TOS -> no one is allowed to use this for cheating

- [x] Loop Detection

```
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
HashSet<Integer> st= new HashSet<>();
for(int i =0; i<n; i++) {
int m = sc.nextInt();
if(st.contains(m)) {
        System.out.println("YES");
        return;
}
st.add(m);
}
System.out.println("NO");

}
```

- [x] Sort the bitonic DLL

```
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
sc.nextLine();

for(int _i = 0; _i<n; _i++) {
String[] tokens = sc.nextLine().split(" ");
ArrayList<Integer> a = new ArrayList<>();
for(String token: tokens) {
        if(!token.equals("-1")) {
        a.add(Integer.parseInt(token));
        }
}
Collections.sort(a);
for(int i =0; i<a.size(); i++) {
        System.out.print(a.get(i)+" ");
}
System.out.println("");
}
```

- [x] Segregate even & odd nodes in a LL

```
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
ArrayList<Integer> even = new ArrayList<>();
ArrayList<Integer> odd = new ArrayList<>();
while(true) {
int a = sc.nextInt();
if(a == -1) break;
```

```
if(a%2 == 0) even.add(a);
else odd.add(a);
}
for(int i: odd) {
System.out.print(i + " ");
}
for(int i: even) {
System.out.print(i + " ");
}
}
```

- [x] Merge sort for DLL

```
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
ArrayList<Integer> al= new ArrayList<>();
for(int i = 0; i<n; i++) {
al.add(sc.nextInt());
}
int m = sc.nextInt();
for(int i = 0; i<m; i++) {
al.add(sc.nextInt());
}
Collections.sort(al);
for(int i = 0; i<n+m; i++) {
System.out.print(al.get(i));
if (i != n+m - 1) {
        System.out.print("->");
}
}
System.out.println("->NULL");
}
```

- [x] Minimum Stack

```
public static void main(String[] args) {
Scanner sc = new Scanner(System.in)   ;
int T = sc.nextInt();
for(int t = 0; t<T; t++) {
int n = sc.nextInt();
ArrayList<Integer> al = new ArrayList<>();
for(int i =0 ; i<n; i++) {
al.add(sc.nextInt());
}
System.out.println(Collections.min(al));
}
}
```

- [x] The Celebrity problem

```
 static int sum(int arr[]) {
        int s = 0;
        for(int i =0 ; i<arr.length; i++) {
                s += arr[i];
        }
        return s;
        }
        public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int arr[][] = new int[n][n];
        for(int i = 0; i<n; i++) {
        for(int j = 0; j<n; j++) {
        arr[i][j] = sc.nextInt();
        }
        }

        for(int i = 0; i<n; i++) {
        if(sum(arr[i]) == 0) {
                // test karo :D
                int vert = 0;

                for(int j = 0; j<n; j++) {
                vert += arr[j][i];
                if(vert == n-1){
                System.out.println(i);
                return;
                }
                }
        }
        }
        System.out.println("No Celebrity");
```

- [x] Iterative Tower of Hanoi

```
static void toh(int n, String from, String aux, String to) {
        if(n == 0) return;
        toh(n-1,from,to,aux); // swap last 2
        System.out.print(from+" "+to+"\n");
        toh(n-1,aux,from,to); // swap first 2
        }
        public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        toh(n,"a","b","c");
        }
```

- [x] Stock Span problem

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

ArrayList<Integer> al = new ArrayList<>();
for(int i =0; i<n; i++) {
al.add(sc.nextInt());
}
for(int i =0 ; i<n ; i++) {
int le =0;
for(int j = i; j>=0; j--) {
        if(al.get(j)<=al.get(i)) le++;
        else break;
}
System.out.print(le +" ");

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

- [x] Sort without extra Space

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

ArrayList<Integer> al = new ArrayList<>();
for (int i = 0; i < n; i++) {
al.add(sc.nextInt());
}
Collections.sort(al);
for (int i = 0; i < n; i++) {
System.out.print(al.get(i)+" ");
}

}
```

- [x] Max Sliding Window

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

ArrayList<Integer> al = new ArrayList<>();
for (int i = 0; i < n; i++) {
al.add(sc.nextInt());
}
int k = sc.nextInt();
```

```
        for(int i =0 ; i<=n-k; i++) {
        int mx = al.get(i);
        for(int j = 0;j<k;j++) {
                if(al.get(i+j)>mx) {
                mx = al.get(i+j);
                }
        }
        System.out.print(mx+" ");
        }
        }
```

