Greedy Algorithms

we will start by 9:10.

WHAT ⟹ local optimal choice at each step helps solving global optimum (minima/ maxima)

APPLICATIONS

1. Fractional Knapsack
2. Activity Selection / Overlapping Intervals / Meeting Rooms
3. Minimum Spanning Tree (Prims (PQ), Kruskal (DSU))
4. Huffman Encoding & Decoding / optimal Merge Pattern
5. Single Source Shortest Path Algo {DIJKSTRA}
6. Job Sequencing
7. Problems based on SORTING. — nlogn

Lecture ①

1. Job Sequencing
2. Meeting Rooms - I
3. Disjoint Intervals - I
4. Disjoint Intervals - II
5. Maximum Chain Length
6. Minimum Balloon Burst

Saturday Morning

Lecture ②

1. Meeting Rooms - II
2. Minimum Platforms
3. Car Pooling
4. Merge overlapping Intervals
5. Insert Interval
6. Intervals Intersection

Saturday Evening

lecture (3)

1. Circular Tour (salesforce)
2. Car Fleet (sprinklr)
3. Two City Scheduling
4. Candy/Temple offerings
5. Chocolate Distribution
6. Queue Reconstruct Height

Sunday Morning

lecture (4)

1. Biased Standings
2. Defense Kingdom
3. Georgivia
4. Amplifiers
5. load Balancing

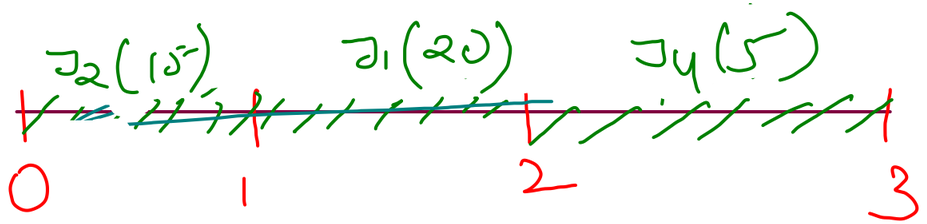
Huffman Coding & Decoding

Sunday Evening

Job Sequencing

1 job → 1 unit of time
Maximum profit

n=5	i⇒ 0	1	2	3	4
Jobs	J_1	J_2	J_3	J_4	J_5
profits	20	15	10	5	1
deadlines	2	2	1	3	3



(1) Order of picking jobs
⇒ decreasing order of profit

(2) Picked job should be placed as last as possible

JA : 1 2 3
Profit = 20 + 40

```java
public static class MyComparator implements Comparator<Job>{
    public int compare(Job obj1, Job obj2){
        if(obj1.profit != obj2.profit){
            return obj2.profit - obj1.profit;
        }
        return obj2.deadline - obj1.deadline;
    }
}

//Function to find the maximum profit and the number of jobs done.
int[] JobScheduling(Job arr[], int n)
{
    Arrays.sort(arr, new MyComparator()); ⇒ nlogn
    int maxDeadline = 0;
    for(int i=0; i<n; i++){
        maxDeadline = Math.max(arr[i].deadline, maxDeadline);
    }

    boolean[] slots = new boolean[maxDeadline];
    int maxProfit = 0;
    int jobsAllocated = 0;

    for(int i=0; i<n; i++) {
        for(int j=arr[i].deadline-1; j>=0; j--){
            if(slots[j] == false){
                slots[j] = true;
                jobsAllocated++;
                maxProfit += arr[i].profit;
                break;
            }
        }
    }

    return new int[]{jobsAllocated, maxProfit};
}
```

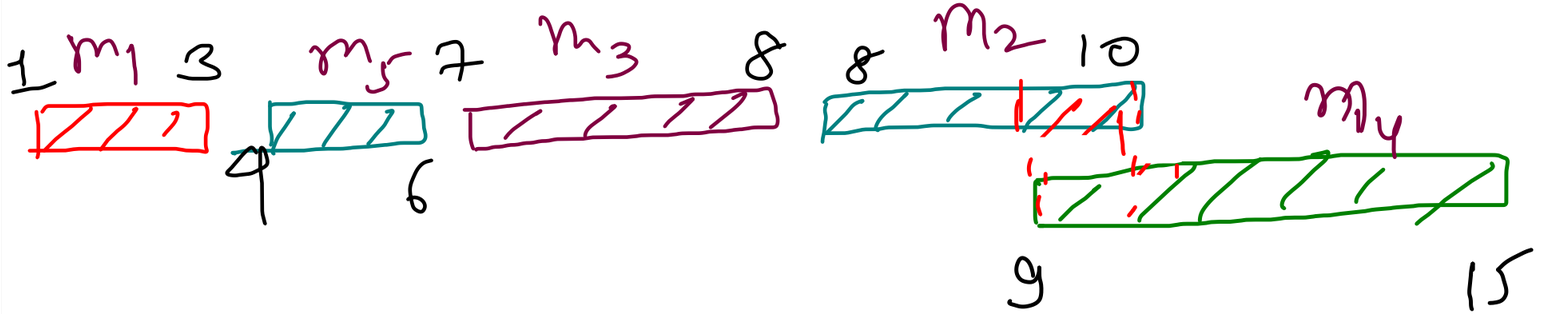
d * n
max deadline

O(dn + nlogn)

Meeting Rooms - I

One room ⇒ one meeting at a time

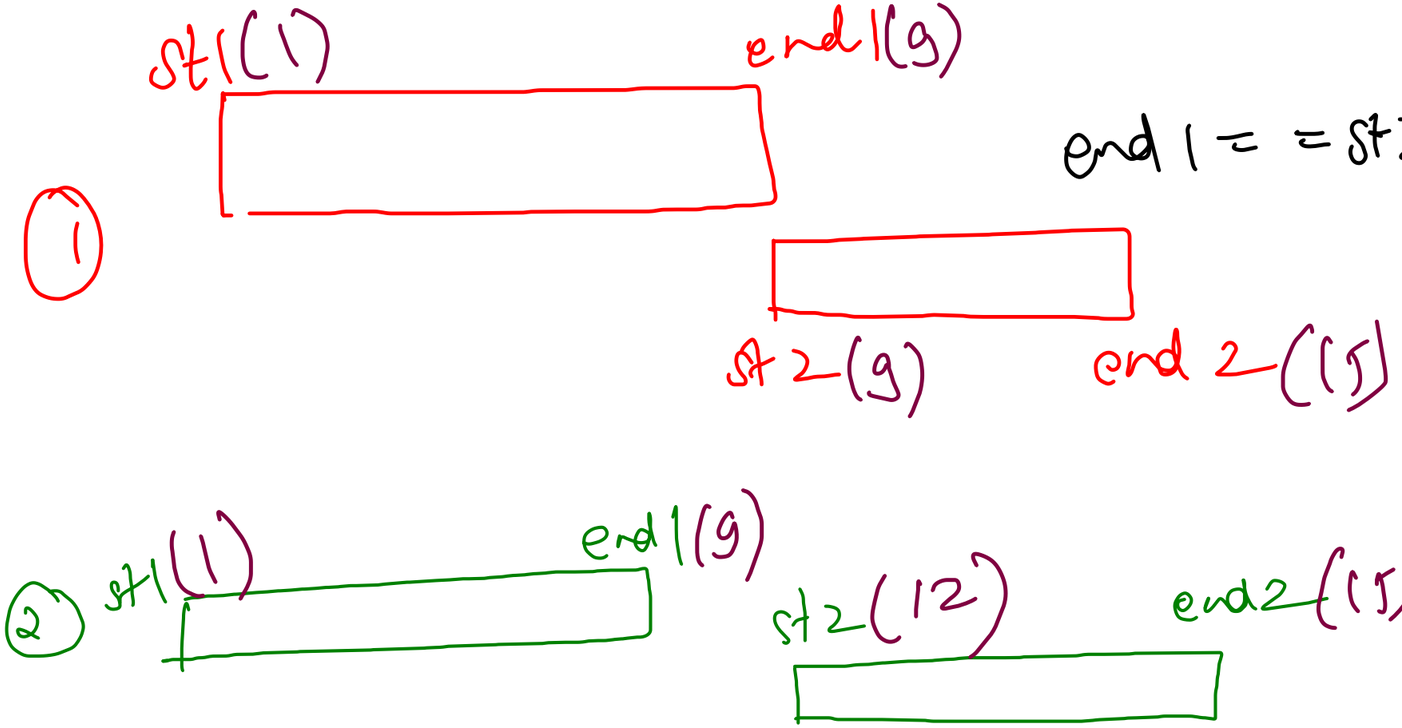
		0	1
		st	et
①	m_1	1	3
②	m_2	8	10
③	m_3	7	8
④	m_4	9	15
⑤	m_5	4	6



① Sort intervals based on ending index

② Check overlapping with previous slot.

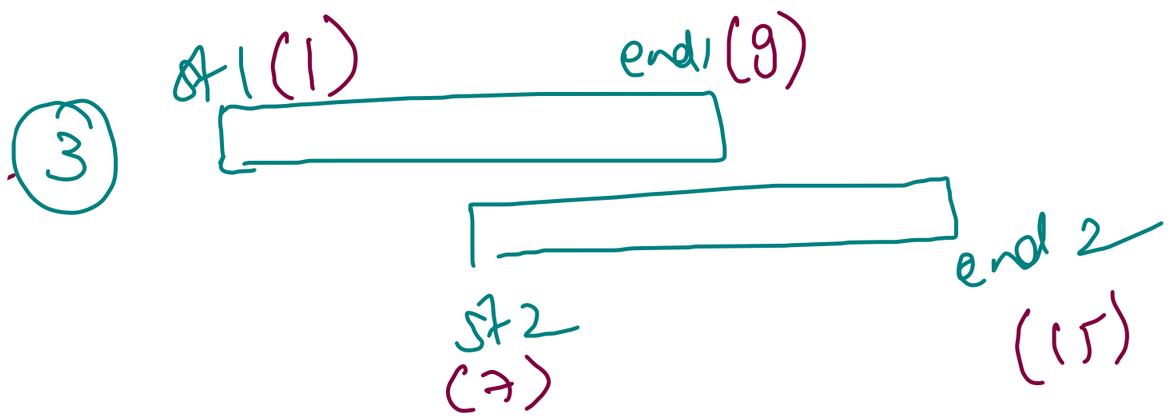
false



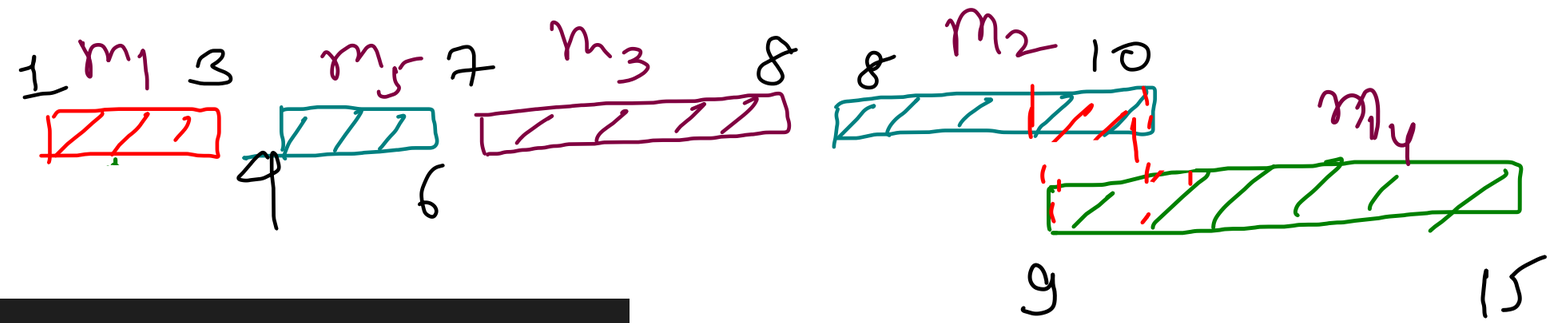
end1 == st2 (T)

end1 ≤ end2

end1 < st2 (T)



end1 > st2 (F)

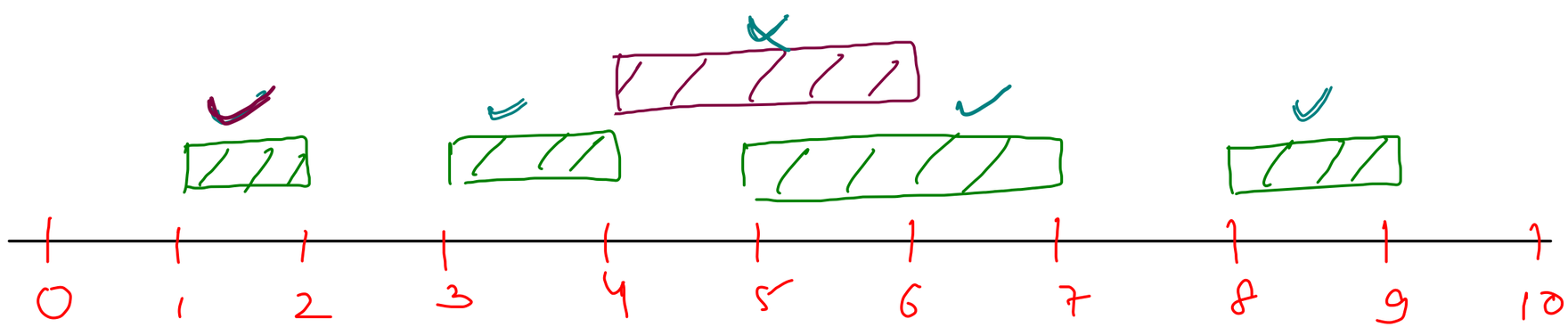



```java
public class Solution {
    public static class MyComparator implements Comparator<Interval>{
        public int compare(Interval obj1, Interval obj2){
            if(obj1.end != obj2.end)
                return obj1.end - obj2.end;
            return obj1.start - obj2.start;
        }
    }

    public boolean canAttendMeetings(List<Interval> intervals) {
        Collections.sort(intervals, new MyComparator());

        int limit = Integer.MIN_VALUE; // last interval's ending time

        for(int i=0; i<intervals.size(); i++){
            if(limit > intervals.get(i).start){
                return false;
            }
            limit = intervals.get(i).end;
        }

        return true;
    }
}
```

