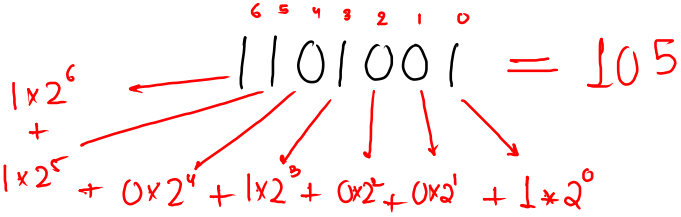
Queue Syntax Learning

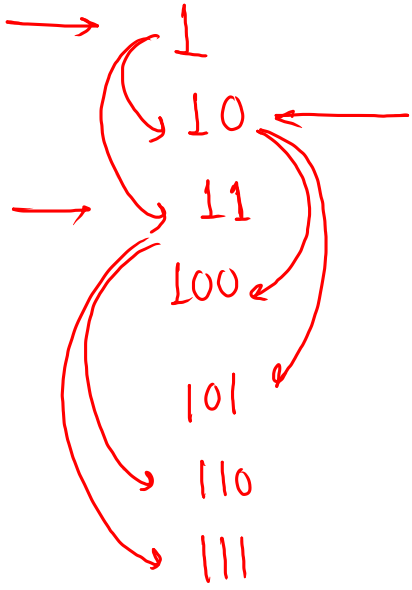
```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    Queue<Integer> que = new LinkedList<>();
    int t = scn.nextInt();
    for (int i = 0; i < t; i++) {
        int c = scn.nextInt();
        if ( c == 1 ) {
            System.out.println(que.size());
        } else if ( c == 2 ) {

            if ( que.size() == 0 ) {
                System.out.println( "-1" );
            } else {
                int ele = que.poll();
                System.out.println( ele );
            }

        } else if ( c == 3 ) {
            int x = scn.nextInt();
            que.add(x);
        } else if ( c == 4 ) {

            if ( que.size() == 0 ) {
                System.out.println( "-1" );
                continue;
            }
            System.out.println( que.peek() );
        }
    }
}
```

Print Binary

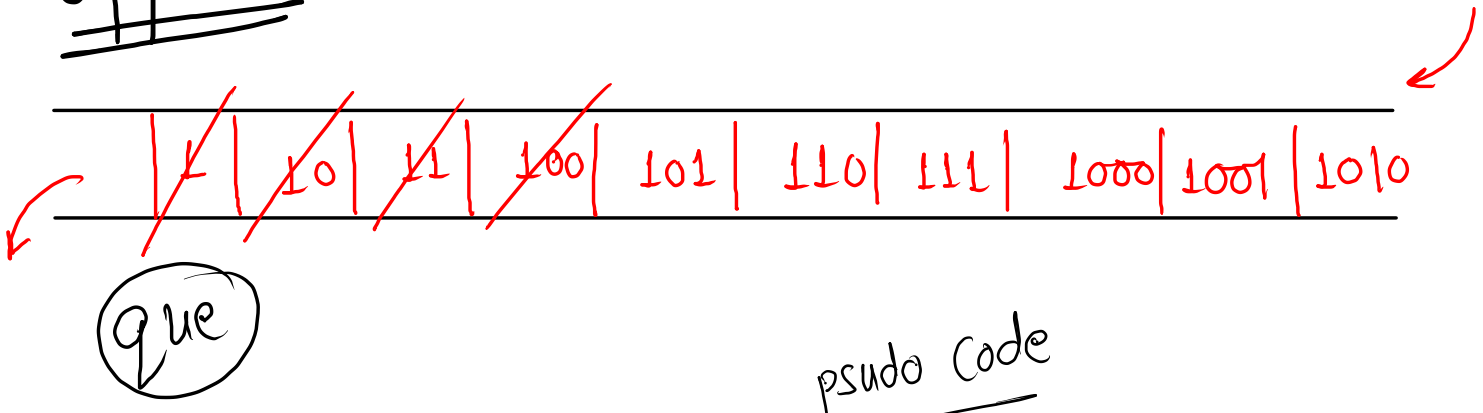


n = 10

0 →	0 ✓
1 →	1
2 →	10
3 →	11
4 →	100
5 →	101
6 →	110
7 →	111
8 →	1000
9 →	1001
10 →	1010



