A stack can be uniquely identified by an array and a top variable. Some common operations on a stack includes:

- bool isFull() - returns true if the stack is full.
- bool isEmpty() - returns true if the stack is empty.
- void push(x) - inserts an element x at the top of the stack.
- void pop() - deletes the top element of the stack.
- int top() - returns the top element of the stack.

The following two situations can arise with a static array implementing a stack.

- Overflow: when top=N-1, It will lead to an overflow in case of insertion in a stack of size N.
- Underflow: when top=-1, It will lead to underflow in case of deletion.

Note: In-built stacks in language libraries are expandable and thus overflow cases are rare. However, we need to handle underflow cases cautiously.

We have been given an integer array Arr[N]. We have to find the next greater element of every array element.

Note: Next greater element of an element is the first element which is strictly greater than it and is towards its right.

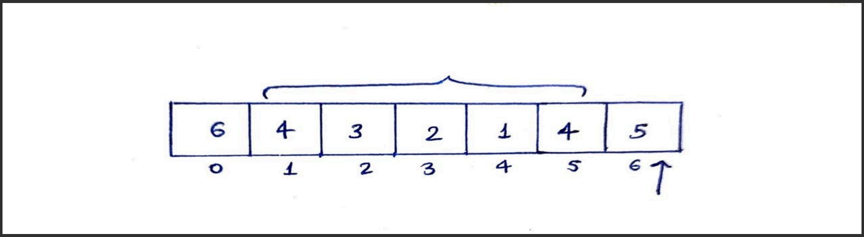
Input: Arr[4] = {4, 5, 2, 25}

Output: NGE[4] = {5, 25, 25, NULL}

Approach:

1 **Brute Force** - Create two nested loops to find the next greater element for each element respectively.
Time complexity: **O(N^2)**
Space complexity: **O(1)**

2 In the example given below, 5 is the next greater element (NGE) of all the elements from index 1 to 5. Taking a hint from this, can we find an efficient method of assigning 5 as the NGE for the elements from index 5 to 1?



We can utilise a stack for this purpose. We can traverse the array and compare Arr[i] with the stack top. We can assign Arr[i] as the NGE of all the elements it is greater than and then pop them out of the stack. After which we can push Arr[i] into the stack.

Time complexity: **O(N)**
Space complexity: **O(N)**

Note: The nature of the stack will always be monotonic-decreasing from bottom to top.

We have been given the height of ‘N’ histograms of width 1 each. Find the area of the largest rectangle.

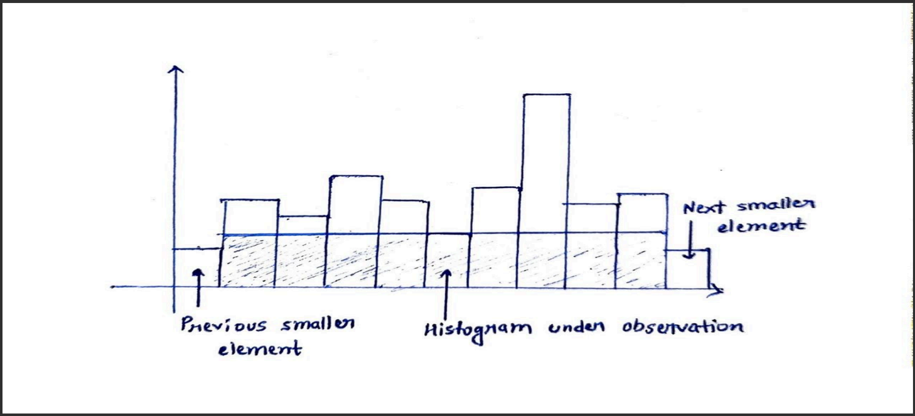
Input: ht[7] = {6, 2, 5, 4, 5, 1, 6}
Output: 12

Approach:

If you think Intuitively, the height of the largest area rectangle will be equal to the height of one of the histograms.