$\Rightarrow O(N \log N)$

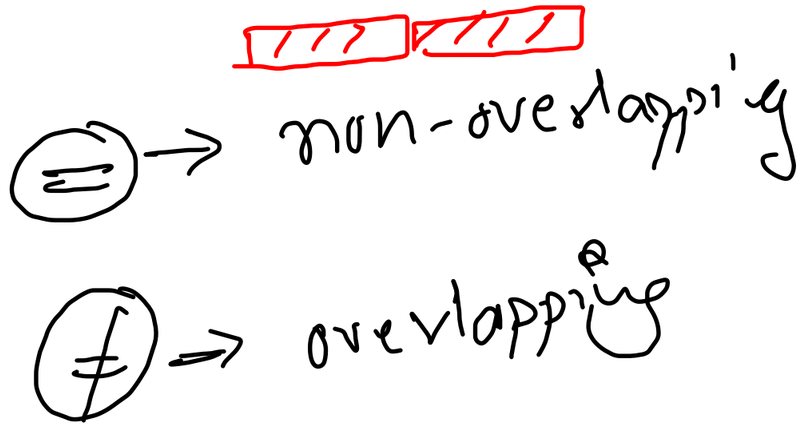
$\Rightarrow O(N)$

S[] =	1	0	3	8	5	~~8~~ 4
F[] =	2	~~6~~ 8	4	9	7	~~7~~ 6
	1	2	3	4	5	6

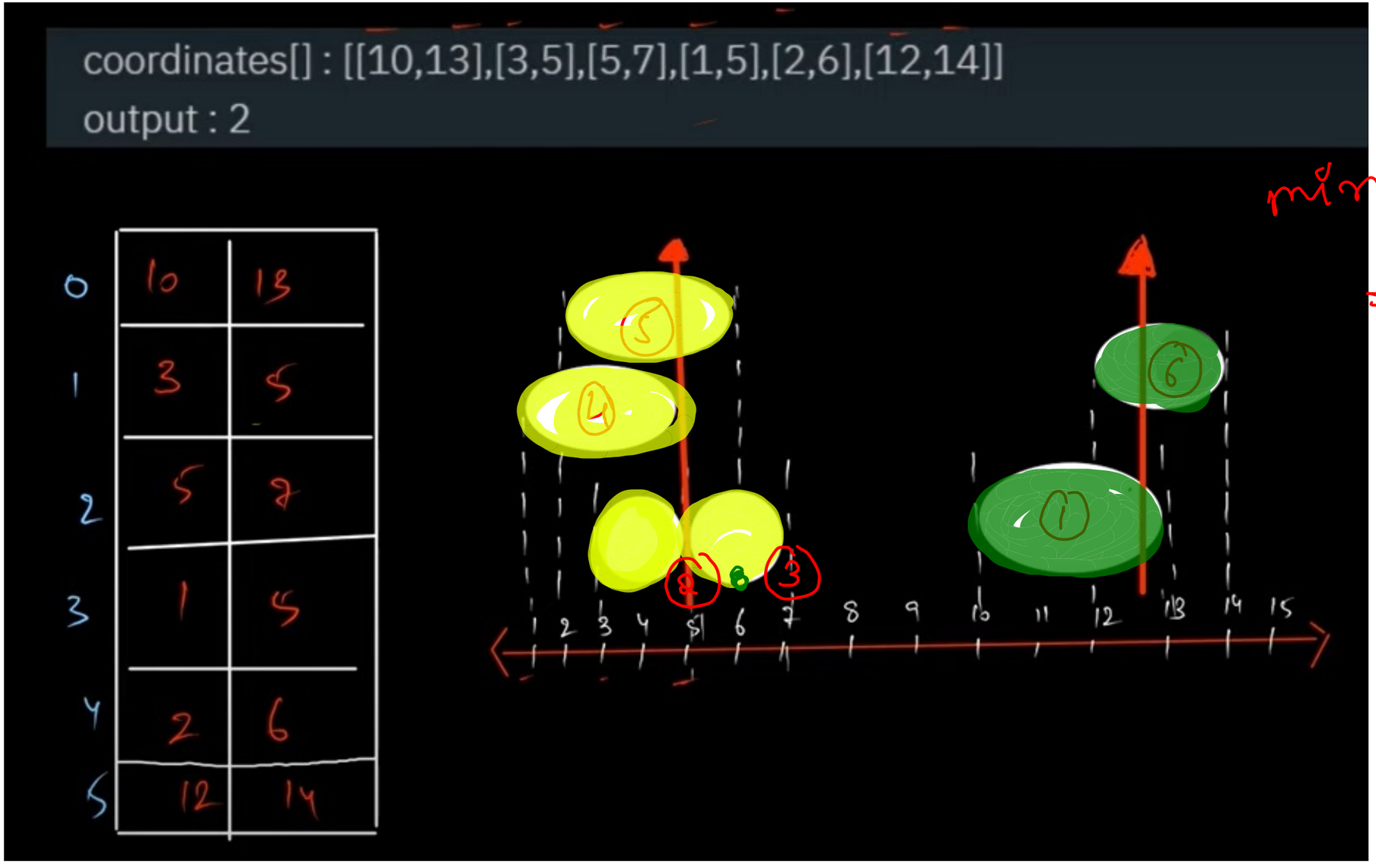


```
int limit = Integer.MIN_VALUE; // last interval's ending time
int count = 0;

for(int i=0; i<n; i++){
    if(limit < intervals[i].start){
        count++;
        limit = intervals[i].end;
    }
}

return count;
```

end2 == 8+1



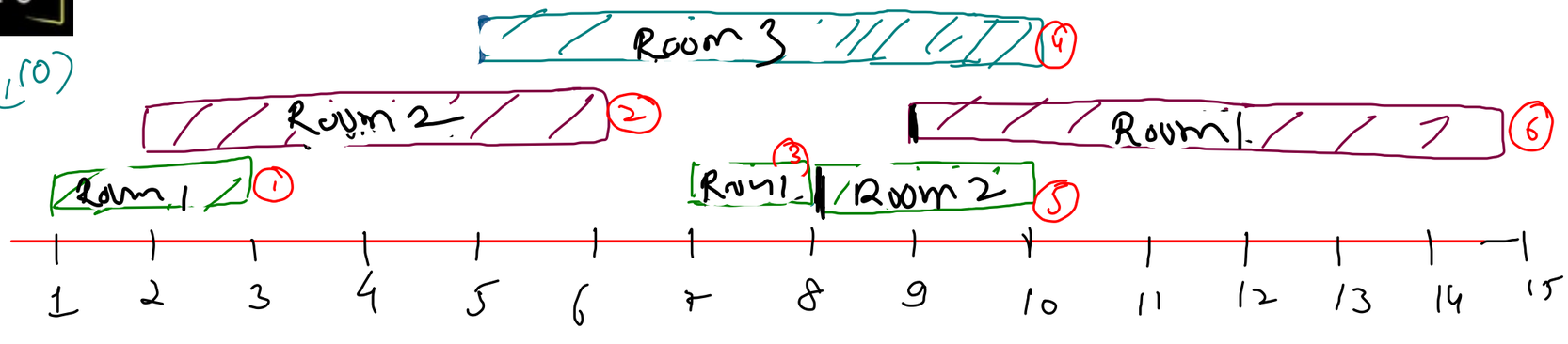
Minimum Balloon Bursts



minimum arrow = maximum meetings that can be accomodated in one room

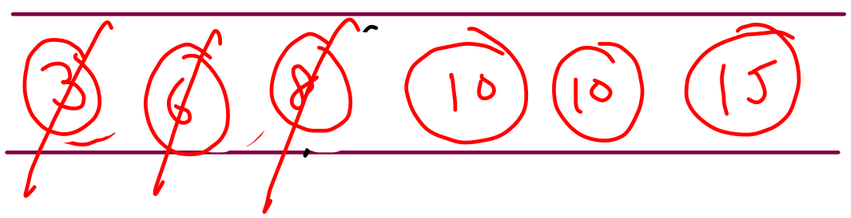
minimum rooms

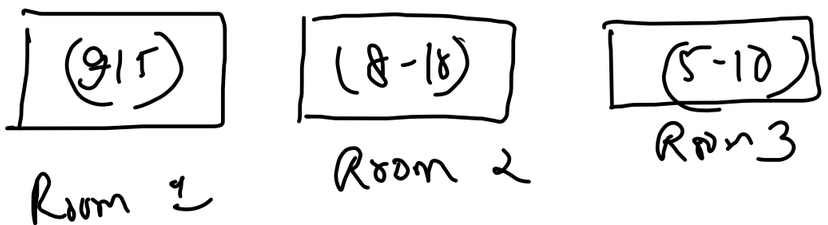
```
5
1 3
8 10
7 8
9 15
2 6
```



(5,10)

Count of Rooms = ~~0~~ ~~1~~ ~~2~~ 3

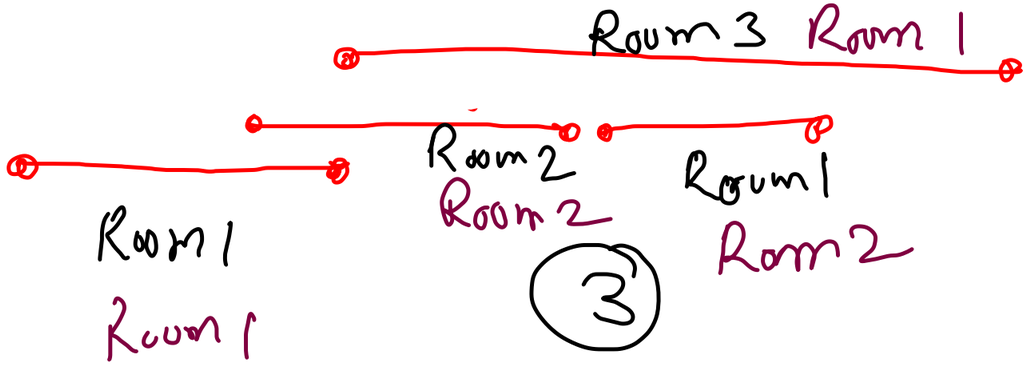




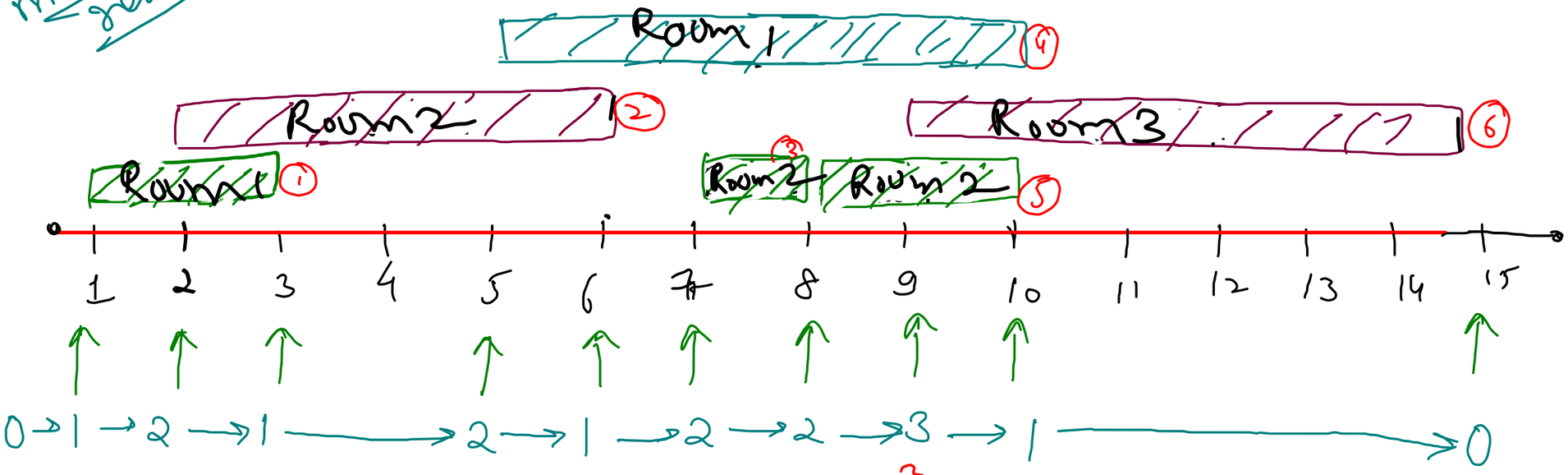

```java
public static int meetingRooms(int intervals[][]) {
    int maxRooms = 0;
    Arrays.sort(intervals, (a, b) -> a[1] - b[1]);

    Queue<Integer> q = new ArrayDeque<>();

    for(int i=0; i<intervals.length; i++){
        if(q.size() == 0 || q.peek() > intervals[i][0]){
            maxRooms++;
        } else {
            q.remove();
        }

        q.add(intervals[i][1]);
    }

    return maxRooms;
}
```

