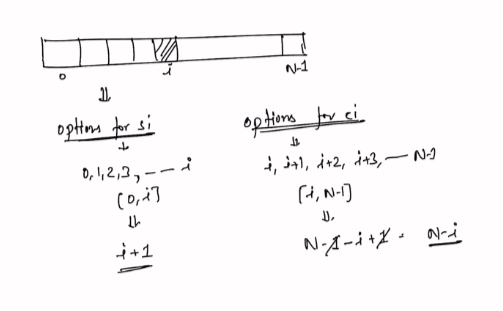
**Subarrays**  
  
1. Sum of all subarrays

Intermedite part 1/Subarray/questions/Sum of all subarrays.txt

In this consider possible start indexes and end indexes



2. Subarray Sum Equals K

Intermediate part 2/Hashing/answers/Subarray Sum Equals K.txt

3. Subsequence-Sum Problem

Intermediate part 2/Subsets and Subsequence/answers/Subsequence-Sum Problem.txt

4.Subset

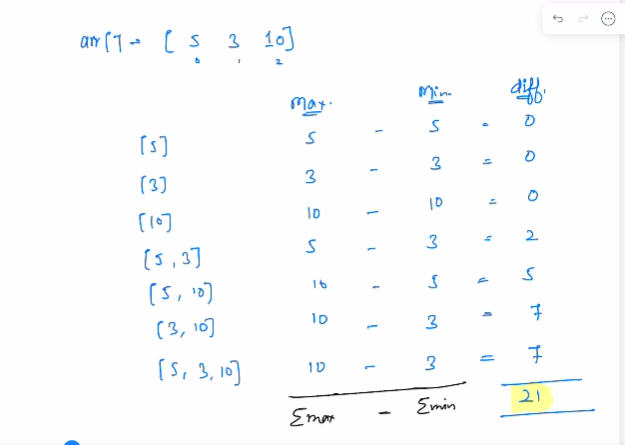
Intermediate part 2/Subsets and Subsequence/answers/Subsets.txt

5. Factors sort (using custom comparator)

Intermediate part 2/Sorting/answer/Factors sort.txt

6. Sum the difference

Intermediate part 2/Subsets and Subsequence/answers/Sum the Difference.txt

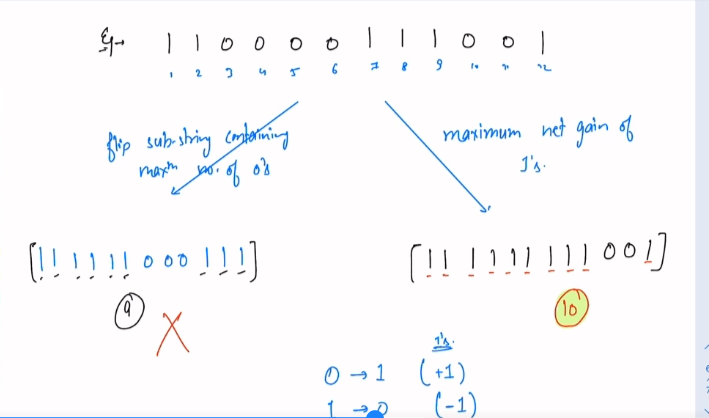


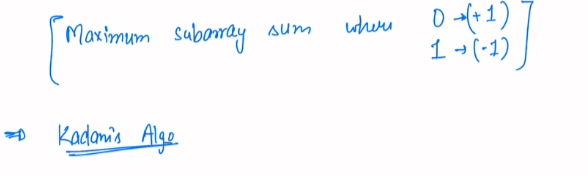
7. Counting Nodes

Intermediate part 2/Trees/answers/Counting nodes.txt

8. Flip

Advanced part 1/Arrays/one dimensional/questions/Flip.txt





9. Max Submatrix Sum

Advanced part 1/Arrays/two dimensional/questions/Maximum Submatrix Sum.txt

10. Minimum Swaps

Advanced part 1/Arrays/two dimensional/answers/Minimum Swaps.txt

11. Merge Intervals

Advanced part 1/Arrays/interview problems/answers/Merge Intervals.txt

12. Merge Overlapping Intervals

Advanced part 1/Arrays/interview problems/answers/Merge Overlapping Intervals.txt