1 **Brute Force** - Create two nested loops to consider all possible rectangles for each histogram and keep the track of maximum area rectangle in an answer variable.
Time complexity: **O(N^2)**
Space complexity: **O(1)**

2 Using the concept of Next smaller element (NSE) and Previous smaller element (PSE)



Using stacks we can precompute NSE and PSE for each histogram. After which we can traverse the histogram array to consider the largest area rectangle formed by that particular histogram and compare it with the maximum area variable.
area = ht[i]*(NSE[i]-PSE[i])

Note:

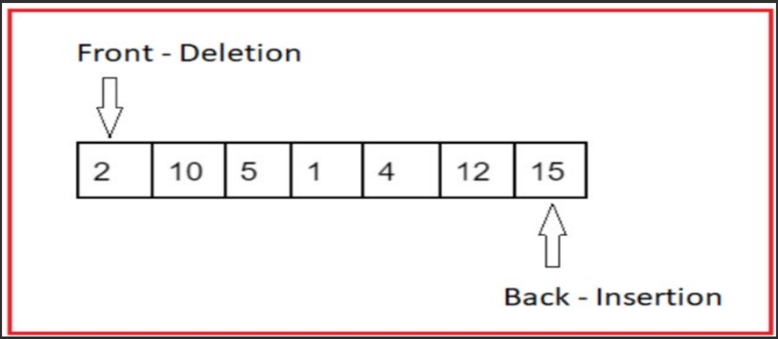
1. *We have stored the index and not the element in the NSE[ ] and PSE[ ] array.*
2. *For elements having no next smaller element, we can assign them N.*
3. *For elements having no previous smaller element, we can assign them -1.*

Time complexity: **O(N)**
Space complexity: **O(1)**

2,3

QUEUES

A queue is a linear data structure that follows the FIFO (First In First Out) principle. It is a two-ended data structure with different ends for insertion(back) and deletion(front).

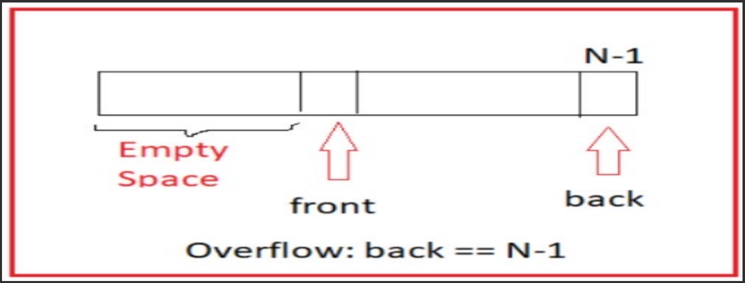


It can be implemented with the help of an array and two variables - front and back. We increment the front variable for deletion and back for insertion. For an array of size N.

Underflow: front > back

Overflow: back == N-1

But it may not be the most efficient implementation of the queue as it may show overflow even when space might be left in the array.



Given an integer N>0. Print the binary representation of all numbers from 1 to N.

Input: N = 5

Output: 1, 10, 11, 100, 101

Approach:

1. **Brute Force** - We can iterate from 1 to N and produce the binary forms of each one of them respectively.
   Time complexity: **O(NlogN)**
   Space complexity: **O(1)**
2. **Using Level Order Traversal** - Can you observe any pattern in the binary representation of numbers?

Level Order Traversal

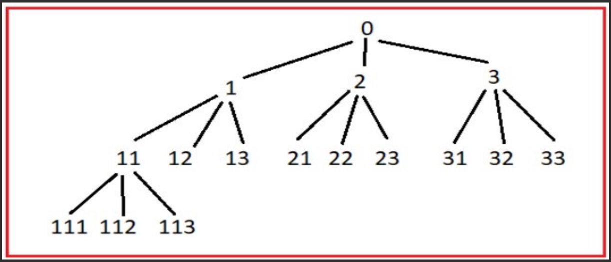
Given a natural number 'N'. We have to print the first 'N' natural numbers whose digits ∈ {1, 2, 3} in a sorted form.

Input: N = 10

Output: 1, 2, 3, 11, 12, 13, 21, 22, 23, 31

Approach:

- **Thinking recursively** - If we have a number 'num' that satisfies the given criteria then we can create the next three numbers [num*10+1, num*10+2, num*10+3] using recursion.



But will this approach work?
No, we will not get the numbers in a sorted manner because recursion works in a depth-based manner.

- **Level Order Traversal** - If you observe carefully, then we can get the elements in a sorted form if we move horizontally.

  Can you think of a data structure that can be used for such a purpose?