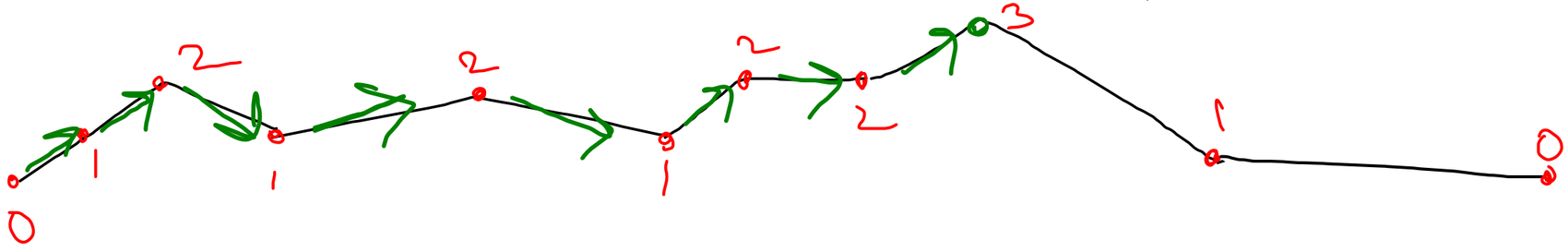
Time → O(NlogN + N)

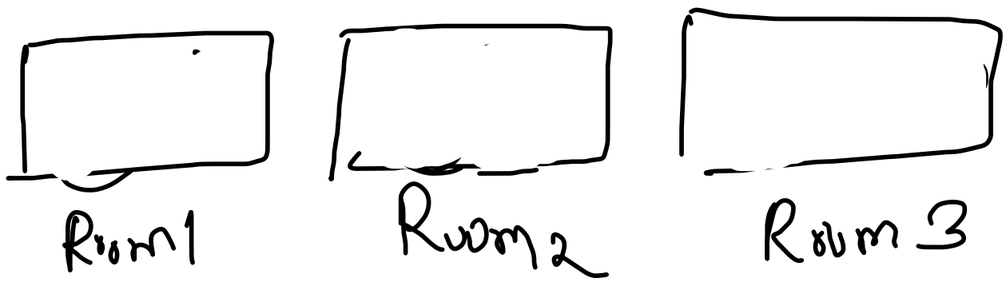
Space → O(N)


Room 3
Room 1
Room 2
Room 2
Room 1
Room 2
Room 1
Room 1
3

minimum rooms







1 2 5 8 9

3 6 8 10 15

```java
public int minMeetingRooms(List<Interval> intervals) {
    ArrayList<Integer> start = new ArrayList<>();
    ArrayList<Integer> end = new ArrayList<>();
    for(int i=0; i<intervals.size(); i++){
        start.add(intervals.get(i).start);
        end.add(intervals.get(i).end);
    }

    Collections.sort(start);
    Collections.sort(end);

    int currentRooms = 0, maxRooms = 0;
    int startIdx = 0, endIdx = 0;

    while(startIdx < intervals.size()){
        if(start.get(startIdx) < end.get(endIdx)){
            startIdx++;
            currentRooms++;
        } else if(end.get(endIdx) < start.get(startIdx)){
            endIdx++;
            currentRooms--;
        } else{
            startIdx++;
            endIdx++;
        }
        maxRooms = Math.max(maxRooms, currentRooms);
    }
    return maxRooms;
}
```

O(N)

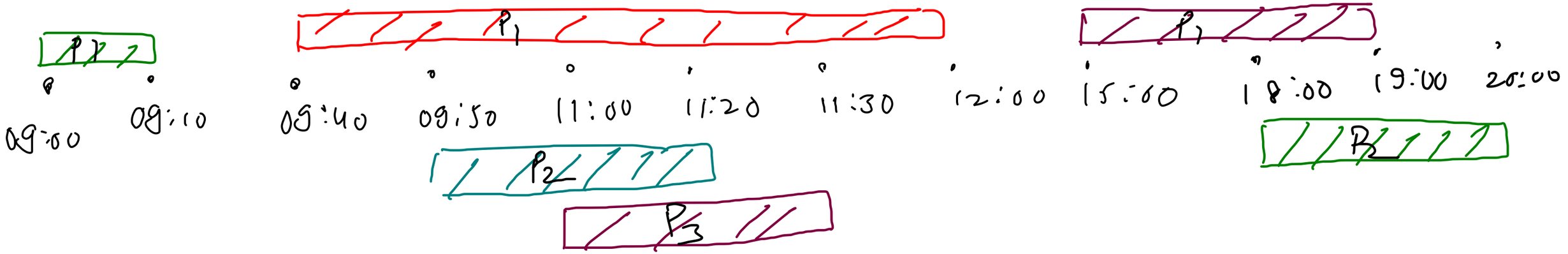
O(NlogN)

O(2N)

Minimum Platforms

{ same as meeting Rooms - II }

```
{0900, 0940, 0950, 1100, 1500, 1800}
{0910, 1200, 1120, 1130, 1900, 2000}
```



Car Pooling

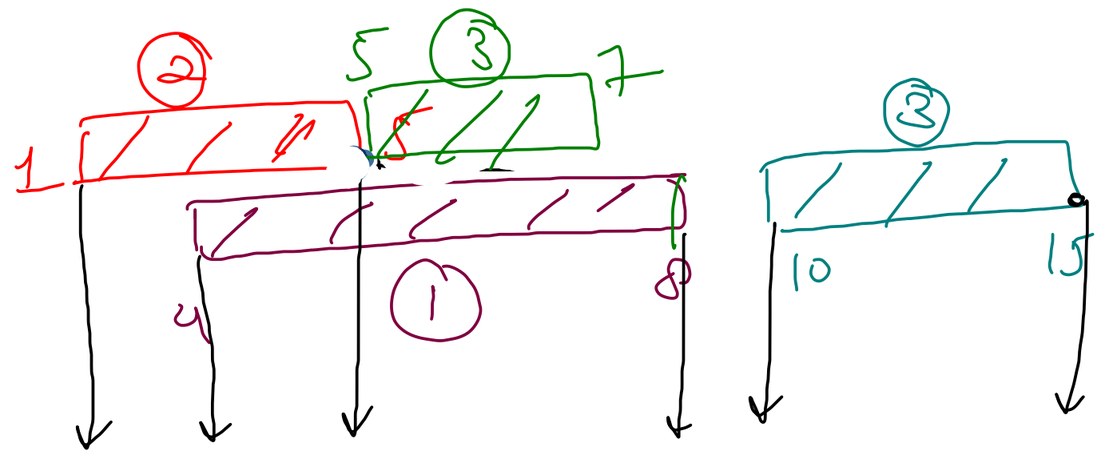
{2, 1, 5}

{1, 4, 8}

{3, 10, 15}

3

direct^n ⇒ left to right , max^m capacity = 3



0 → 2 → 3 → 1 → 0 → 3 → 0

HINT

TreeMap { Red Black Tree }

↓

Ordered (keys)

```
TreeMap<Integer, Integer> changes = new TreeMap<>();
for(int i=0; i<trips.length; i++){
    int passengers = trips[i][0];
    int start = trips[i][1];
    int end = trips[i][2];

    changes.put(start, changes.getOrDefault(start, 0) + passengers);
    changes.put(end, changes.getOrDefault(end, 0) - passengers);
}

int currPassengers = 0;
for(Integer key: changes.keySet()){
    Integer val = changes.get(key);
    currPassengers += val;

    if(currPassengers > capacity) return false;
}
return true;
```

O(N log N)

O(N)

