Sample Input 1

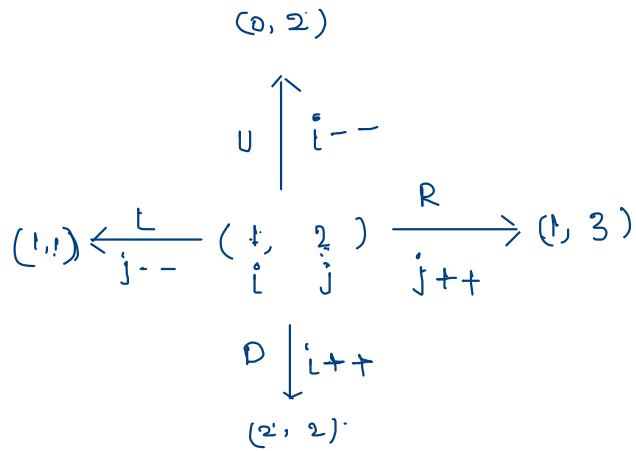
```
1
3 4
RRDR
LLUD
LLLL
```

Sample Output 1

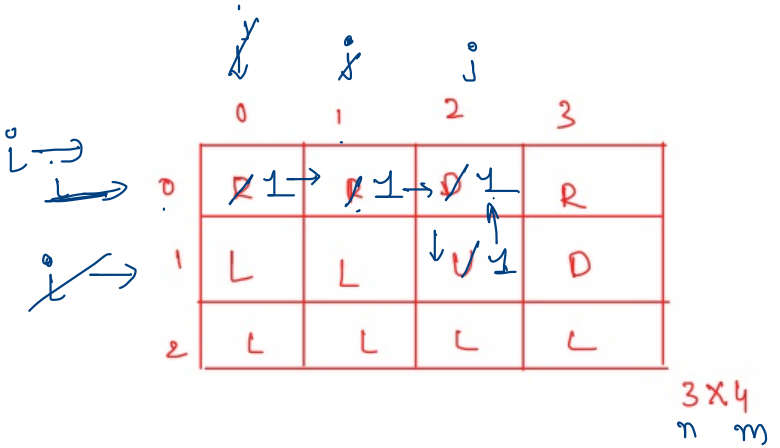
```
4
```

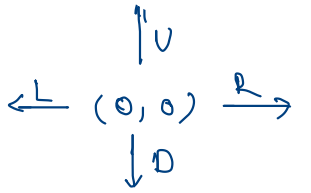


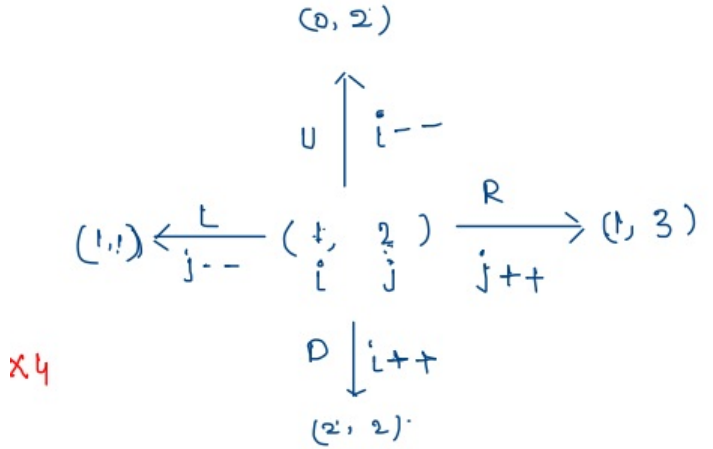




```
function countSteps(arr[][],n,m)
{
        i=0,j=0,count=0
        while(i>=0 && i<n && j>=0 && j<m)
        {
                if(arr[i][j]==1)
                        return count
                temp=arr[i][j]
                arr[i][j]=1
                if(temp=='L')
                        j--
                else if(temp=='R')
                        j++
                else if(temp=='U')
                        i--
                else
                        i++
                count++
        }
        return count
}
```







Repeated and Missing

Ended | Edit

Description

Given an array of size n. Array elements are in the range from 1 to n. In the given array, one number from set {1, 2, …n} is missing and one number occurs twice in the array. Find these two numbers.

Print the missing one first and then the repeated one with a space character in between them.