- [x] Stack permutations

```
   public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = 3;
         ArrayList<Integer> ip = new ArrayList<>();
         ArrayList<Integer> op = new ArrayList<>();
         for (int i = 0; i < n; i++) ip.add(sc.nextInt());
        for (int i = 0; i < n; i++)  op.add(sc.nextInt());

        int j =0;
        Stack<Integer> st = new Stack<>();
           // push each element, pop if top matches
        for(int num: ip) {
           st.push(num);
           while(!st.isEmpty() && st.peek() == op.get(j)) {
              st.pop();
              j++;
           }
        }

        if(!st.empty())
                System.out.println("Not Possible");
        else
                System.out.println("YES");
```

- [x] Priority Queue using DLL

```
import java.io.*;
import java.util.*;

public class Solution {
        public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```

```
        ArrayList<int[]> arr = new ArrayList<>();

        while (true) {
        int n = sc.nextInt();

        if (n == 1) {
                int num = sc.nextInt();
                int p = sc.nextInt();
                arr.add(new int[]{num, p});
        }
        else if (n == 2) {
                if (!arr.isEmpty()) {
                arr.sort(Comparator.comparingInt(a -> a[1]));
                arr.remove(0);
                }
                else {
                System.out.println("Error List Empty");
                }
        }
        else if (n == 3) {
                arr.sort(Comparator.comparingInt(a -> a[1]));
                if(arr.size() == 0) {
                System.out.println("Empty");
                return;
                }
                System.out.print("Queue: ");
                for (int[] a : arr) {
                System.out.printf("%d %d ", a[0], a[1]);
                }
                System.out.println();
        }
        else if (n == 4) {
                return;
        }
        else {
                System.out.println("Wrong Choice");
        }
        }
        }
}
```

MCQs

1. Loop Detection in a Linked List

- **Algorithm:** Floyd's Cycle Detection (Tortoise and Hare)
- **Time Complexity: O(N)** (where N is the number of nodes in the list)
- **Space Complexity: O(1)** (uses two pointers only)

---

2. Sort a Bitonic Doubly Linked List

- **Algorithm:** Modified Merge Sort
- **Time Complexity: O(N log N)**
- **Space Complexity: O(log N)** (recursive stack space for merge sort)

---

3. Segregate Even and Odd Nodes in a Linked List

- **Algorithm:** Two-pointer approach
- **Time Complexity: O(N)**
- **Space Complexity: O(1)** (rearranges pointers without extra space)

---

4. Merge Sort for a Doubly Linked List

- **Algorithm:** Merge Sort (recursive or iterative)
- **Time Complexity: O(N log N)**
- **Space Complexity: O(log N)** (recursive stack for merge sort)

---

5. Minimum Stack (Supporting O(1) Min Operation)

- **Algorithm:** Use an auxiliary stack or maintain a min element in the stack
- **Time Complexity: O(1)** for push, pop, and getMin operations
- **Space Complexity: O(N)** (for storing min values in an auxiliary stack)

---

6. The Celebrity Problem (Find the Celebrity in a Party)

- **Algorithm:** Two-pointer elimination approach
- **Time Complexity: O(N)**
- **Space Complexity: O(1)**

---

7. Iterative Tower of Hanoi

- **Algorithm:** Use an explicit stack to simulate recursion
- **Time Complexity: $O(2^n - 1) \approx O(2^n)$**
- **Space Complexity: O(N)** (stack space to store moves)

---

8. Stock Span Problem

- **Algorithm:** Stack-based approach
- **Time Complexity: O(N)** (each element is pushed and popped once)
- **Space Complexity: O(N)** (stack storage)

---

9. Priority Queue using Doubly Linked List

- **Algorithm:** Insert in sorted order, extract max/min in O(1)
- **Time Complexity:**
    - **Insertion: O(N)** (traversing to find the correct position)
    - **Deletion (max/min): O(1)** (head/tail removal)
- **Space Complexity: O(N)**

---

10. Sort Without Extra Space (DLL or LL)

- **Algorithm:** Insertion Sort (if singly LL) or Merge Sort (if DLL)
- **Time Complexity: $O(N^2)$ (Insertion Sort) or O(N log N) (Merge Sort)**
- **Space Complexity: O(1) (Insertion Sort) or O(log N) (Merge Sort recursive calls)**

---

11. Maximum Sliding Window (of size k)

- **Algorithm:** Monotonic Deque
- **Time Complexity: O(N)** (each element is pushed/popped once)
- **Space Complexity: O(K)** (stores at most K elements in deque)

---

12. Stack Permutations (Check if One Stack Permutation is Possible from Another)

- **Algorithm:** Use an auxiliary stack and simulate the process
- **Time Complexity: O(N)**
- **Space Complexity: O(N)** (auxiliary stack)