1 → +2

4 → +1

5 → −2

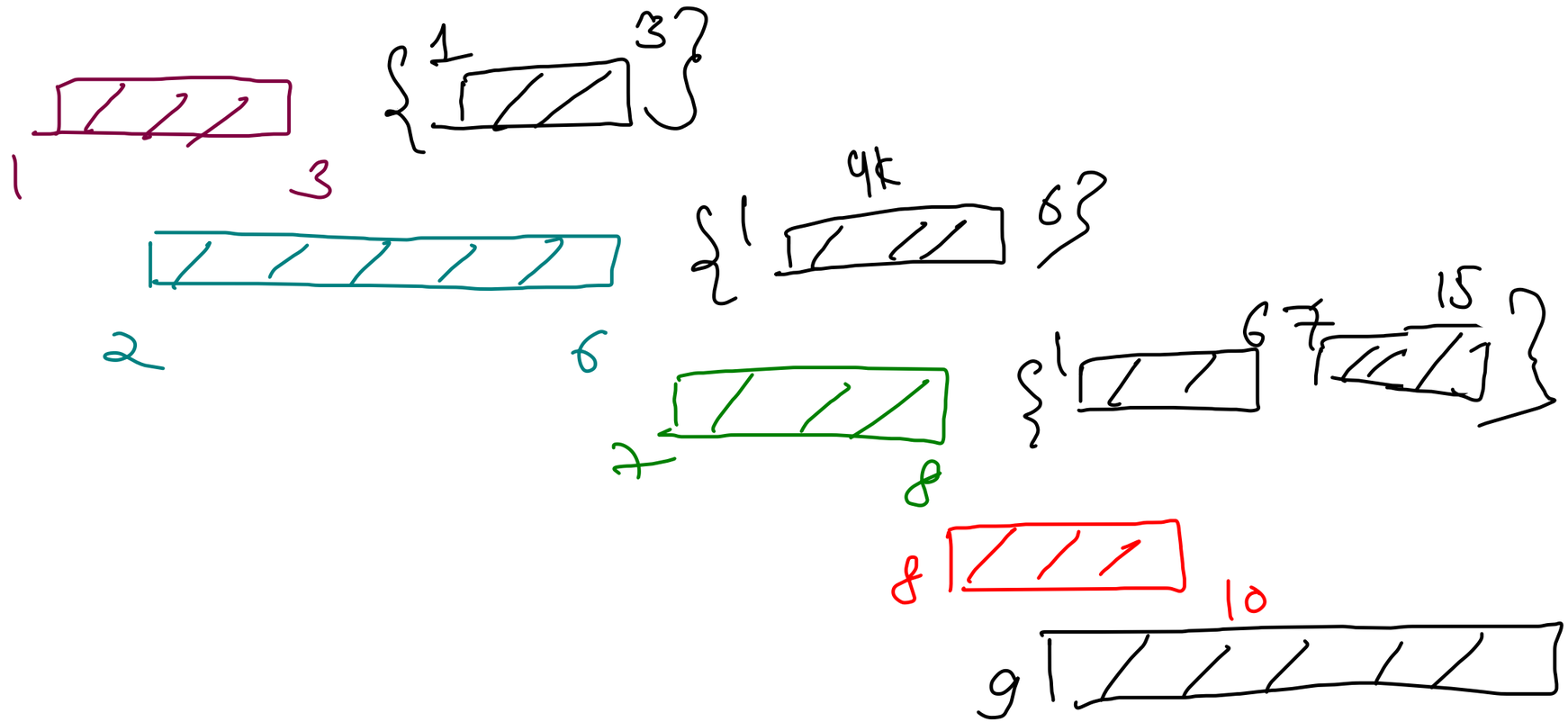
8 → −1

10 → +3

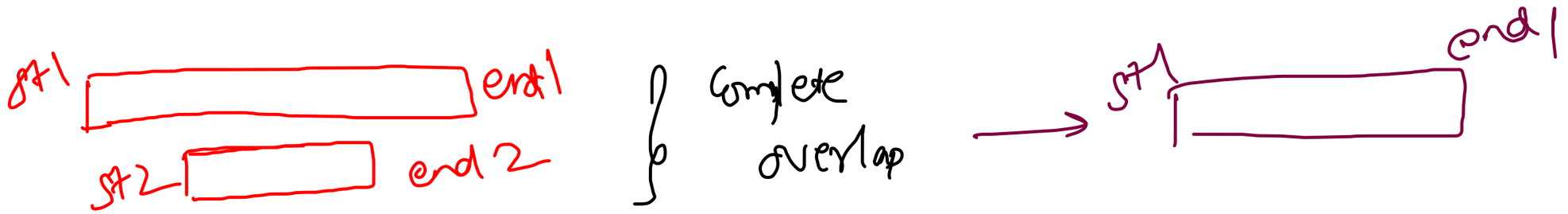
15 → −3

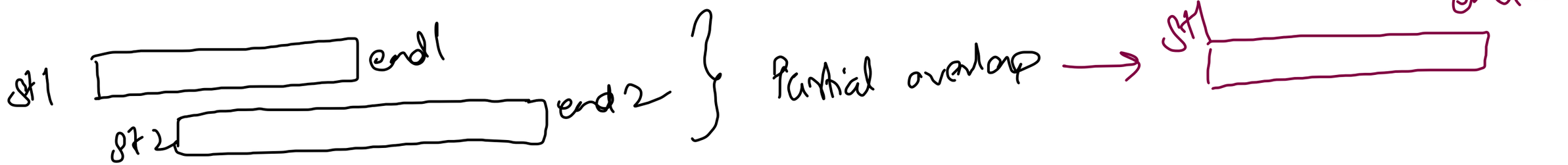
Merge Overlapping Intervals

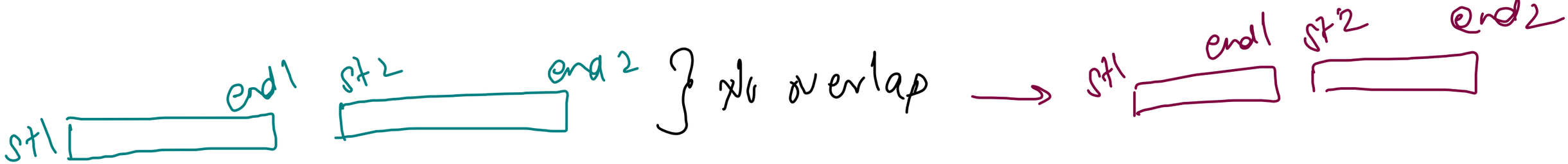
	0	1
0	1	3
1	8	10
2	7	8
3	9	15
4	2	6



Sort on basis of starting index ⇒ st1 <= st2







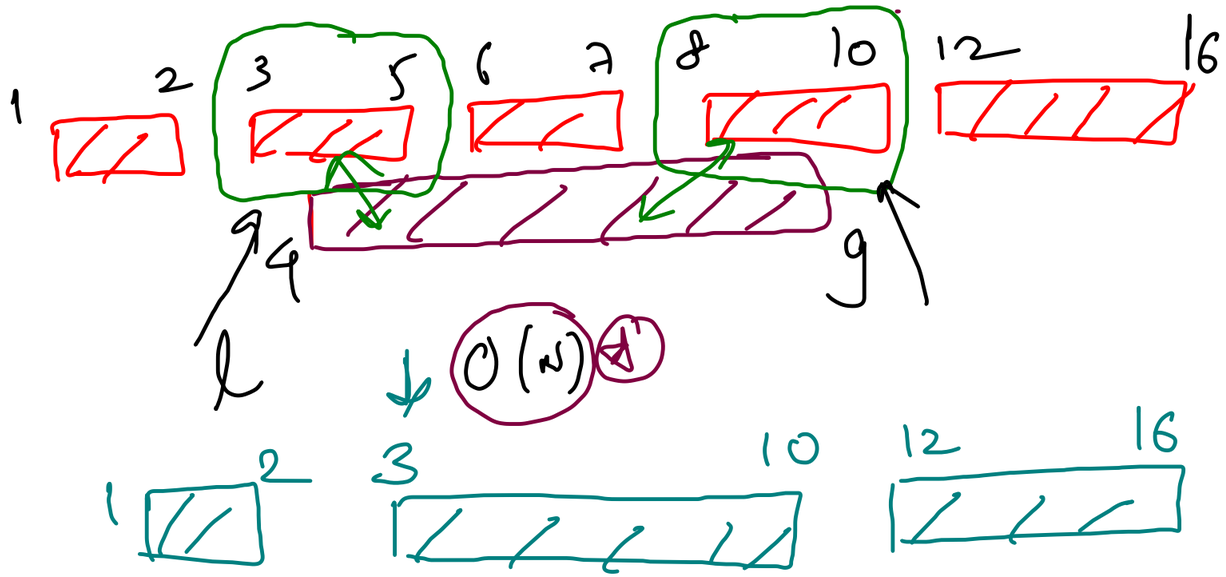
```java
public int[][] merge(int[][] intervals) {
    Arrays.sort(intervals, (a, b) -> a[0] - b[0]);
    ArrayList<int[]> merged = new ArrayList<>();
    merged.add(intervals[0]);

    for(int i=1; i<intervals.length; i++){
        int[] lastInt = merged.get(merged.size() - 1);
        int[] currInt = intervals[i];

        if(lastInt[1] >= currInt[0]){
            // merge
            lastInt[1] = Math.max(lastInt[1], currInt[1]);
        } else {
            merged.add(currInt);
        }
    }

    int[][] res = new int[merged.size()][2];
    for(int i=0; i<merged.size(); i++){
        res[i] = merged.get(i);
    }
    return res;
}
```

TC :→ O(NlogN)

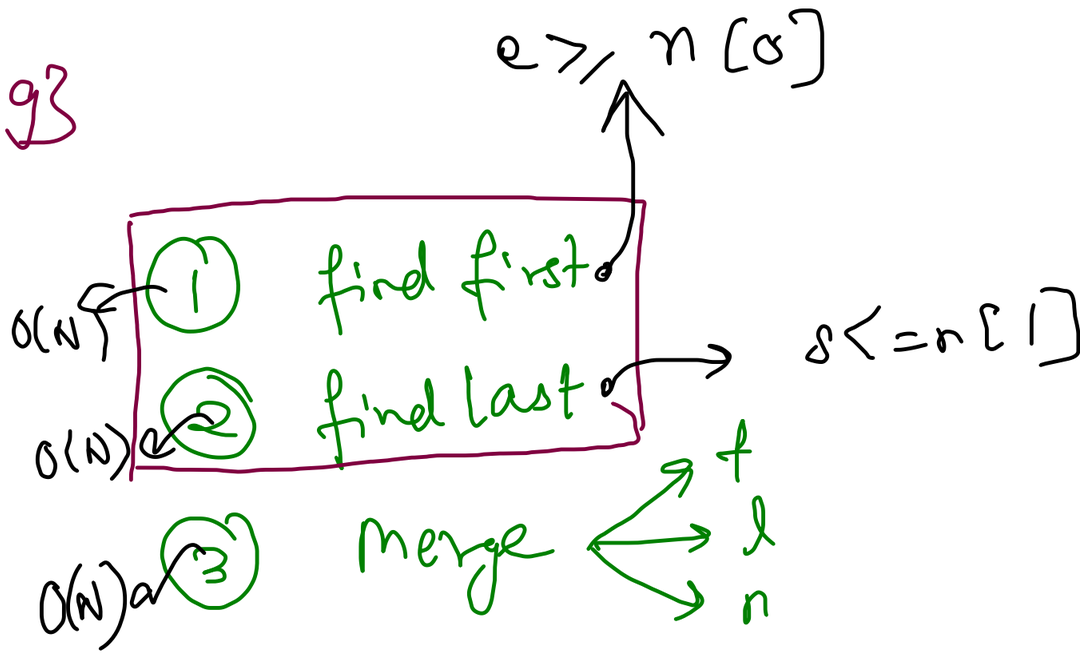
SC
=
O(N) + O(N) + O(1)
output input extra space

Insert Interval

```
[[1,2],[3,5],[6,7],[8,10],[12,16]].
```



24, 93



{ min(f[0], n[0]),
max(l[1], n[1]) }

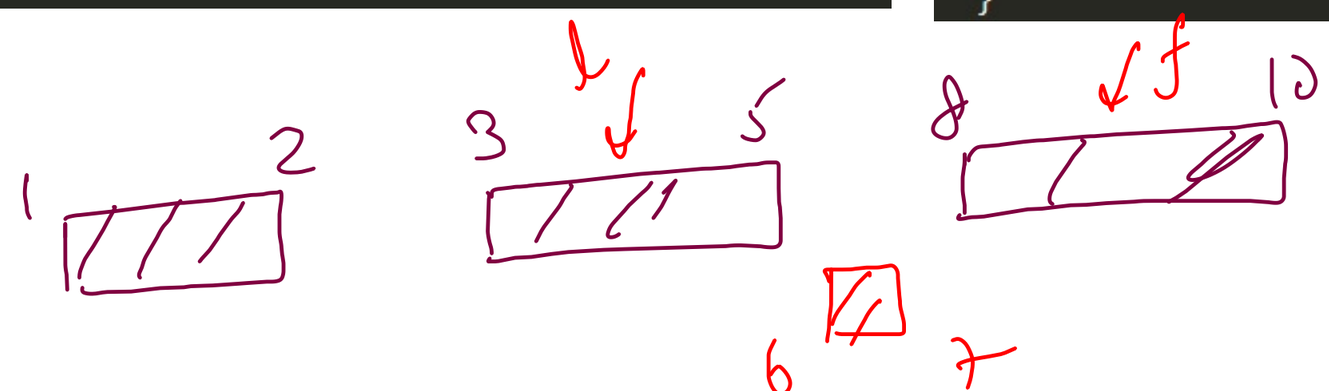
```
int[] merged = new int[2];
merged[0] = Math.min(intervals[firstIdx][0], newInterval[0]);
merged[1] = Math.max(intervals[lastIdx][1], newInterval[1]);
res.add(merged);
```

```java
int findFirst(int[][] intervals, int[] newInterval){
    for(int idx=0; idx<intervals.length; idx++){
        if(intervals[idx][1] >= newInterval[0]){
            return idx;
        }
    }
    return intervals.length;
}

int findLast(int[][] intervals, int[] newInterval){
    for(int idx=intervals.length-1; idx>=0; idx--){
        if(intervals[idx][0] <= newInterval[1]){
            return idx;
        }
    }
    return -1;
}
```

```java
int firstIdx = findFirst(intervals, newInterval);
int lastIdx = findLast(intervals, newInterval);

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

```java
// non merging -> firstIdx > lastIdx
if(firstIdx > lastIdx){
    for(int i=0; i<=lastIdx; i++){
        res.add(intervals[i]);
    }
    res.add(newInterval);
    for(int i=firstIdx; i<intervals.length; i++){
        res.add(intervals[i]);
    }
```

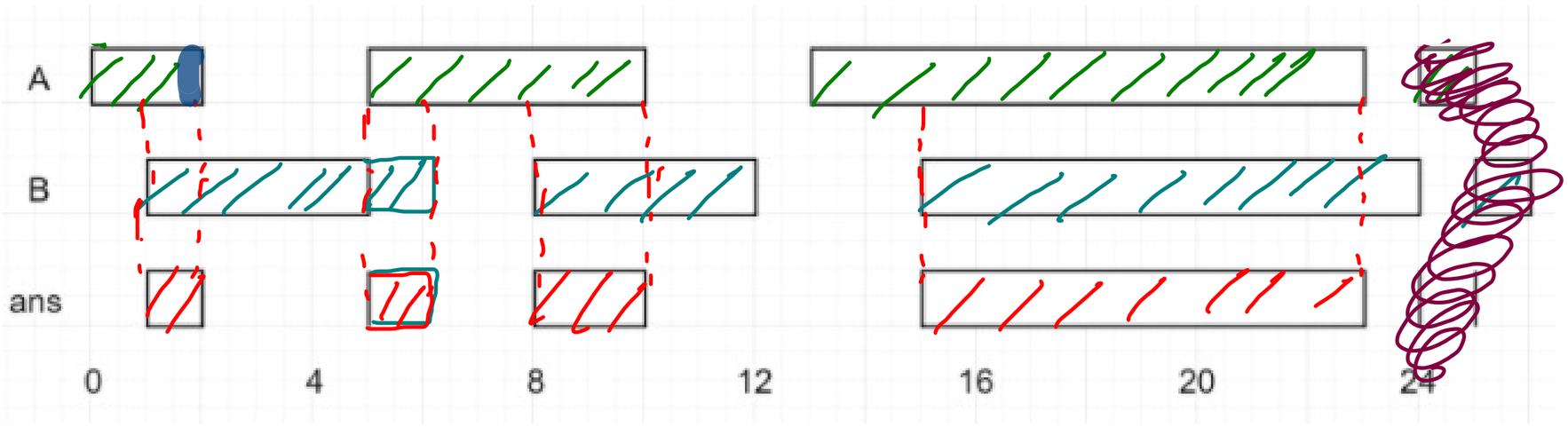
```java
} else {
    for(int i=0; i<firstIdx; i++){
        res.add(intervals[i]);
    }

    int[] merged = new int[2];
    merged[0] = Math.min(intervals[firstIdx][0], newInterval[0]);
    merged[1] = Math.max(intervals[lastIdx][1], newInterval[1]);
    res.add(merged);

    for(int i=lastIdx+1; i<intervals.length; i++){
        res.add(intervals[i]);
    }
}
```



Intervals Intersection

non-overlapping

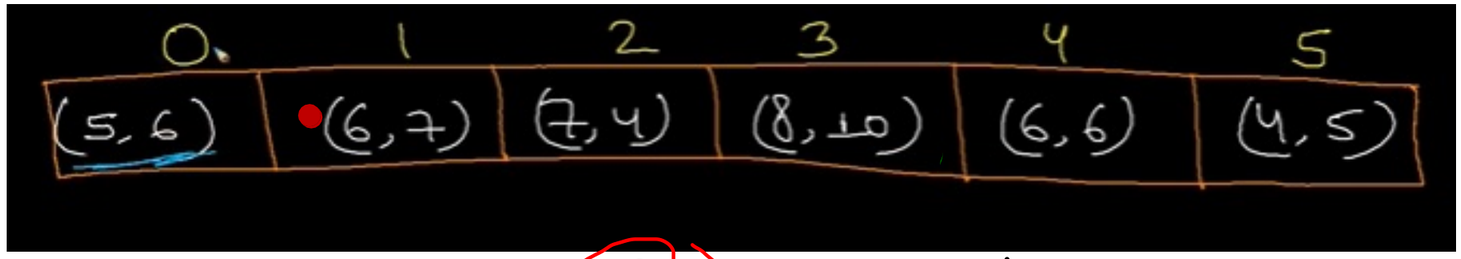
. Each list of intervals is pairwise **disjoint** and in **sorted order**.