13.Trapping Rain Water

Advanced part 1/Arrays/interview problems/answers/Rain Water Trapped.txt

14. 3Sum

Advanced part 2/2 Pointers/answers/3 Sum.txt

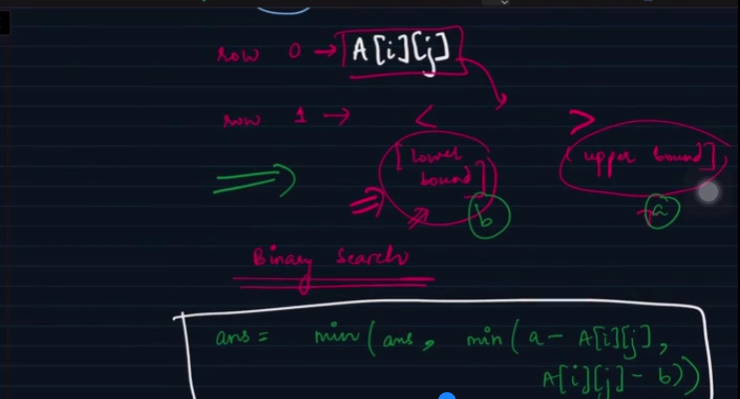
15. Pair with given Difference

Advanced part 2/2 Pointers/answers/Pairs with Given Difference.txt

16. Pairs with given sum II  
Advanced part 2/2 Pointers/answers/Pairs with given sum II.txt

17. Minimum Difference

Advanced part 2/Binary Search1/answers/Minimum Difference.txt

  
When we are doing binary search and rows are sorted in increasing fashion. If there is any element 5 and array is [1,3,4,9]. Remember that applying binary search if the exact element is not found, we always land at the upper bound ie: nearest element greater than the ele we want to find

18. Rotated Sorted Array Search

Advanced part 2/Binary Search1/answers/Rotated Sorted Array Search.txt

19. Search for range

Advanced part 2/Binary Search1/answers/Search for a Range.txt

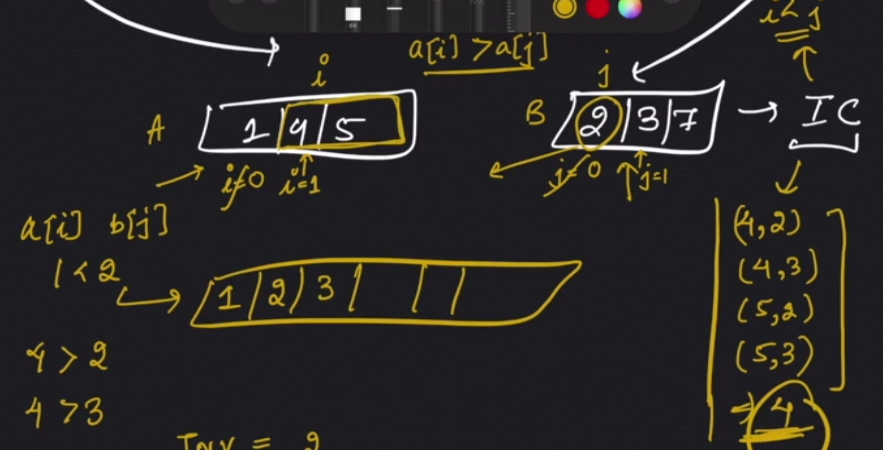
20. Sorted Insert Position

Advanced part 2/Binary Search1/answers/Sorted Insert Position.txt

21. Aggressive Cows

Advanced part 2/Binary Search2/answers/Aggressive cows.txt

22. Inversion Count



Advanced part 2/Selection, Merge Sort, Insertion and Radix Sort/answers/Inversion count in an array.txt

23. Merge Sort

Advanced part 2/Selection, Merge Sort, Insertion and Radix Sort/answers/Merge Sort.txt