Input

The first line of the input contains one integer t (1 ≤ t ≤ 10) — the number of test cases. Then t test cases follow.

The first line of each test case contains a single integer n (2 ≤ n ≤ 100000).

The second line of the test case contains n integers (1 ≤ Ai ≤ n).

Output

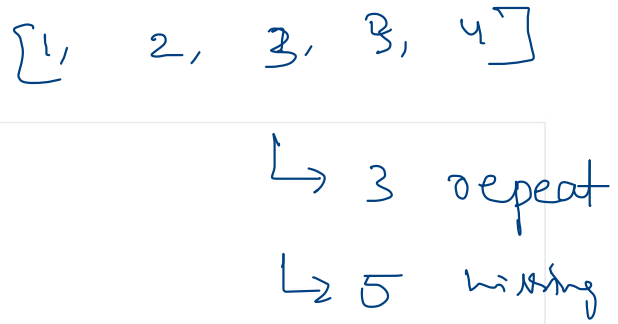
For each test case, print the answer.

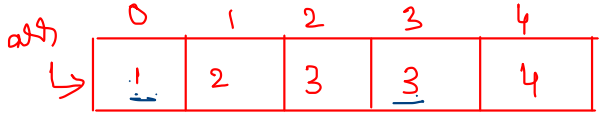
Sample Input 1

```
3
5
1 2 3 3 4
2
1 1
3
1 2 2
```

Sample Output 1

```
5 3
2 1
3 2
```





n = 5

Set: { 1, 2, 3 }

.has( )

dup = 3 ✓

$n \cdot \log n$ ⇒ Sort + loop } duplicates

Step 1:- Set / Key + value (Obj) ⇒ dup = 3 ✓

Step 2:- $sum\_n = \frac{n(n+1)}{2}$ ⇒ 1 + 2 + 3 + 4 + 5

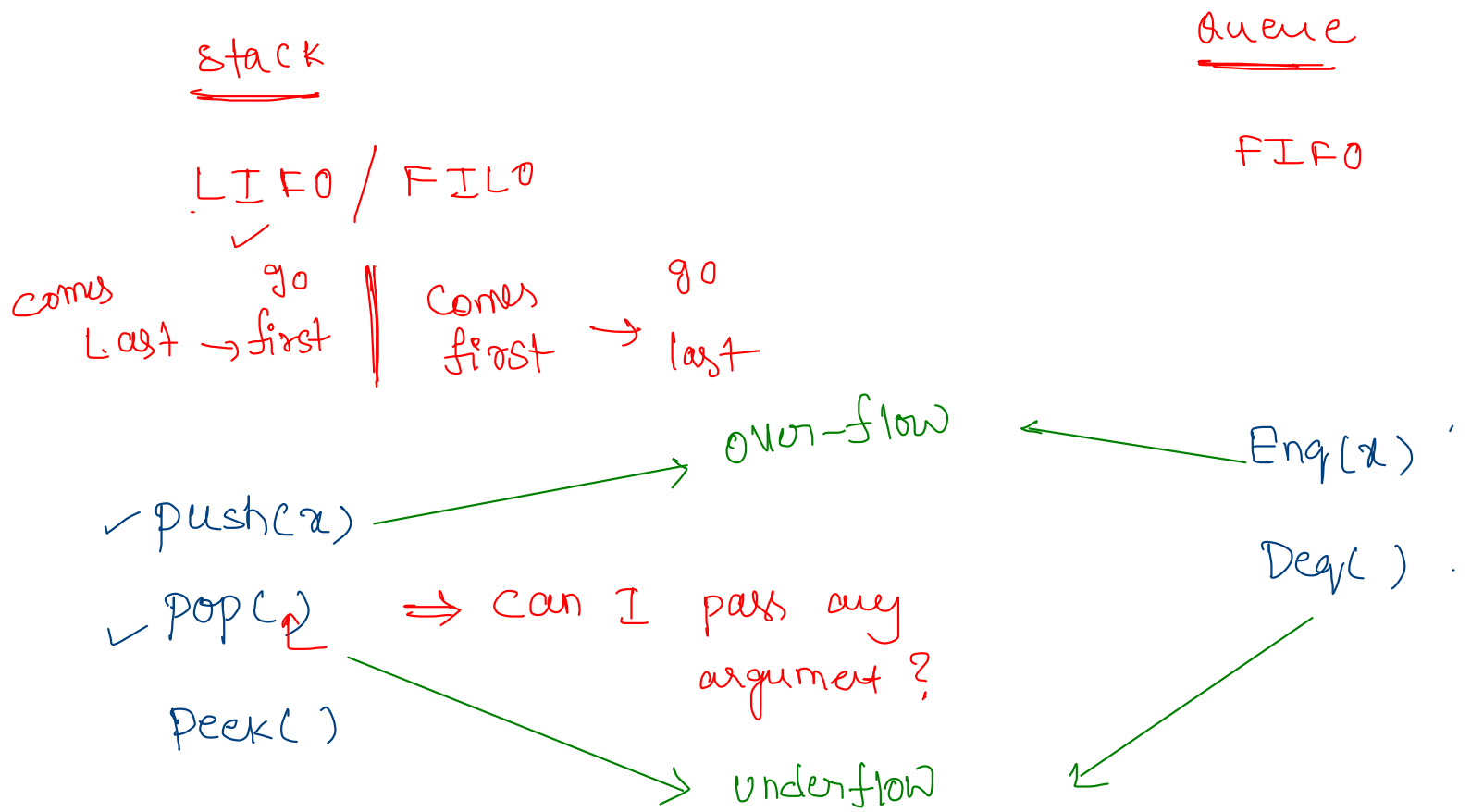
Step 3:- sum_arr = 1 + 2 + 3 + 3 + 4 (sum of all elements of the given array)