Approch



front = 1 10 11 100 101

n = 5

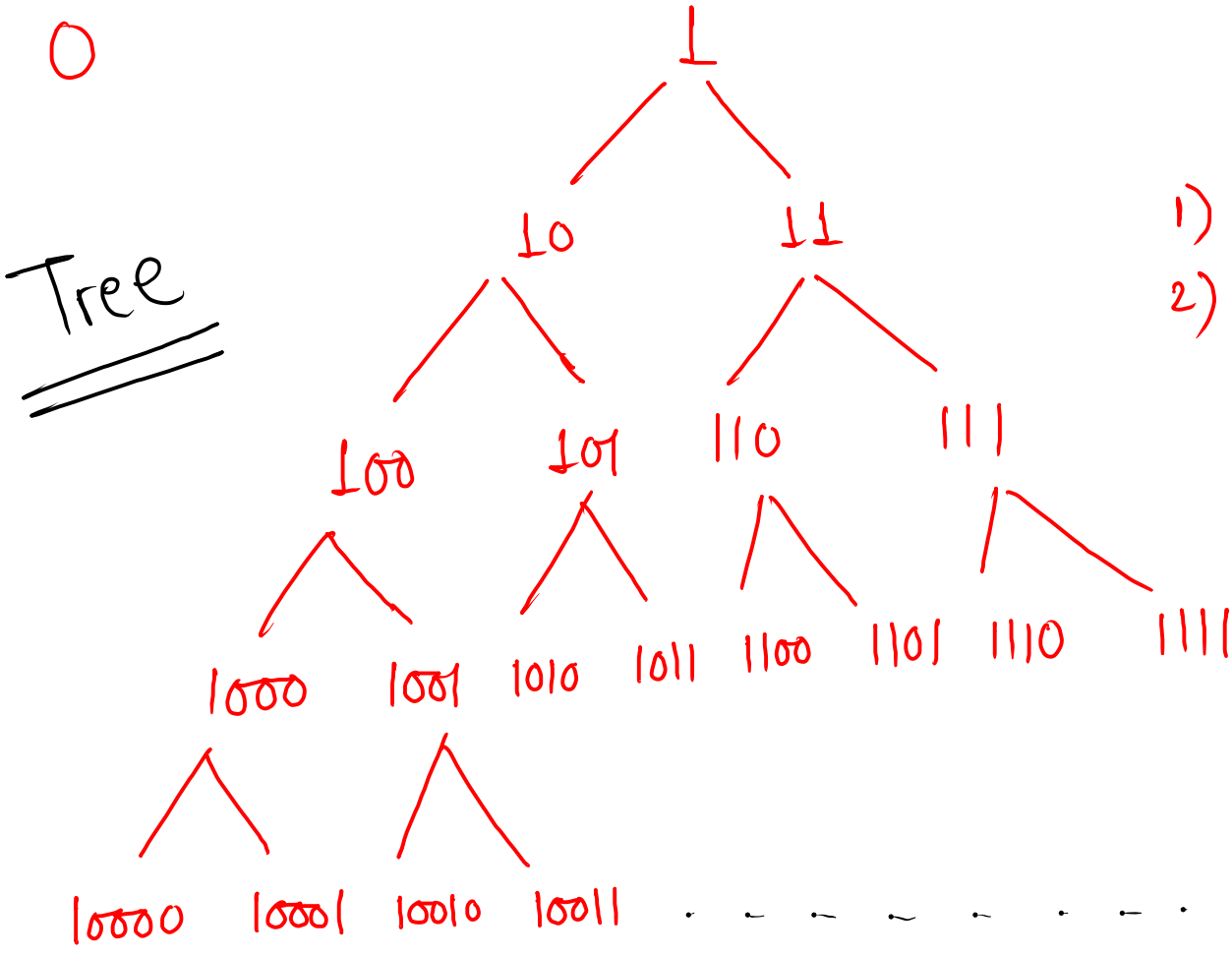
0
1
10
11
100

psudo Code

1) declare a queue
2) add "1" in que
3) keep iterating n time (input)
   3.1) get front element
   3.2) add front + "0" in que
   3.3) add front + "1" in que

0

Figure



Tree

1) num+0
2) num≠1

Code

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    printBinary(n);
}

public static void printBinary(int n) {
    Queue<String> que = new LinkedList<>();
    que.add("1");
    for (int i = 1; i <= n; i++) {
        String rem = que.poll();
        System.out.print(rem + " ");

        String str1 = rem + "0";
        que.add(str1);

        String str2 = rem + "1";
        que.add(str2);
    }
}
```

T.C = O(N)

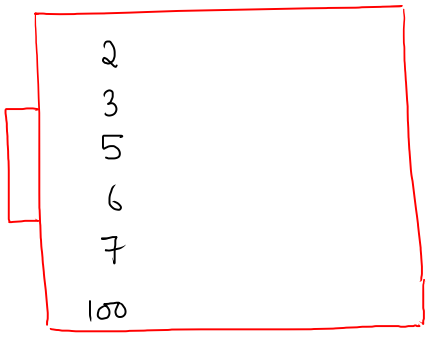
given n

S.C = O(N)

≅ 2*N

given 'n'

⇒ Priority Queue



remove → 2
→ 3
→ 5
→ 6
add → 9
remove → 7
remove → 9

Time taken by PQ to add an element will be $O(\log N)$ and to remove it, also $O(\log N)$

$(N * \log N) + (N \log N)$

means:- $O(2 * N \log N)$

Syntax:-

```
PriorityQueue<Integer> pq = new PriorityQueue<>();
```

Inbuilt functions

```
pq.add(x);    // to add element
pq.remove();  // to remove element
pq.poll();    // to remove element
pq.peek();    // to return top element
              without removig it
```

In Java)

default PQ is in ascending order

```
public static void main(String[] args) {
    PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());
    pq.add(4);
    pq.add(5);
    pq.add(5);
    pq.add(1);
    pq.add(3);
    pq.add(99);

    System.out.println( pq.peek() );
    System.out.println( pq.poll() );
    System.out.println( pq.poll() );
    System.out.println( pq.peek() );

    System.out.println( pq.size() );
}
```

used for descending order

arr 8 1 5 3 2 7 4

1 5 3 7 8 2 4

```
Arrays.sort(arr, (a, b) -> {
    if(a%2==0 && b%2!=0){
        return 1;
    }else{
        return -1;
    }
});
```

```
PriorityQueue<Integer> pq = new PriorityQueue<>((a, b) -> {
    return a - b;
});
```

lambda function is valid here