3 June

Date : __ / __ / ____

1. Maximum Group Fund for Flood Relief

→ You have volunteers.
→ Each has collected some funds.
→ Some volunteers are directly or transitively connected.
→ Find the maximum total funds from any one connected group.

→ Code:

```
public class FloodReliefFund {
    static int[] parent;
    static int[] fund;
Public static int find(int x) {
    if( parent[x] != x)
      parent[x] = find(parent[x]);
    return parent[x];
    }
Public static void union( int u, int v) {
    int pu = find(u);
    int pv = find(v);
    if( pu != pv) {
      Parent[pv] = pu; }
    }.   fund = new int[n+1]; Parent = new int[n+1]
for(int i = 1; i <= n; i++) {
  fund[i] = sc.nextInt();
    Parent[i] = i;
    }.
```

Date : __/__/____

```
Map<Integer, Integer> groupFund = new HashMap<>();
    for(int i=1; i<=n; i++) {
        int groupLeader = find(i);
groupFund.put(groupLeader, groupFund.getOrDefault(groupLeader,0)
                + fund[i]);
    }
    int maxFund=0;
    for(int total: groupFund.values()){
    maxFund = Math.max(maxFund, total);
                }
    }
}
```

Dry Run :

val : 1 2 3 4 5

funds : 23 43 123 54 2

Connections : 1 - 3 → union(1,3)

2 - 3 → union(2,3).

1 - 2 → redundant since 1 & 2 already connected via 3.

Group funds :

1 2 3 : 23 + 43 + 123 = 189.

4 : 54

5 : 2.

Max = 189.

Time Complexity = O(N+P)

2. Server Load Balancing - Minimize Peak Load

→ You drive towards a target distance.
→ Start with startFuel.
→ Some fuel stations along the way, each at a position with a fuel amount.
→ You want to reach the destination by making minimum refueling stops.
You consume 1 fuel per 1 distance unit.

Logic:
→ Stop at a station if can't reach further.
→ Always pick the maximum fuel available among station you passed.
→ Use a max-heap to track potential fuel amounts.

Code:

```
public class minRefuel {
PriorityQueue<Integer> maxHeap = new PriorityQueue<>(
        Collections.reverseOrder());
int stops = 0, idx = 0, n = stations.length;
int fuel = startFuel;
while (fuel < target) {
while (idx < n && stations[idx][0] <= fuel) {
maxHeap.add(stations[idx][1];
        idx++;
    }
if(maxHeap.isEmpty()) return -1;
fuel += maxHeap.poll(); stops++;
    }
return stops;
}
```

Date : __ / __ / ____

Dry Run :

→ Start with 10 fuel.

→ Can reach station 10 : Add 60 to max-heap.
fuel = 0., Refuel = pick 60 from heap.
Total fuel = 60.
Stops = 1

→ Next station is at 20.
fuel left ; 60 - 10 = 50.
Add 30 to heap.

→ Next station at 30.
fuel left : 50 - 10 = 40
Add 30 to heap.

→ Next station at 60.
fuel left : 40 - 30 = 10
Add 40 to heap.

→ Destination at 100
fuel left : 10 - 40 = -30
Need more, Pick 40 from heap.
New fuel : 10 + 40 = 50
stops : 2.

→ Now 30 fuel is enough to cover remaining 40.
Total Stops : 2.

Time Complexity : $O(N \log N)$

Space complexity : $O(N)$