Step 4:- sum_arr = sum_arr − dup ⇒ 1 + 2 + 3 + 4

Step 5:- missing = (sum_n) − (sum_arr) ⇒ 1 + 2 + 3 + 4 + 5 − 1 − 2 − 3 − 4 = 5

O(n)

✓ Stack & Queue


Stack
LIFO / FILO
comes Last → go first
Comes first → go last
Queue
FIFO
push(x)
pop()
peek()
over-flow
Enq(x)
Deq()
⇒ can I pass any argument ?
underflow

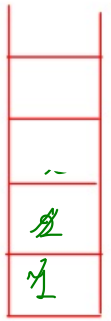
Which of the following permutations can be obtained in the output (in the same order) using a stack assuming that the input is the sequence 1, 2, 3, 4, 5 in that order?

A. 3, 4, 5, 1, 2

B. 3, 4, 5, 2, 1

C. 1, 5, 2, 3, 4

D. 5, 4, 3, 1, 2



3 4 5 2 1

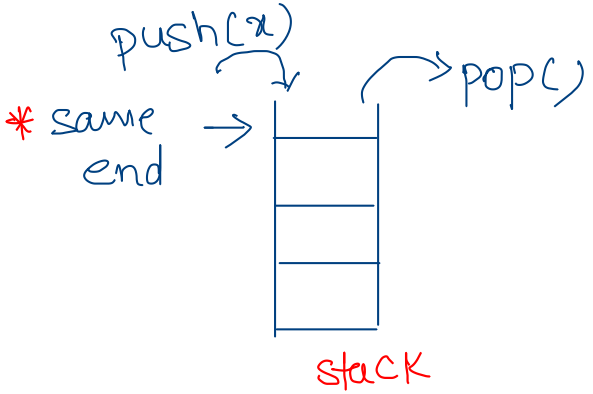
None

Stack [LIFO / FILO]

→ ① push(x) → overflow

② pop( ) → underflow

→ both operations, will happen on the same end

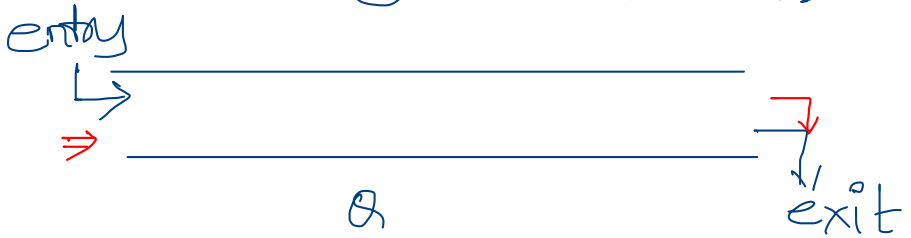


Queue (FIFO)

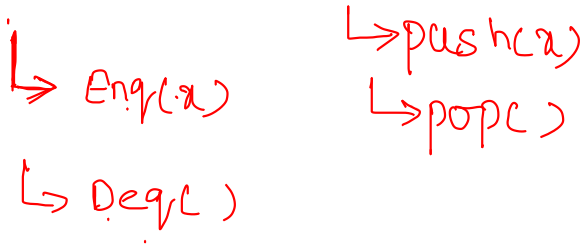
① Enq(x) → OF

② Deq( ) → UF

→ both operations will happen @ different ends.