```java
public int[][] intervalIntersection(int[][] firstList, int[][] secondList) {
    int firstIdx = 0, secondIdx = 0;
    ArrayList<int[]> intersect = new ArrayList<>();

    while(firstIdx < firstList.length && secondIdx < secondList.length){
        int s1 = firstList[firstIdx][0];
        int e1 = firstList[firstIdx][1];
        int s2 = secondList[secondIdx][0];
        int e2 = secondList[secondIdx][1];

        int start = Math.max(s1, s2);
        int end = Math.min(e1, e2);

        if(start <= end){
            intersect.add(new int[]{start, end});
        }

        if(e1 < e2) firstIdx++;
        else secondIdx++;
    }

    int[][] res = new int[intersect.size()][2];
    for(int i=0; i<intersect.size(); i++){
        res[i] = intersect.get(i);
    }
    return res;
}
```

1. SRS - GitHub
2. Notes
3. Recordings
4. CB videos
5. R & B - level 2

Circular Tour {has station} {Cost, gas}



journey : 0 : (5,6) → (6,7) → (7,4) → ∝(8,11)
0 → 1 → 2 → 3
6-5 = 1
8-6 = 2
6-7 = -1

∝ journey : 1 : (6,7) → (7,4) → ∝
1 → 2
7-6 = 1
5-7 = -2

∝ journey 2 : (7,4) → ∝
2 → 3
4-7 = -3

journey 3 : (8,10) → (6,6) → (4,5) → (5,6) → (6,7) → (7,4) → (8,10)
3 → 0 → 1 → 2 → 3
10-8 = 2
8-6 = 2
6-4 = 2
5-6 = -1
7-6 = 1
4-7 = -3
(+1)

Circular Tour

distance, petrol

0	1	2	3	4	5
(5, 6)	(6, 7)	(7, 4)	(8, 10)	(6, 6)	(4, 5)

journey : 0th

+6 —(-5)→ +1+7 —(-6)→ +2+4 —(-7)→ -1

0 1 2 3

journey : 1st

+2 —(-6)→ +1+4 —(-7)→ -2

1 2 3

journey : 2nd

+4 —(-7)→ -3

2 3

journey : 3rd

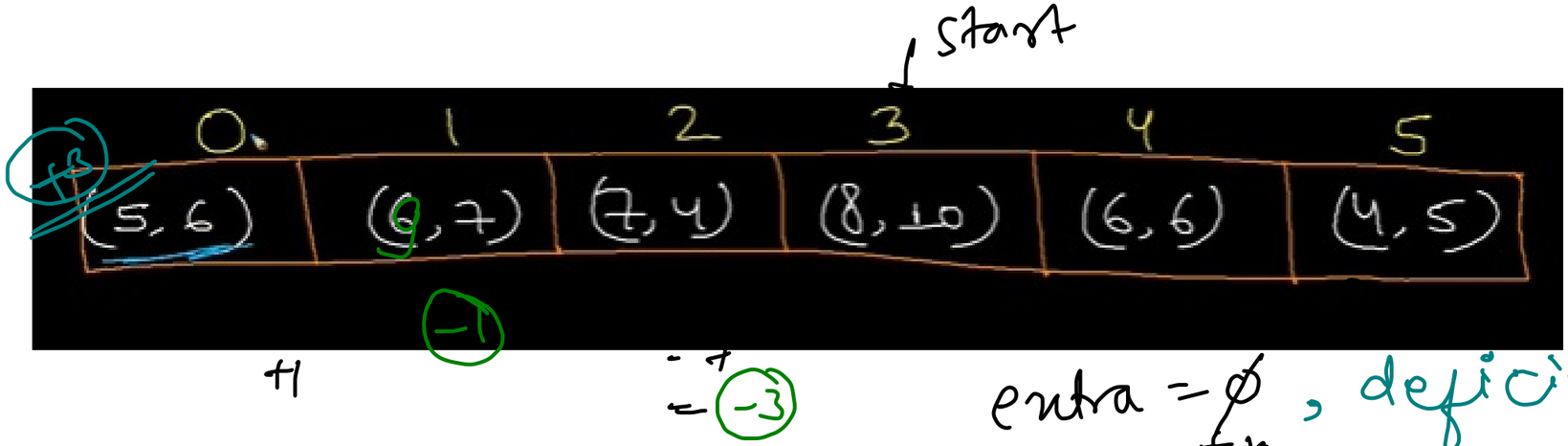
+10 —(-8)→ +2+6 —(-6)→ +2+5 —(-4)→ +3 —(-5)→ +6 0

3 4 5

+4 —(-6)→ +7 —(-5)→ +2 3

1 2

$O(N^2) \rightarrow O(2N) \rightarrow O(N)$




```java
public int canCompleteCircuit(int[] gas, int[] cost) {
    int extra = 0, start = 0, deficit = 0;

    for(int i=0; i<gas.length; i++){
        extra = extra + (gas[i] - cost[i]);

        if(extra < 0){
            start = i + 1;
            deficit -= extra;
            extra = 0;
        }
    }

    if(extra >= deficit) return start;
    return -1;
}
```

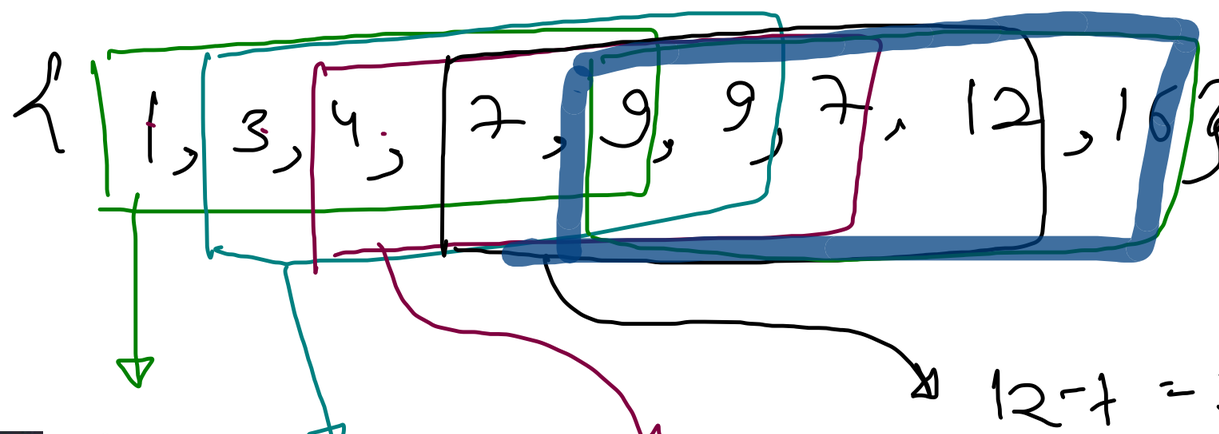
Chocolate Distribution

{ 3, 4, 1, 9, 16, 7, 9, 12 }

Codeforces → A, B

5

OA { Sorting, Searching, Greedy }



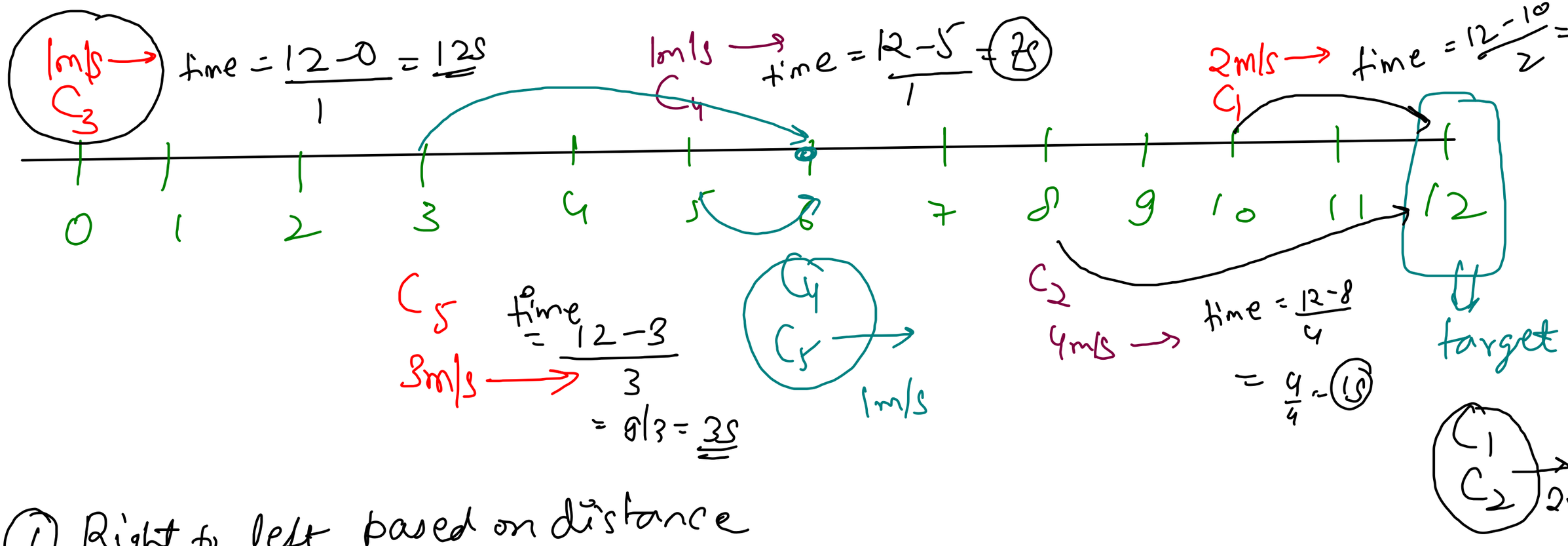

```java
public long findMinDiff (ArrayList<Long> a, long n, long m)
{
    Collections.sort(a);
    long ans = Long.MAX_VALUE;
    for(int w=0; w<=(int)(n-m); w++){
        long Max = a.get(w + (int)m - 1);
        long Min = a.get(w);

        ans = Math.min(ans, Max - Min);
    }
    return ans;
}
```

→ O(NlogN)

O(N-M)

Car Fleet

```
target = 12, position = [10,8,0,5,3], speed = [2,4,1,1,3]
```



(1) Right to left based on distance

(2) Double

(3) maxTime, count of Groups

maxTime = ~~1s~~ ~~2s~~ 12s

Count of Group = ~~1~~ ~~2~~ 3

this - other → min Heap, Inc order

other - this → max Heap, Dec order

```java
public static class Pair implements Comparable<Pair>{
    int distance;
    int speed;
    double time;

    Pair(int distance, int speed){
        this.distance = distance;
        this.speed = speed;
        this.time = ((distance * 1.0) / speed);
    }

    public int compareTo(Pair other){
        return this.distance - other.distance;
    }
}
```

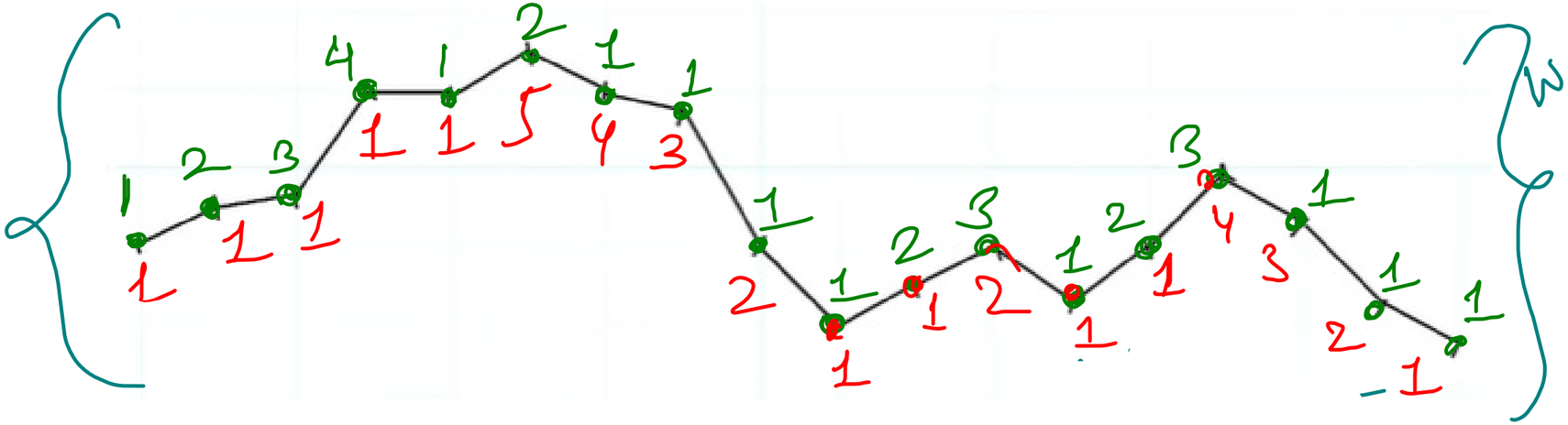
```java
public int carFleet(int target, int[] position, int[] speed) {
    int n = position.length;
    Pair[] cars = new Pair[n];

    for(int i=0; i<n; i++){
        cars[i] = new Pair(target - position[i], speed[i]);
    }

    Arrays.sort(cars);

    double maxTime = 0.0;
    int countOfGroups = 0;
    for(int i=0; i<n; i++){
        if(cars[i].time > maxTime){
            countOfGroups++;
            maxTime = cars[i].time;
        }
    }
    return countOfGroups;
}
```

O(N) → O(N)

O(NlogN)

break till 11:30.

Candy / Temple offerings



WHAT

left to right

i → left neighbour

1+2+3+4+1+5+4+3+2+1+2+3+1+2+4+3+2+1

right to left :→

right neighbour

There are `n` children standing in a line. Each child is assigned a rating value given in the integer array `ratings`.

You are giving candies to these children subjected to the following requirements:

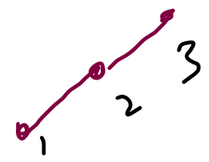
- Each child must have at least one candy.
- Children with a higher rating get more candies than their neighbors.

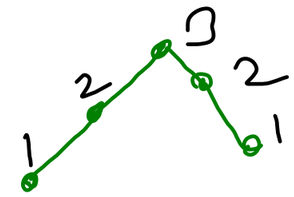
Return *the minimum number of candies you need to have to distribute the candies to the children*.