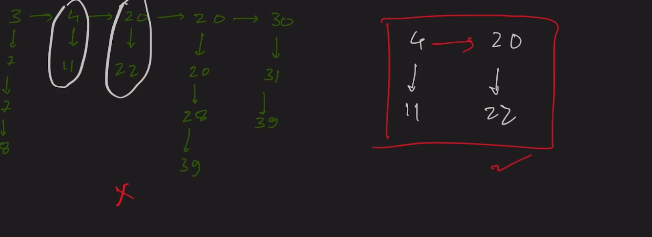
**Adv DSA 3**

1. Copy List

Advanced part 3/Link List/answers/Copy List.txt

1. Flatten a link list

Advanced part 3/Link List/answers/Flatten a linked list.txt



1. Longest Palindromic List

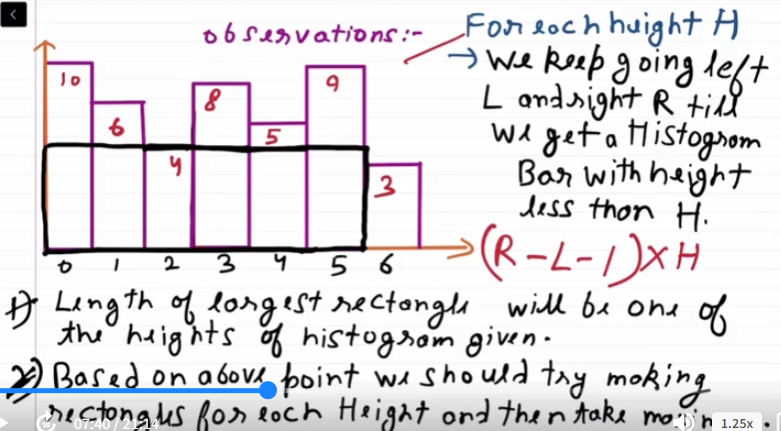
Advanced part 3/Link List/answers/Longest Palindromic List.txt

1. Merge 2 sorted lists

Advanced part 3/Link List/answers/Merge Two Sorted Lists.txt

1. Reverse Link List II  
   Advanced part 3/Link List/answers/Reverse Link List II.txt
2. Largest Rectangle in Histogram

Advanced part 3/stacks/answers/Largest Rectangle in Histogram.txt



Note:   
  
for left if no min is found

if(stk.empty()) {

arr[i] = -1;

}

For right if no min is found

if(stk1.empty()){

arr1[i]=A.length;

}

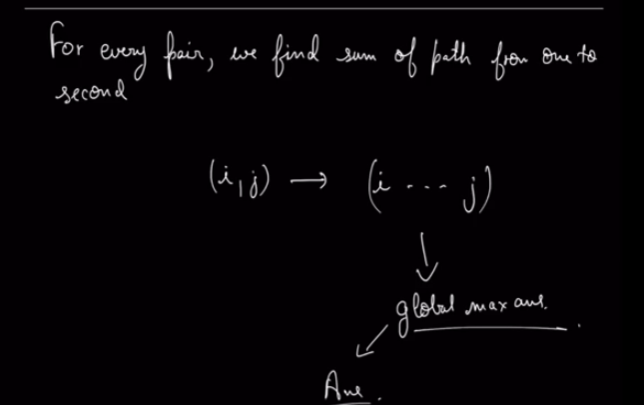
1. Next Greater Element

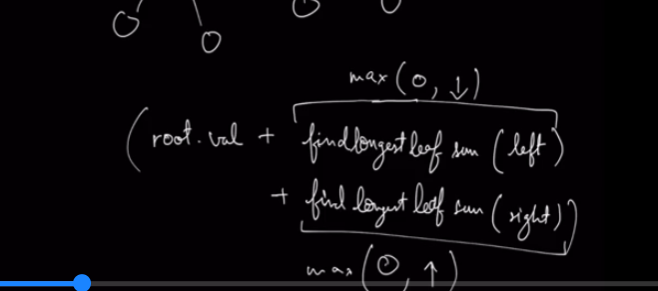
Advanced part 3/stacks/answers/Next Greater.txt

**Trees**

1. Max Sum Path in Binary Tree

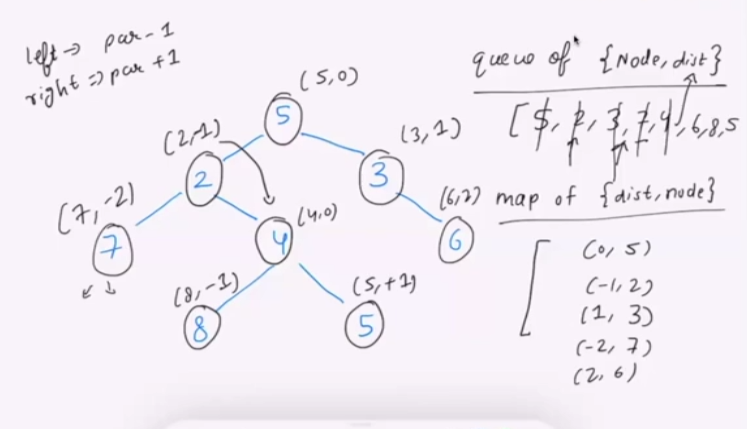
Advanced part 3/Trees/Views and Types/answers/Max Sum Path in Binary Tree.txt



  
  
We take 0 into comparison because if a path returns sum as -ve value we should not consider it.