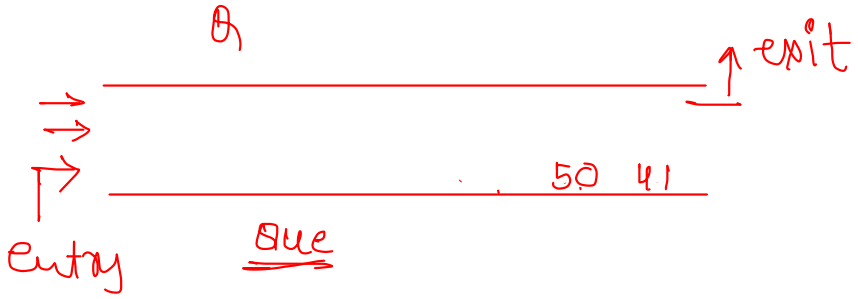
Implement Queue using Stacks



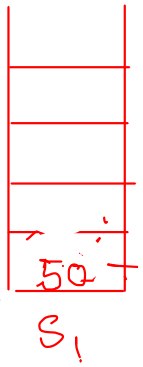

```
Enq(x)
{
    push(x) | pop()
}

Deq()
{
    push(x) | pop()
}
```

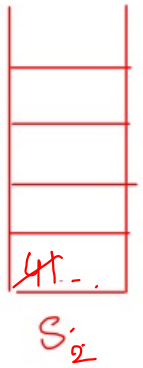
E(10), E(20), D(), E(21), E(41), D(), D(), E(50), D()

10 → 20 → 21 → 41



2-ptr:- stack





Enq(x) :-

S_1.push(x)

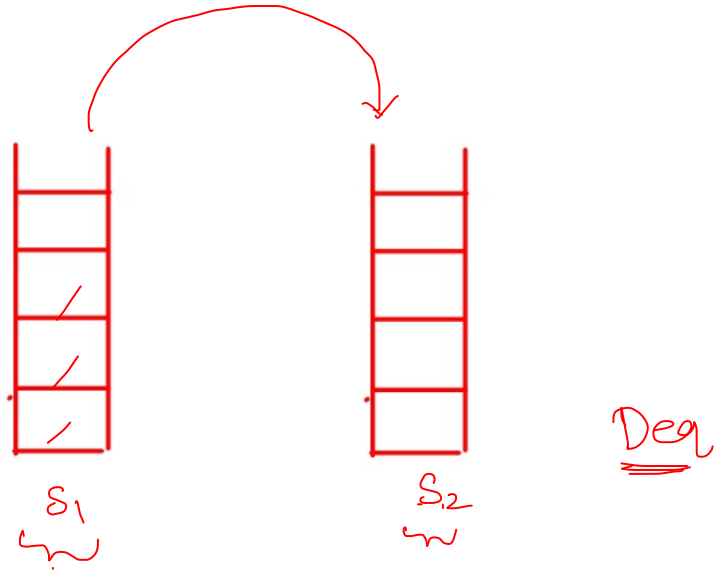
DC) :-

1. [move all ele's $S_1 \rightarrow S_2$] ⇒ when S_2 'isEmpty

2. S_2.pop() ✓

PS-code

```
void enqueue(Q,x)
{
      push(S1,x)
}

dequeue(Q)
{
  if(S2.isEmpty())
    {
      if(S1.isEmpty())
        {
           print("Q is empty")
           return
        }
        else
        {
          while(!S1.isEmpty())
          {
             x=S1.pop()
             S2.push(S2,x)
          }
        }
     }
     x=S2.pop()
     return x
}
```



Deq

O(1)

4

O(1)

- Implement Stack using Queues

↳ push(x)

↳ pop()

↳ enq(x)

deq()

→ push(x)

{

Enq(x) | Deq()

}

→ pop()

