DEC 28:

Find all factorial numbers less than or equal to N

Basic Accuracy: **48.65%** Submissions: **10K+** Points: **1**

A number **N** is called a factorial number if it is the factorial of a positive integer. For example, the first few factorial numbers are 1, 2, 6, 24, 120, Given a number N, the task is to return the list/vector of the factorial numbers smaller than or equal to N.

Example 1:

```
Input: N = 3
Output: 1 2
Explanation: The first factorial number is
1 which is less than equal to N. The second
number is 2 which is less than equal to N,
but the third factorial number is 6 which
is greater than N. So we print only 1 and 2.
```

Example 2:

```
Input: N = 6
Output: 1 2 6
Explanation: The first three factorial
numbers are less than equal to N but
the fourth factorial number 24 is
greater than N. So we print only first
three factorial numbers.
```

Your Task:

You don't need to read input or print anything. Your task is to complete the function **factorialNumbers**() which takes an integer N as an input parameter and return the list/vector of the factorial numbers smaller than or equal to N.

Expected Time Complexity: O(K), Where K is the number of factorial numbers.
Expected Auxiliary Space: O(1)

Constraints:
$1<=N<=10^{18}$

Code section:-

```
class Solution
{
public:
    vector<long long> v;

    long long fact(long long i){

        if(i==1  || i==0){
            return 1;
        }
        return i*fact(i-1);
    }

    void factnumber(long long N){

        long long x;
        for(long long i=1;i<=(N/2)+1;i++){

            x=fact(i);
            if(x<=N){
                v.push_back(x);
            }
            else{
                return;
            }
        }
    }
    vector<long long> factorialNumbers(long long N)
    {
        // Write Your Code here
       factnumber(N);
       return v;
    }
};
```

2nd Question:-

Implement Queue using Linked List

Basic Accuracy: **45.6%** Submissions: **70K+** Points: **1**

Implement a Queue using Linked List.
A Query **Q** is of 2 Types
(i) 1 x (a query of this type means pushing **'x'** into the queue)
(ii) 2 (a query of this type means to pop an element from the queue and print the poped element)

Example 1:

```
Input:
Q = 5
Queries = 1 2 1 3 2 1 4 2
Output: 2 3
Explanation: n the first testcase
1 2 the queue will be {2}
1 3 the queue will be {2 3}
2   poped element will be 2 the
    queue will be {3}
1 4 the queue will be {3 4}
2   poped element will be 3.
```

Example 2:

```
Input:
Q = 4
Queries = 1 2 2 2 1 3
Output: 2 -1
Explanation: In the second testcase
1 2 the queue will be {2}
```

```
2   poped element will be {2} then
    the queue will be empty.
2   the queue is empty and hence -1
1 3 the queue will be {3}.
```

Your Task:
Complete the function **push()** which takes an integer as input parameter and **pop()** which will remove and return an element(-1 if queue is empty).

Expected Time Complexity: O(1).
Expected Auxiliary Space: O(1).

Constraints:
1 <= Q <= 100
1 <= x <= 100

Code Section :-

```
//2nd question


/* Structure of a node in Queue
struct QueueNode
{
    int data;
    QueueNode *next;
    QueueNode(int a)
    {
        data = a;
        next = NULL;
    }
};

And structure of MyQueue
struct MyQueue {
    QueueNode *front;
    QueueNode *rear;
    void push(int);
    int pop();
    MyQueue() {front = rear = NULL;}
}; */
```

```
void MyQueue:: push(int x)
{
        // Your Code

        QueueNode *t=new QueueNode(x);

if(t==NULL){
    cout<<" queue is full "<<endl;
}

else{

    // t->data=data;
    // t->next=NULL;
    if(front==NULL){front=rear=t;}
    else{
         rear->next=t;
         rear=t;
}

}




}

//Function to pop front element from the queue.
int MyQueue :: pop()
{
        // Your Code

         int x=-1;
    if(front==NULL){



    }else{

      QueueNode *p=front;
      front=front->next;
      x=p->data;
      delete p;
    }
      return x;


}
```