1. Top View of Binary Tree

Advanced part 3/Trees/Views and Types/answers/Top View of Binary tree.txt



For every node we keep assigning a distance. And store the next level of node which need to explored in a queue. Suppose there is already a node present in the map for a given distance then we do not assign it again.

**BST and BBST**

**BST (Binary Search Tree)** and **BBST (Balanced Binary Search Tree)** are both types of binary trees used in computer science for efficient searching, insertion, and deletion operations. However, they have distinct differences in terms of structure and performance:

**1. Binary Search Tree (BST):**

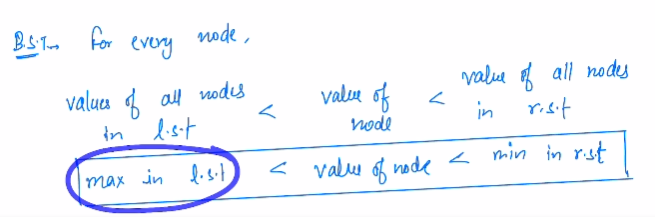
* **Structure**: A BST is a binary tree where each node has at most two children, and the left child is less than the parent node, while the right child is greater.
* **Performance**: The time complexity of operations (search, insert, delete) depends on the height of the tree. In the worst case (when the tree becomes skewed), the height can be O(n), leading to O(n) time complexity.
* **Unbalanced**: A regular BST does not ensure that the tree remains balanced, so if nodes are inserted in a sorted order, it can degrade to a linked list-like structure.

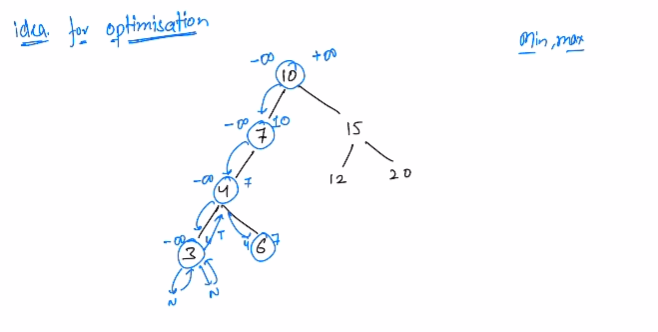
**2. Balanced Binary Search Tree (BBST):**

* **Structure**: A BBST is a type of BST where the height of the tree is kept balanced. This means the difference in height between the left and right subtrees of any node is small (usually 1 or less).
* **Performance**: BBST ensures that the tree height remains logarithmic (O(log n)), which guarantees that operations like search, insert, and delete all have O(log n) time complexity.
* **Types of BBSTs**: There are various types of balanced trees, such as:
  + **AVL Tree**: Maintains strict balancing by ensuring the height difference between left and right subtrees of any node is at most 1.
  + **Red-Black Tree**: A less strictly balanced tree that allows some imbalance but still guarantees logarithmic height.
  + **B-Trees**: Used in databases and file systems, B-trees keep data sorted and allow search, insertion, and deletion in logarithmic time.

For a **Binary Search Tree (BST)**, an **in-order traversal** will always produce the node values in increasing (non-decreasing) order