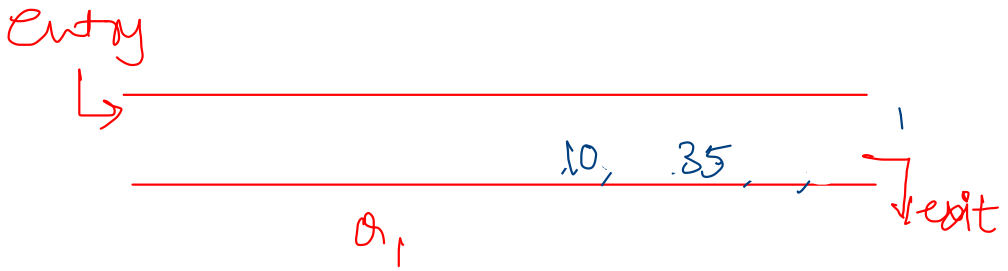
{

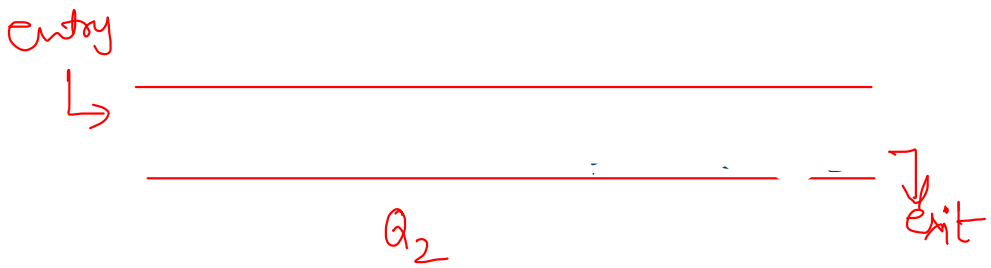
Enq(x) | Deq()

}

push(10), push(20), push(30), pop(), pop(), push(35), push(40), pop()

30 20 40





push(x):-

1. add to Q_2
2. $Q_1 \rightarrow Q_2$
3. Swap(Q_1, Q_2)

pop():-

Q_1.deq()

```
Let Q1,Q2 be two Queues
size=0

void push(x)
{
    1. Q2.enqueue(x)
    2. while(!Q1.isEmpty())
       {
            Q2.enqueue(Q1.front))
            Q1.dequeue()
        }
    3. let temp be a Queue
       temp=Q1
       Q1=Q2
       Q2=temp
}

pop()
{
    return Q1.dequeue()
}

top()
{
    return Q1.front()
}
```

Q1 → Q2

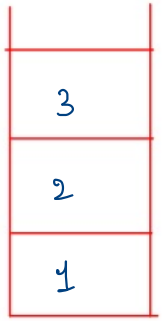
swap (Q1, Q2)

⇒ f, g

11:05

Reverse the contents of Stack

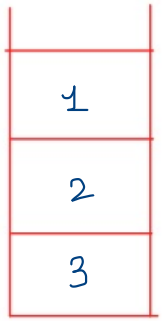
2-ptr tech + swap() } ⇒ array[]



S

after reversing

?



S

Ap:—

→ use another stack (S_2)

S_1: input()