  We can use a queue for level order traversal. We can insert 1, 2 and 3 and then print 1, delete 1 while simultaneously inserting [1*10+1, 1*10+2, 1*10+3]. Similarly, we can get the first N natural numbers containing {1, 2, 3} and in a sorted form.
  Time complexity: **O(N)**
  Space complexity: **O(N)**

  Note:
  - *In the worst case, our Queue may store extra elements from the next level even after printing the first N natural numbers. This situation can be optimised using a few checks in the code.*
  - *Will level order traversal always produce output in sorted order?*
    No, it will not. Eg. Consider all numbers with a prime factorisation of the form 2^x.3^y.5^z.

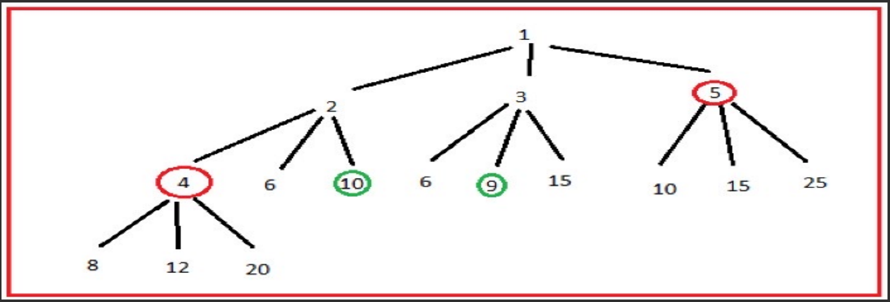
But will this approach work?
No, we will not get the numbers in a sorted manner because recursion works in a depth-based manner.

- **Level Order Traversal** - If you observe carefully, then we can get the elements in a sorted form if we move horizontally.

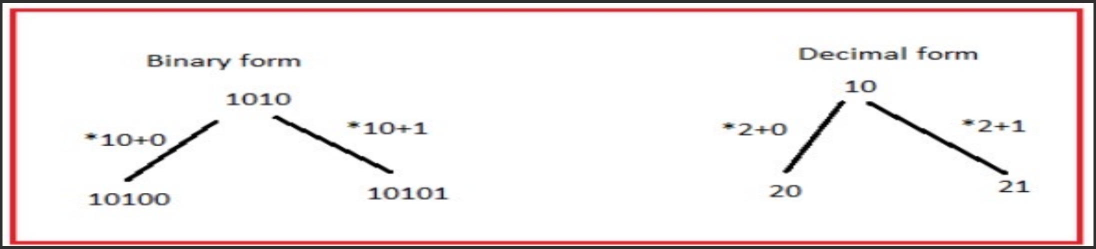
  Can you think of a data structure that can be used for such a purpose?

  We can use a queue for level order traversal. We can insert 1, 2 and 3 and then print 1, delete 1 while simultaneously inserting [1*10+1, 1*10+2, 1*10+3]. Similarly, we can get the first N natural numbers containing {1, 2, 3} and in a sorted form.
  Time complexity: **O(N)**
  Space complexity: **O(N)**

  Note:
  - *In the worst case, our Queue may store extra elements from the next level even after printing the first N natural numbers. This situation can be optimised using a few checks in the code.*
  - *Will level order traversal always produce output in sorted order?*
    No, it will not. Eg. Consider all numbers with a prime factorisation of the form 2^x.3^y.5^z.



Using Level Order Traversal - Can you observe any pattern in the binary representation of numbers?



Therefore, if we can store the binary representations in an integer then we can easily produce the binary forms of all the numbers from 1 to N using level order traversal.
Time complexity: **O(N)**
Space complexity: **O(N)**

Note:

1. *In the worst case, our Queue may store extra elements from the next level even after printing the first N natural numbers. This situation can be optimised using a few checks in the code.*
2. *We may not be able to store the large binary representations in an integer or a long variable, therefore we may have to use strings for the purpose.*

_ _ _ _ _ _ _ _. -> n indexes
function(int n){
there is a code . o(n)
return function(n/2)+1;

T(n) = T(n/2)+n;

t(n) = t(n-1)+1;

O(n)

```
                  0
             /        \
01 00
/\
010 011  001 000
```

1,2,3,4,5

5 —> 2^5

0 —> 2^5-1

00000 —> null set

00001—>{5}

00010—>{4}

00011

…

11111