```java
public int candy(int[] ratings) {
    int[] left = new int[ratings.length];
    int[] right = new int[ratings.length];

    left[0] = 1;
    for(int i=1; i<ratings.length; i++){
        if(ratings[i] > ratings[i - 1]){
            left[i] = left[i - 1] + 1;
        } else {
            left[i] = 1;
        }
    }

    right[ratings.length - 1] = 1;
    for(int i=ratings.length-2; i>=0; i--){
        if(ratings[i] > ratings[i + 1]){
            right[i] = right[i + 1] + 1;
        } else {
            right[i] = 1;
        }
    }

    int sum = 0;
    for(int i=0; i<ratings.length; i++){
        sum += Math.max(left[i], right[i]);
    }
    return sum;
}
```

{ 10, 40, 60 }

1 2 3



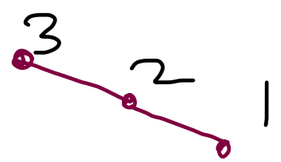


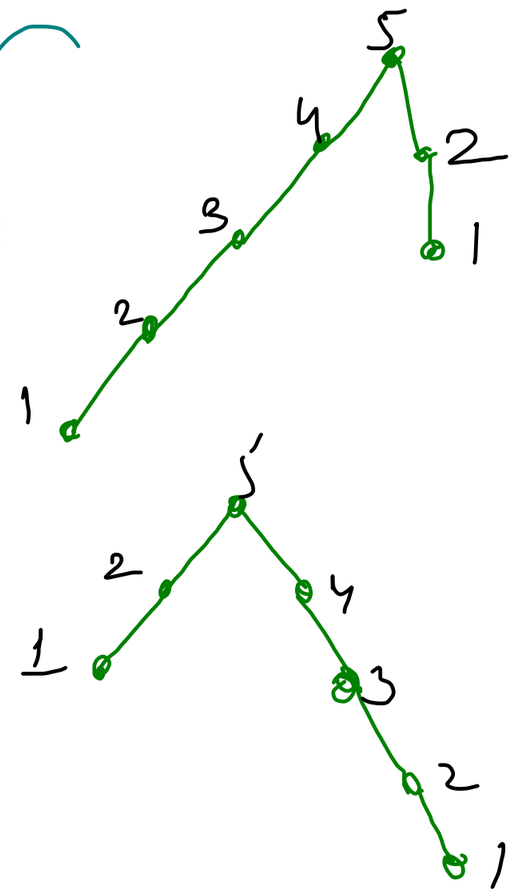
{ 10, 30, 70, 60, 20 }

1 2 3 2 1

{ 50, 30, 20 }

3 2 1





{ 10, 30, 70, 90, 60, 50, 40 }

1 2 3 4 5 2 1

{ 20, 20, 20 }

1 1 1

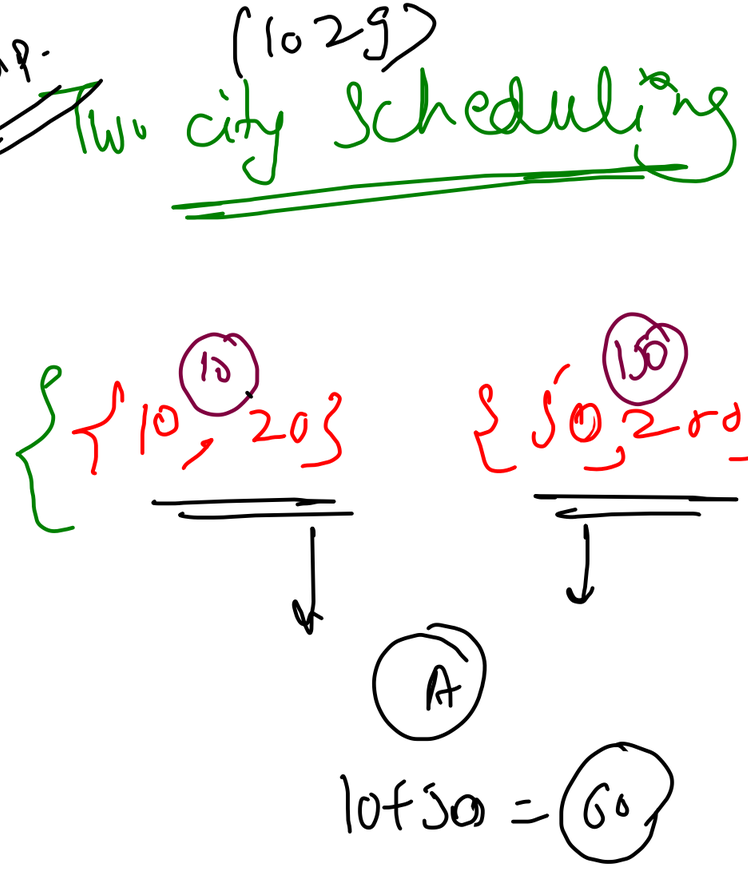
WHY

{ 10, 90, 100, 80, 60, 40, 30 }

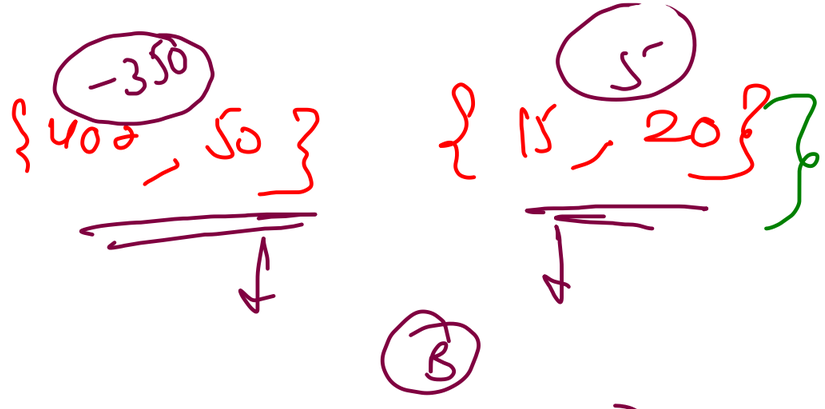
1 2 5 4 3 2 1

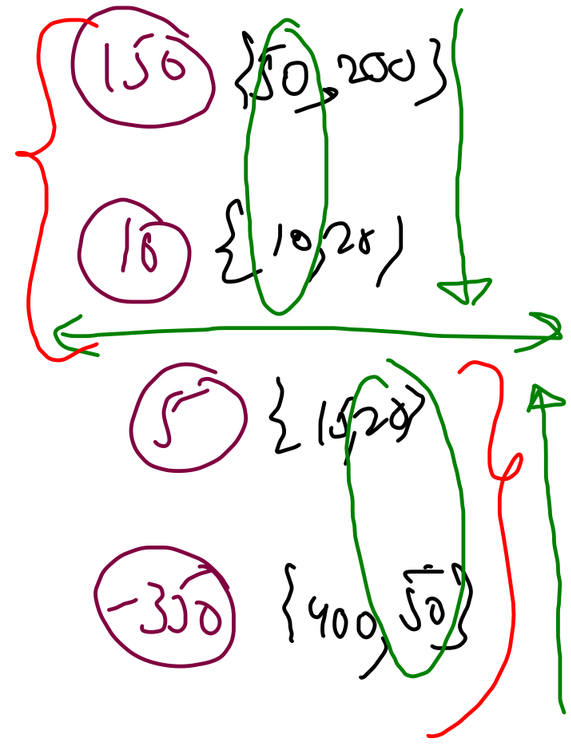
IMP.

Two city Scheduling (1029)



sort on the basis of net gain in going to A wrt B.



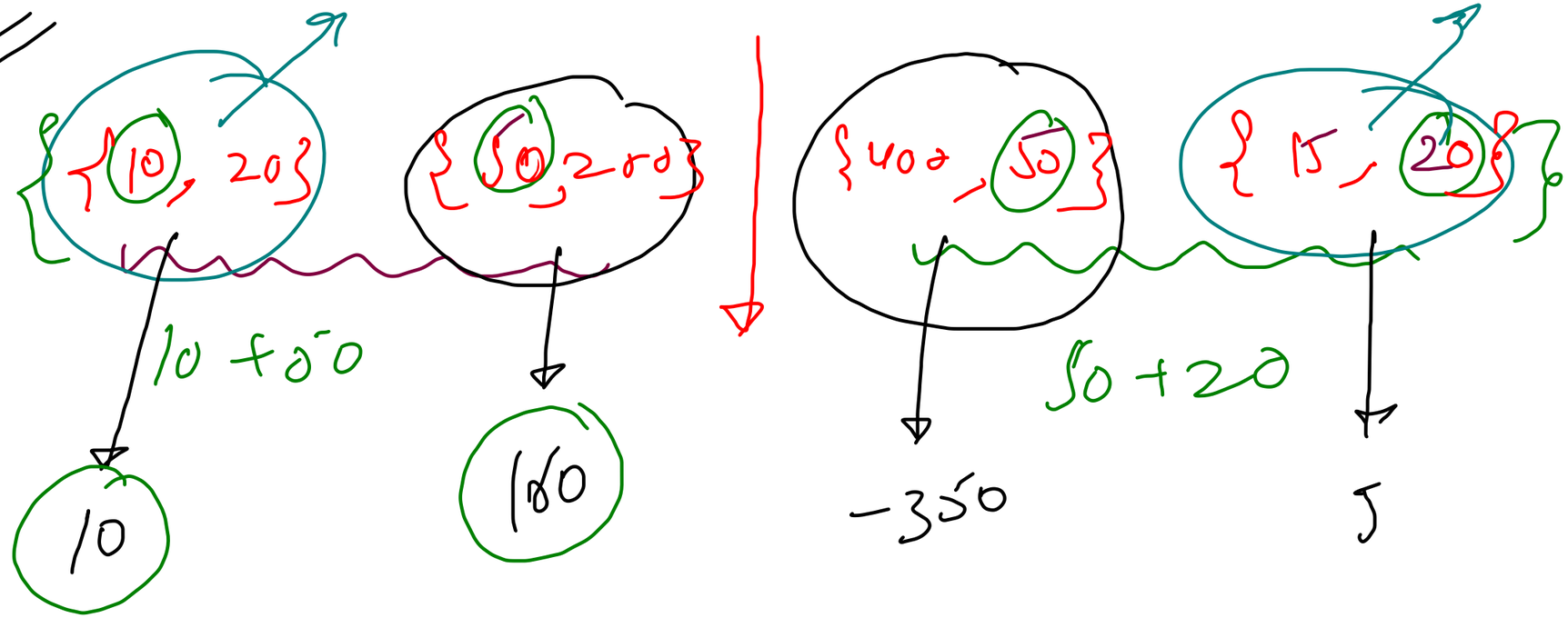


A company is planning to interview `2n` people. Given the array `costs` where `costs[i] = [aCost_i, bCost_i]`, the cost of flying the `i^th` person to city `a` is `aCost_i`, and the cost of flying the `i^th` person to city `b` is `bCost_i`.

Return *the minimum cost to fly every person to a city* such that exactly `n` people arrive in each city.

```java
public int twoCitySchedCost(int[][] costs) {
    Arrays.sort(costs, (x, y) ->  ((y[1] - y[0]) - (x[1] - x[0])));
                                                  other − this
    int sum = 0;
    for(int i=0; i<costs.length/2; i++){
        sum += costs[i][0];
        sum += costs[costs.length - 1 - i][1];
    }
    return sum;
}
```

IMP.



$N = 10^5$

{50, 200} → 150

{10, 20} → 10

{15, 20} → 5

{400, 50} → -350

lecture (4)

1. Biased Standings
2. Defense Kingdom
3. Georgivia
4. Amplifiers
5. load Balancing

} SPOJ

Huffman Coding & Decoding

Sunday Evening

Biased Standings

Usually, results of competitions are based on the scores of participants. However, we are planning a change for the next year of IPSC. During the registration each team will be able to enter a single positive integer : their preferred place in the ranklist. We would take all these preferences into account, and at the end of the competition we will simply announce a ranklist that would please all of you.

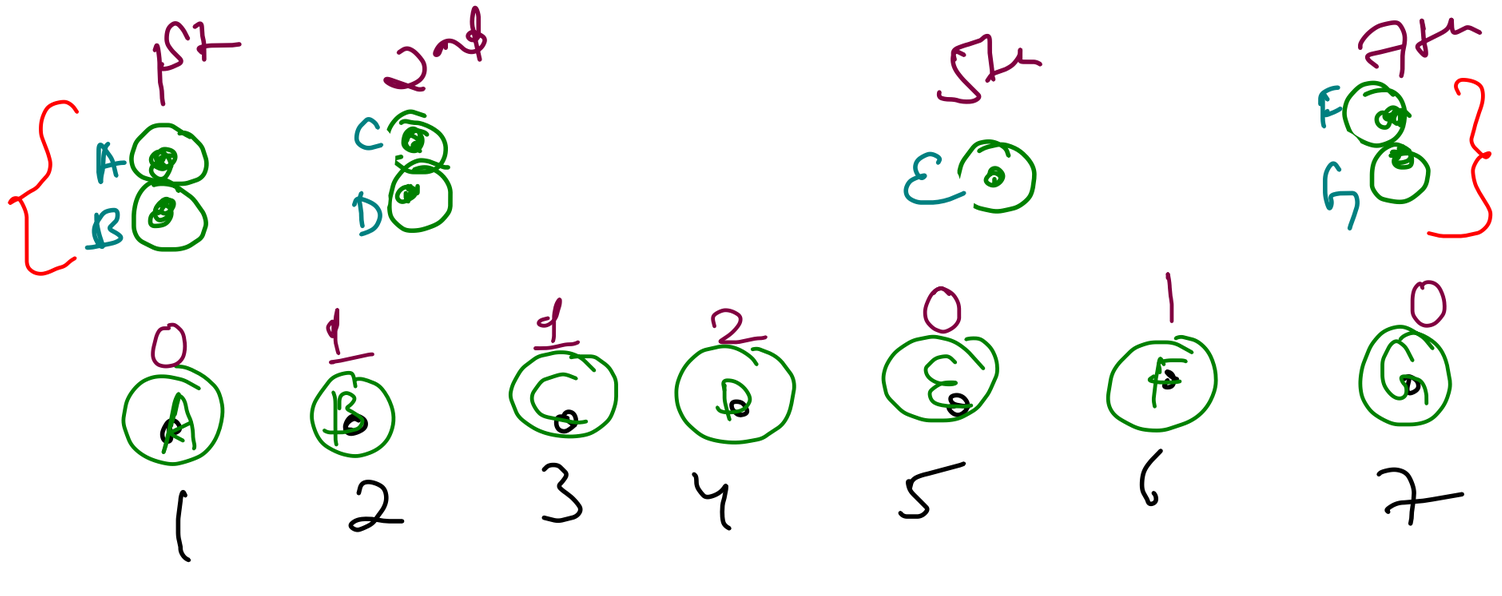
But wait... How would that ranklist look like if it won't be possible to satisfy all the requests?

Suppose that we already have a ranklist. For each team, compute the distance between their preferred place and their place in the ranklist. The sum of these distances will be called the badness of this ranklist.

$1 \le a[i].rank \le N$

input