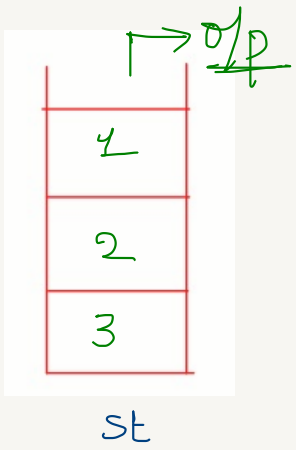
S_2: move all $S_1 \rightarrow S_2$

O(n) T.C

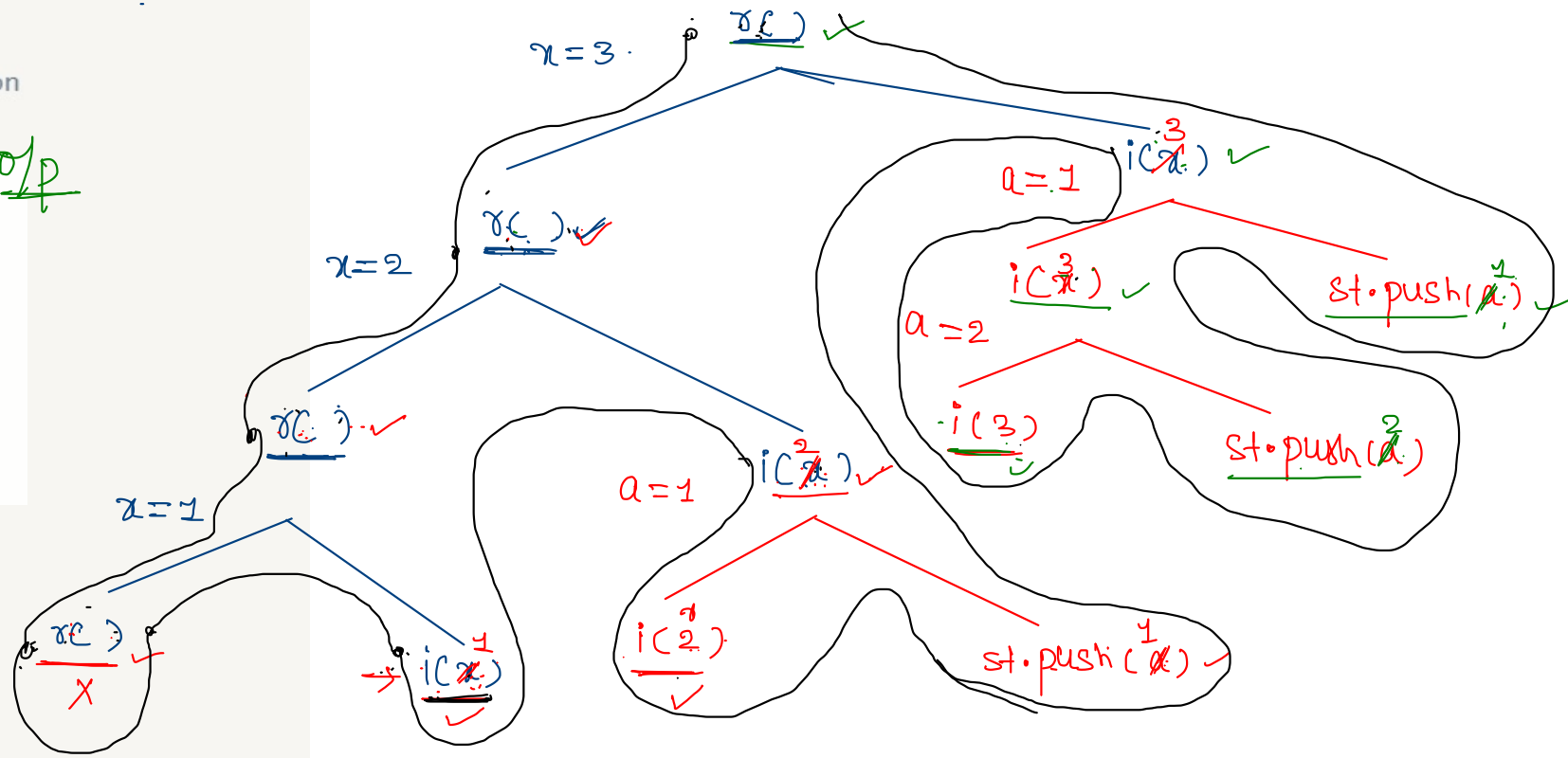
O(n) S.C

```
Declare a global stack called st
main()
{
   → add some elements to stack
     reverse() // call the reverse function
}
reverse()
{
  → if(st.size>0)
    {
        1. x=st.peek()
        2. st.pop()
        3. reverse()
        4. insert(x)
    }
}

insert(x)
{
    if(st.isEmpty())
        st.push(x)
    else
    {
        1. a=st.peek()
        2. st.pop()
        3. insert(x)
        4. st.push(a)
    }
}
```

peek() v/s pop()



recursion - tree



n+ → i(1) → 1
i(2) → 2
i(3) → 3

$n + n^2 \Rightarrow O(n^2)$

$1+2+3$

$1+2+3+\dots+n = \frac{n(n+1)}{2}$

$O(n^2)$

log n
2

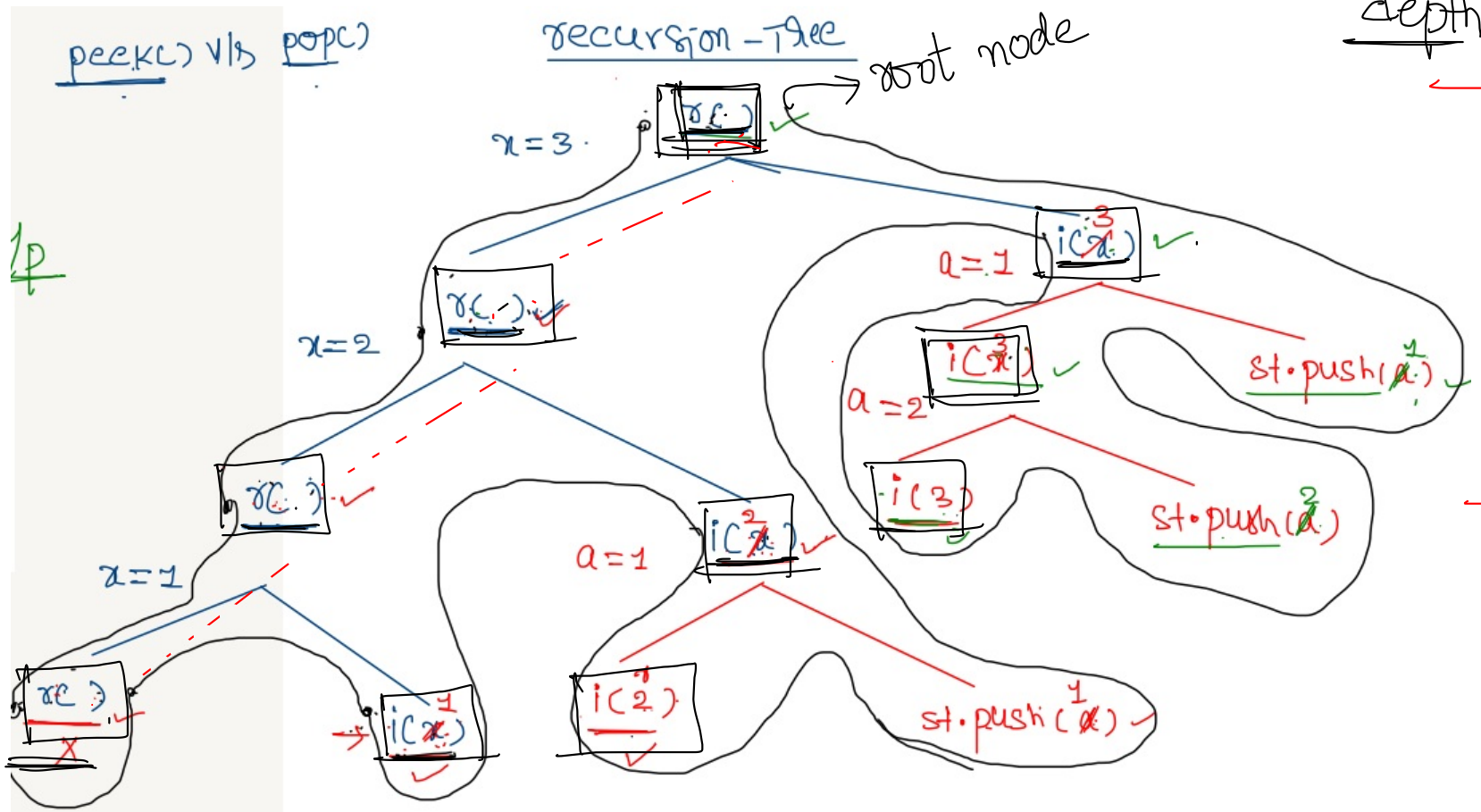
n

2^n

n^2 not sure.

O(1)

peek() v/s pop()



3 → edges [root → last level

n ⇒ O(n) S.C ; T.C : $O(n^2)$

Rec.