```
7
noobz 1
llamas 2
Winn3rz 2
5thwheel 1
NotoricCoders 5
StrangeCase 7
WhoKnows 7
```



HINT : left to right, total badness

$0+1+1+2+1 = 5$

B → 2 C → 3 = B → 3 C → 2

2-1=1 3-2=1

1+1=2

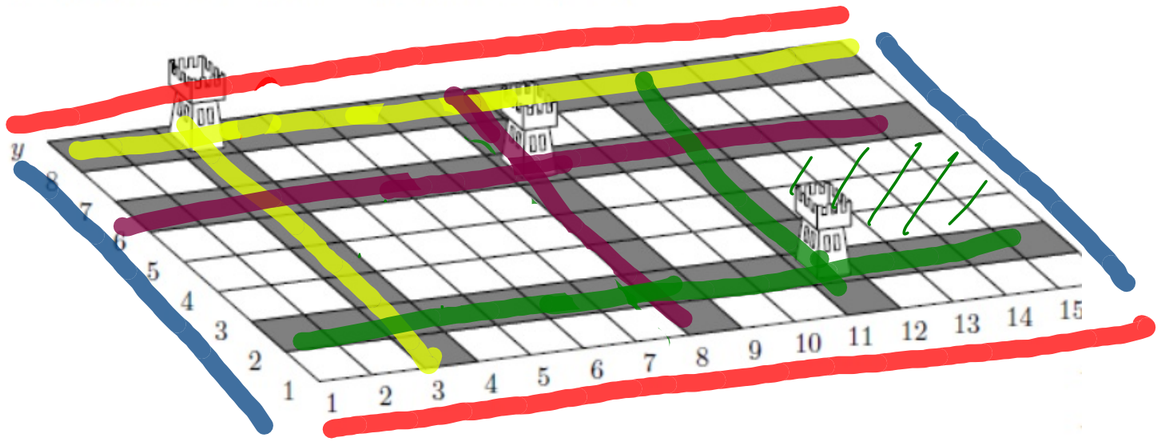
3-1=2 2+2=0

2+0=2

Defense of Kingdom

Theodore implements a new strategy game “Defense of a Kingdom”. On each level a player defends the Kingdom that is represented by a rectangular grid of cells. The player builds crossbow towers in some cells of the grid. The tower defends all the cells in the same row and the same column. No two towers share a row or a column.

The penalty of the position is the number of cells in the largest undefended rectangle. For example, the position shown on the picture has penalty 12.



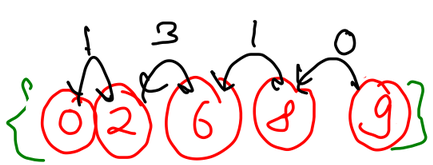
Help Theodore write a program that calculates the penalty of the given position.

HINT

Row & Column are independent ★

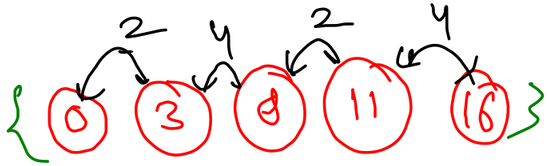
Corner case

Rectangle may or may not be surrounded by towers



rows

max^m diff b/w adj rows = max^m length = 3



cols

max^m diff b/w adj col = max^m breadth = 4

★

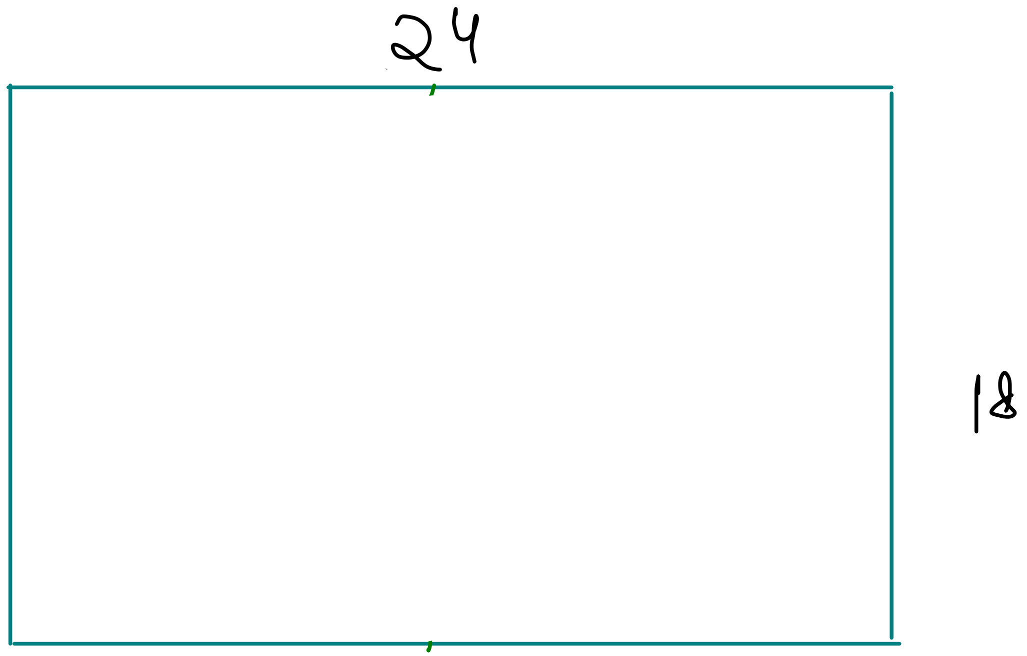
```java
int length = scn.nextInt();
int breadth = scn.nextInt();
int tower = scn.nextInt();

ArrayList<Integer> rows = new ArrayList<>();
rows.add(0);
rows.add(length + 1);
ArrayList<Integer> cols = new ArrayList<>();
cols.add(0);
cols.add(breadth + 1);

for (int i = 0; i < tower; i++) {
    int row = scn.nextInt();
    int col = scn.nextInt();
    rows.add(row);
    cols.add(col);
}

Collections.sort(rows);
Collections.sort(cols);
int maxLength = 0, maxBreadth = 0;
for (int i = 1; i < rows.size(); i++) {
    maxLength = Math.max(maxLength, rows.get(i) - rows.get(i - 1) - 1);
    maxBreadth = Math.max(maxBreadth, cols.get(i) - cols.get(i - 1) - 1);
}
System.out.println(maxLength * maxBreadth);
```

O(NlogN)

O(N)


```java
public int mice(int[] A, int[] B) {
    Arrays.sort(A);
    Arrays.sort(B);

    int maxTime = 0;
    for(int i=0; i<A.length; i++){
        maxTime = Math.max(maxTime, Math.abs(A[i] - B[i]));
    }
    return maxTime;
}
```

Cutting Rectangle



1*1 ⇒ 24*9

2*2 ⇒ $\frac{(L*B)}{(2*2)} = \frac{24*18}{2*2}$ = 108

3*3 ⇒ $\frac{24*18}{3*3}$ = 48

6*6 ⇒ $\frac{24*18}{6*6}$ = 12

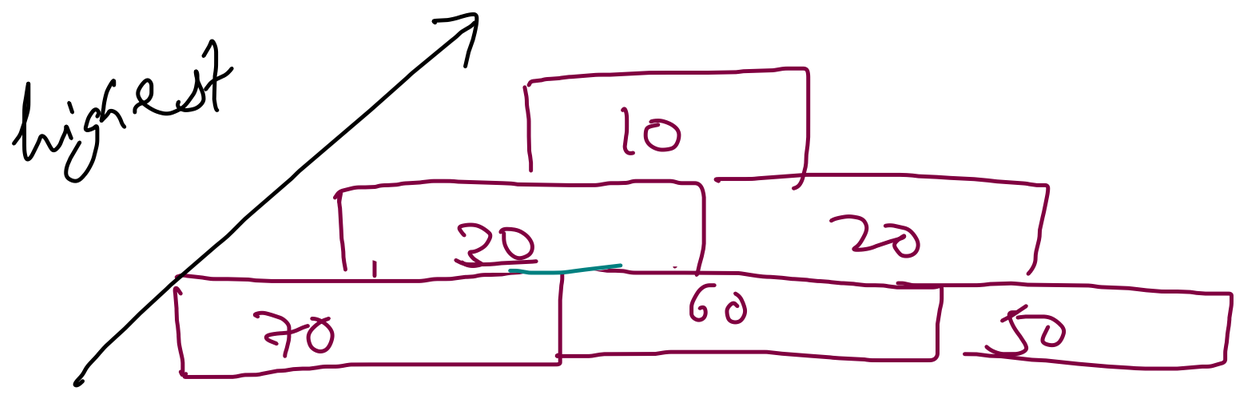
gcd

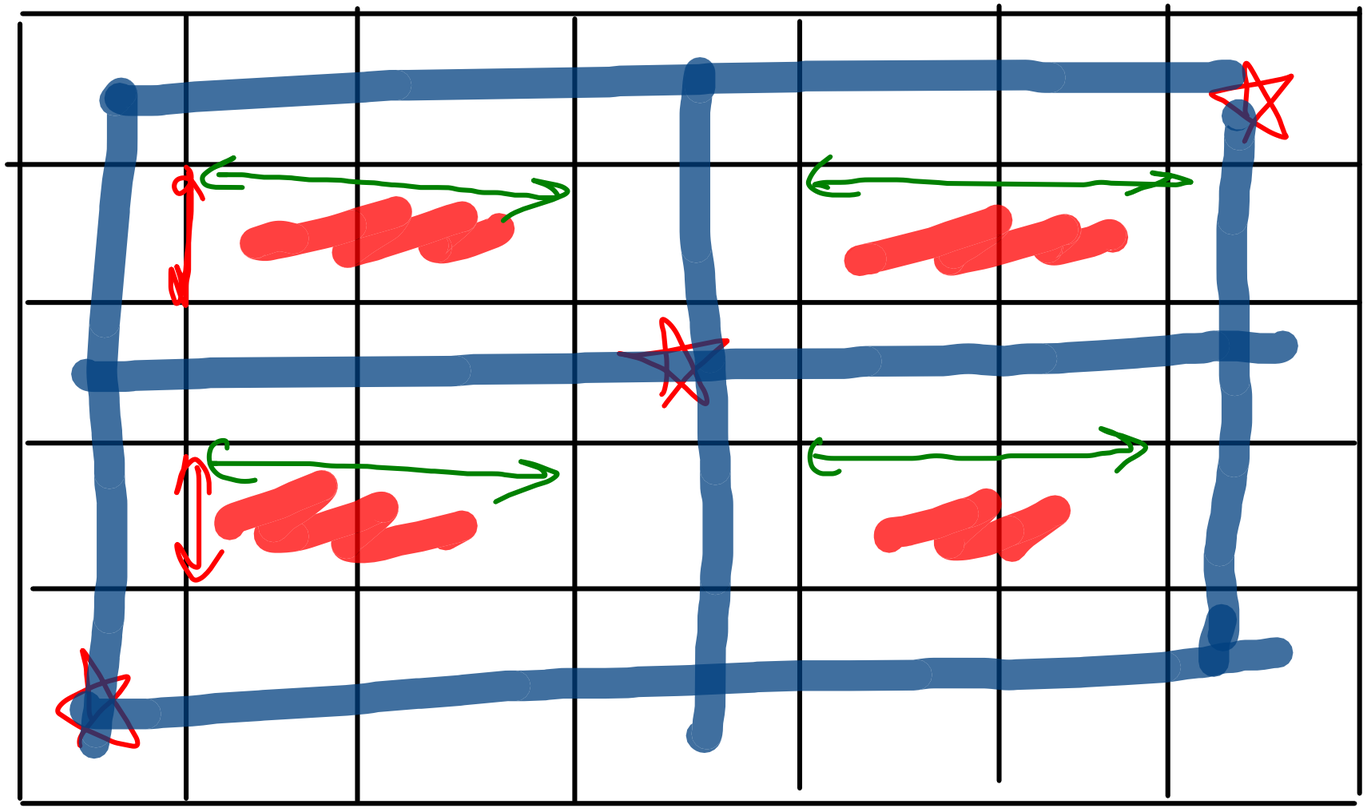
```
public static long gcd(long a, long b){
    if(b == 0) return a;
    return gcd(b, a % b);
}

static List<Long> minimumSquares(long L, long B)
{
    Long squareSide = gcd(L, B);
    Long noOfSquare = (L * B) / (squareSide * squareSide);