↓

R.T

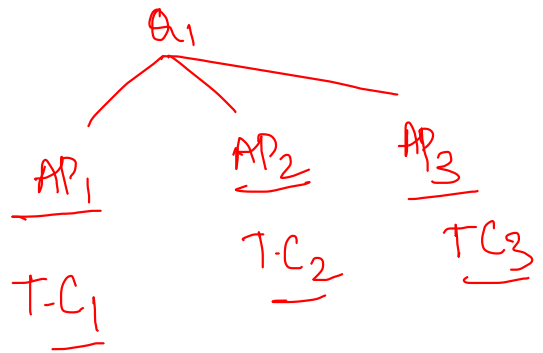
T.C — # of fn calls

S.C — max depth

Time and Space Complexity

1) if code is given, then what is the T.C of that code

2) if two Time Complexites are given, then you should know which one is best



	Notation	Name
1.	$O(1)$	constant
2.	$O(\log \log n)$	double logarithmic
3.	$O(\log n)$	logarithmic
4.	$O((\log n)^c)$ $c > 1$	polylogarithmic
5.	$O(n^c)$ $0 < c < 1$	fractional power
6.	$O(n)$	linear
X 7.	$O(n \log^* n)$	*n* log-star *n*
8.	$O(n \log n) = O(\log n!)$	linearithmic, loglinear, quasilinear, or "*n* log *n*"
9.	$O(n^2)$	quadratic
10	$O(n^c)$	polynomial or algebraic
X	$L_n[\alpha, c] = e^{(c+o(1))(\ln n)^\alpha (\ln \ln n)^{1-\alpha}}$ $0 < \alpha < 1$	L-notation or sub-exponential
** 11.	$O(c^n)$ $c > 1$	exponential
** 12.	$O(n!)$	factorial

Θ ⇒ Big-oh

$\log_2 n$ ↕ Algo; (By-default)

$c > 2$ n^2, n^3, n^4 → n^2, n^3, n^4 ... (poly)

⇒ 2^n, 3^n, 4^n → 2^n, 3^n, 4^n ... (expo)

steps to decide which T.C is better?

1) cancel the common terms [if present]
2) apply log to the both T.C's given
3) put some very very larger values of n

then see which is having less values, then that is better [it will take less time, it means runs faster]

lesstime ⇒ better.

$T.C_1 \Rightarrow n^2$

$n * n$

n

$TC_2 \Rightarrow n \cdot \log_2^n$

$n \cdot \log_2^n$

$\log_2^n$

let $n = 2^{1024}$

2^{1024}

$= \log_2^{2^{1024}} = 1024 \cdot \log_2^2 = 1024 \cdot 1 = 1024$

$TC_2 < TC_1$ ∴ TC_2 is better

$TC_1 : \sqrt{n}$	$TC_2 : \log_2 n$
$\sqrt{n} = n^{1/2}$	$\log_2 n$
$n = 2^{1024}$	$\log_2 2^{1024}$
$= \left[2^{1024}\right]^{1/2}$	$1024 \cdot \frac{\log_2 2}{1}$
$= 2^{512}$	1024

$TC_2 < TC_1 \Rightarrow TC_2$ is better

$TC_1 : n \cdot \log_2 n$

$n \cdot \log_2 n$

n

let $n = 2^{1024}$

2^{1024}

$TC_2 : \sqrt{n} \cdot \log_2 n$

$\sqrt{n} \cdot \log_2 n$

$\sqrt{n} = n^{1/2}$

$2^{1024 \cdot 1/2} = 2^{512}$

$TC_2 < TC_1 \Rightarrow TC_2$ is better.

1

Consider the following three functions.

$f_1 = 10^n$ $f_2 = n^{\log n}$ $f_3 = n^{\sqrt{n}}$

Which one of the following options arranges the functions in the increasing order of asymptotic growth rate?

A f_3, f_2, f_1

B f_2, f_1, f_3

C f_1, f_2, f_3

D f_2, f_3, f_1

1st < 2nd < 3rd

Small Small

Consider the following functions from positive integers to real numbers:

$10, \sqrt{n}, n, log_2 n, \frac{100}{n}$.

The CORRECT arrangement of the above functions in increasing order of asymptotic complexity is:

A $log_2 n, \frac{100}{n}, 10, \sqrt{n}, n$

B $\frac{100}{n}, 10, log_2 n, \sqrt{n}, n$

C $10, \frac{100}{n}, \sqrt{n}, log_2 n, n$

D $\frac{100}{n}, log_2 n, 10, \sqrt{n}, n$

Which of the given options provides the increasing order of asymptotic complexityoffunctions f1, f2, f3 and f4?

$f_1(n) = 2^n;$
$f_2(n) = n^{3/2};$
$f_3(n) = nlog_2n;$
$f_4(n) = n^{log_2n}$

A f_3, f_2, f_4, f_1

B f_3, f_2, f_1, f_4

C f_2, f_3, f_1, f_4

D f_2, f_3, f_4, f_1