    ArrayList<Long> res = new ArrayList<>();
    res.add(noOfSquare); res.add(squareSide);
    return res;
}
```

Highest Pyramid

1. i^{th} level no of blocks > $(i-1)^{th}$ level of no blocks
2. i^{th} level breadth > $(i-1)^{th}$ level breadth

```
{10, 20, 30, 50, 60, 70}
```





length = 1

breadth = 2

Wine Trading

Gergovia consists of one street, and every inhabitant of the city is a wine salesman. Everyone buys wine from other inhabitants of the city. Every day each inhabitant decides how much wine he wants to buy or sell. Interestingly, demand and supply is always the same, so that each inhabitant gets what he wants.

There is one problem, however: Transporting wine from one house to another results in work. Since all wines are equally good, the inhabitants of Gergovia don't care which persons they are doing trade with, they are only interested in selling or buying a specific amount of wine.

In this problem you are asked to reconstruct the trading during one day in Gergovia. For simplicity we will assume that the houses are built along a straight line with equal distance between adjacent houses. Transporting one bottle of wine from one house to an adjacent house results in one unit of work.

Arranging Amplifiers

Scientists at the TIFR, Mumbai, are doing some cutting edge research on the Propagation of Signals. A young researcher comes up with a method of progressively amplifying signals, as they progress along a path. The method involves the placing of Amplifiers at regular distances along the line. Each amplifier is loaded with a number a(i), which is called its amplification factor. The method of amplification is simple: an amplifier which recieves a signal of strength X, and has Y loaded in it, results in a signal of strength Y^X [Y to the power X]. In course of his research, the young scientist tries to find out, that given a set of n amplifiers loaded with a(0), a(1), a(2), ..., a(n-1), which particular permutation of these amplifiers, when placed at successive nodes, with the initial node given a signal of strength 1, produces the strongest output signal.

this is better illustrated by the following example : 5 6 4
4^(5^(6^1)) is the strength of the strongest signal, which is generated by putting amplifier loaded with 6 in first place, 5 in second place and 4 in third place.

Given a list of integers specifying the set of amplifiers at hand, you must find out the order in which they must be placed, to get the highest signal strength. In case their exist multiple permutations with same output, you should print the one which has bigger amplifiers first.

Load Balancing

SuperComputer Inc. have built a super-fast computer server consisting of N hyper-scalar lightning-fast processors Beta 007. These processors are numbered from 1 to N and are used to process independent jobs. Every new incoming job is assigned to an arbitrary processor. Sometimes, a processor may be assigned too many jobs while other processors have a relatively light load (or even wait idly). In that case, the whole system undergoes rebalancing.

Rebalancing proceeds in rounds. In each round, every processor can transfer at most one job to each of its neighbors on the bus. Neighbors of the processor i are the processors i-1 and i+1 (processors 1 and N have only one neighbor each, 2 and N-1 respectively). The goal of rebalancing is to achieve that all processors have the same number of jobs.

Given the number of jobs initially assigned to each processor, you are asked to determine the minimal number of rounds needed to achieve the state when every processor has the same number of jobs, or to determine that such rebalancing is not possible.

Huffman Encoding & Decoding

Input String

BCCABBDDAECCBBAEDDCC

1 char → 1 byte ASCII

↓

4 bits

Encrypted → Decrypted

160 bits

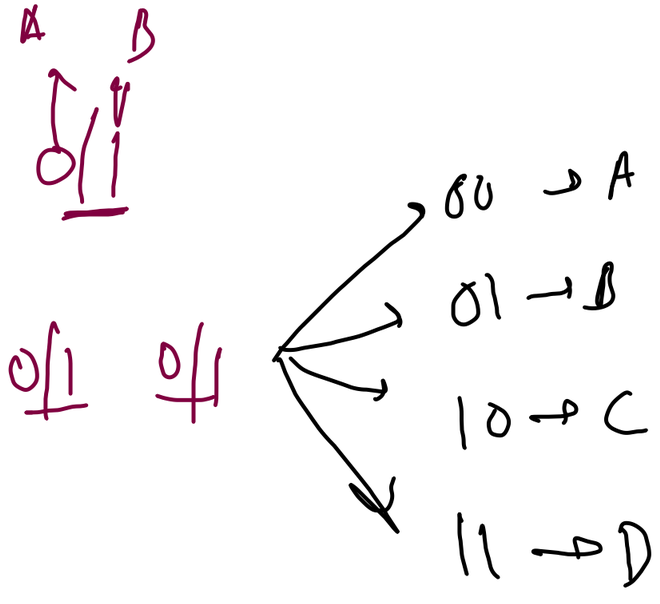
↓

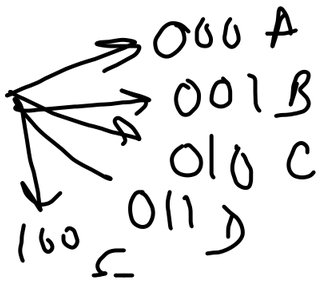
Shortest possible encoded string

< 160 bits

① Fixed size Encoding

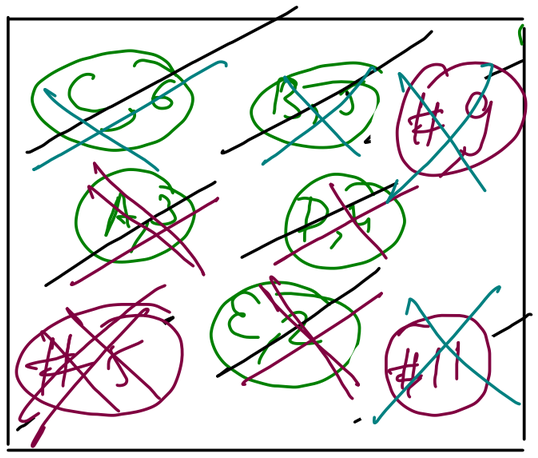
20 length, 3 bits = 60 bits





② Variable sized Encoding

BCCABBDDAECCBBAEDDCC



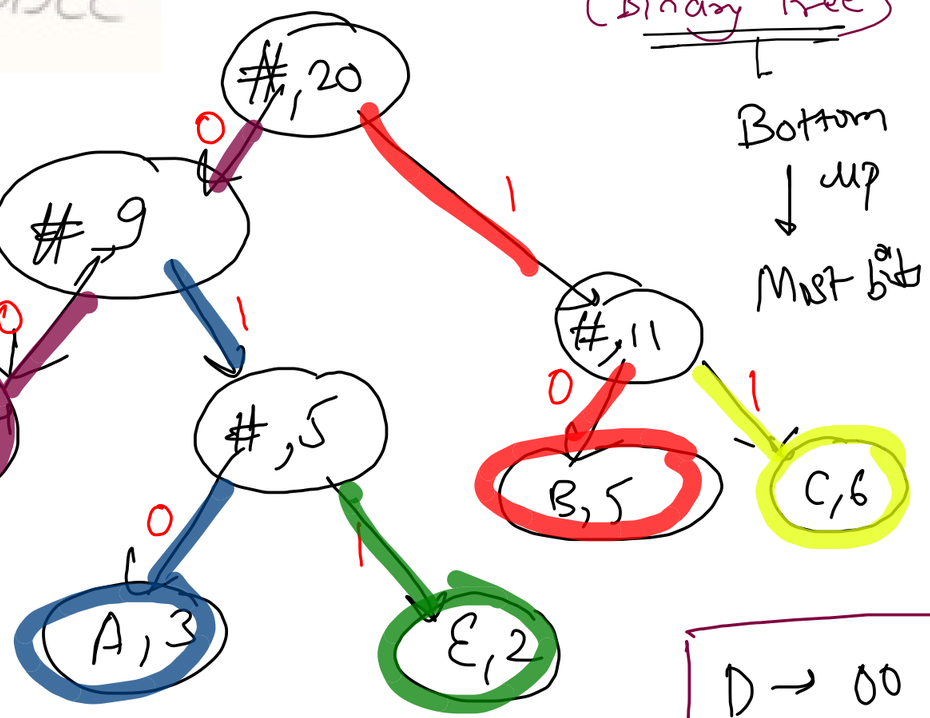
Min Heap

Bottom level, Low freq, High Bit

Top level, High freq, Less Bit

Preprocessing
Encoding
(Binary Tree)

Bottom
↓ up
Most bit



4*2 = 8

3*3 = 9

2*3 = 6

5*2 = 10

6*2 = 12

45 bits

D → 00
A → 010
E → 011
B → 10
C → 11

10 11 11 010---

```java
public static class Node implements Comparable<Node>{
    char ch;
    // int idx;
    int freq;
    Node left;
    Node right;

    Node(char ch, int freq){
        this.ch = ch;
        // this.idx
        this.freq = freq;
        this.left = null;
        this.right = null;
    }

    public int compareTo(Node other){
        return this.freq - other.freq;
    }
}
```

Encoding

```java
PriorityQueue<Node> q = new PriorityQueue<>();
for(int i=0; i<N; i++){
    q.add(new Node(S.charAt(i), f[i]));
}

while(q.size() > 1){
    Node left = q.remove();
    Node right = q.remove();
    Node root = new Node('#', left.freq + right.freq);
    root.left = left;
    root.right = right;
    q.add(root);
}

Node root = q.remove();
encoding = new ArrayList<>();
preorder(root, "");

return encoding;
```

O(NlogN)

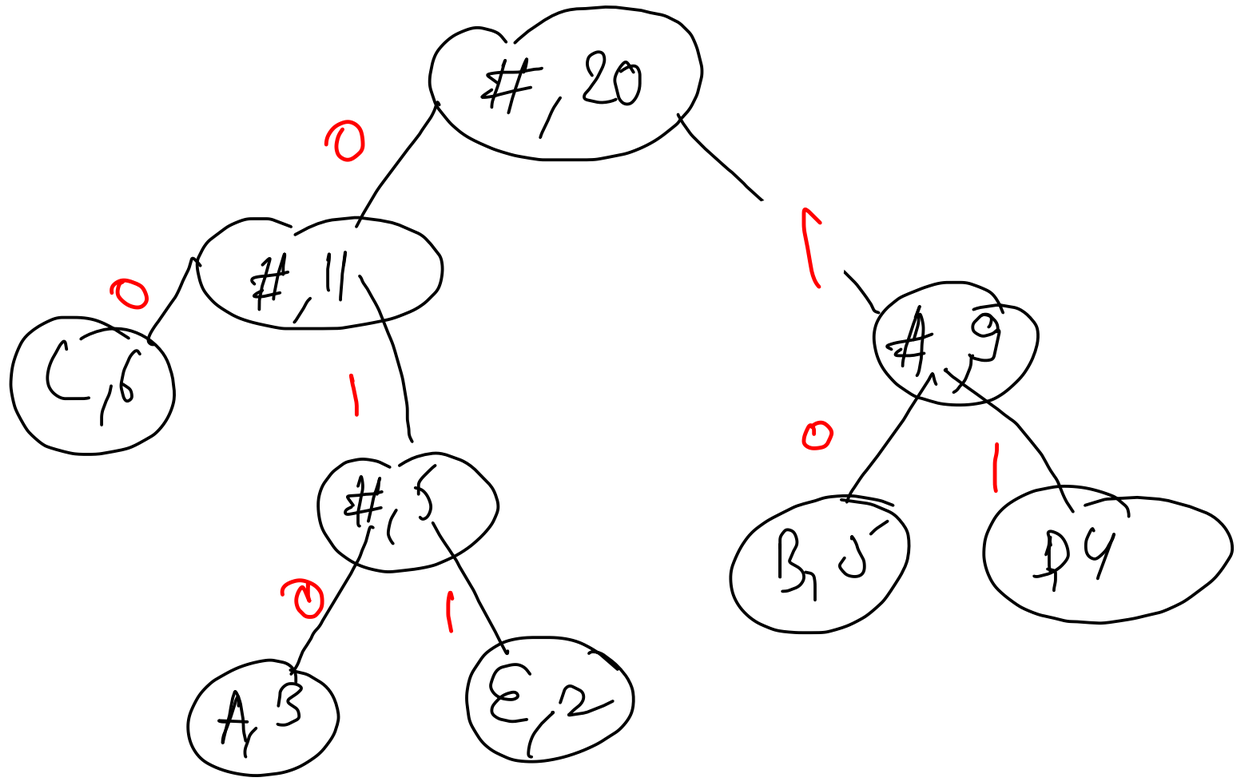
O(NlogN)

```java
ArrayList<String> encoding;
public void preorder(Node root, String str){
    if(root.left == null && root.right == null){
        encoding.add(str);
        return;
    }

    if(root.left != null)
        preorder(root.left, str + "0");

    if(root.right != null)
        preorder(root.right, str + "1");
}
```

3 -> O(N)

Decoding

```
int idx = 0;
string decodedString = "";
while(idx < encodedString.size())
{
    MinHeapNode* curr = root;
    while(curr->data == '$')
    {
        if(encodedString[idx] == '0')
            curr = curr->left;
        else curr = curr->right;
        idx++;
    }
    decodedString = decodedString + curr->data;
}
return decodedString;
```



C → 00 6*2 = 12

A → 010 3*3 = 9

E → 011 2*3 = 6

B → 10 5*2 = 10

D → 11 4*2 = 8

45 bits