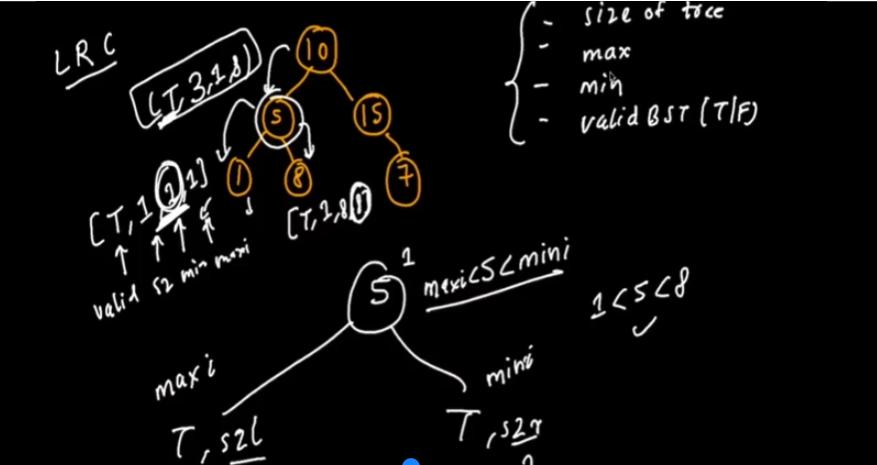
1. Valid BST

Advanced part 3/Trees/BST/questions/Valid Binary Search Tree.txt  




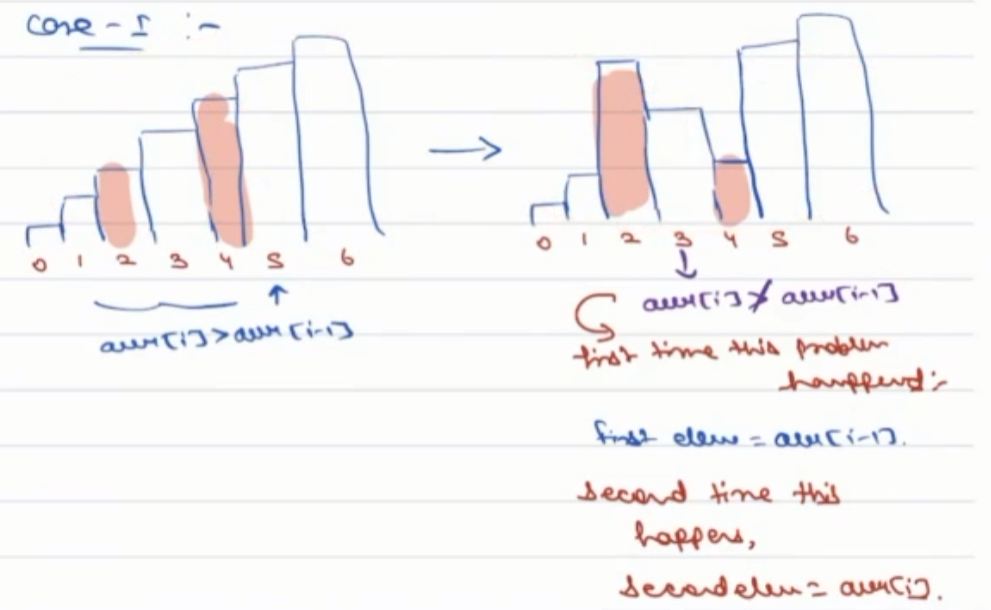
1. Largest BST

Advanced part 3/Trees/BST/answers/Largest BST Subtree.txt

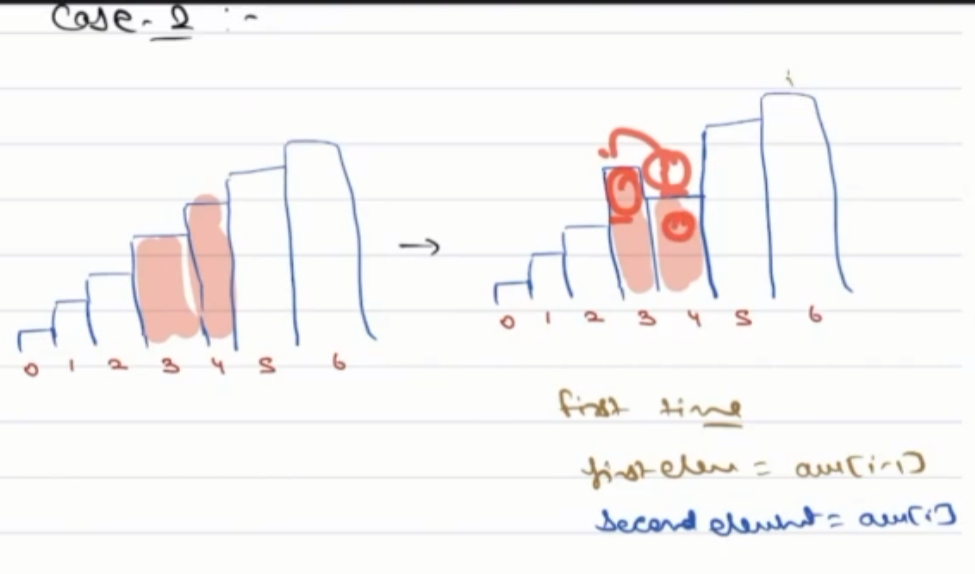


1. Recover BST

Using the knowledge that for a BST the in order traversal should be strictly increasing.

There can be 2 cases:  


In this case we will be able 2 find both the elements using this pattern on the right but its somewhat different in case 2

  
To handle case 2 like scenarios we will update the second element as soon as we update the first element. In future if we again get a scenario where a[i]<a[i-1] we will again update the 2nd element.  
  
  
  
  
4. LCA in BST

The least point of division between the two nodes. Say I am at node curr and the nodes x and y are respectively in left and right wrt my curr then curr is the LCA

**Heap**

1. Connect ropes

Advanced part 3/Trees/Heaps/questions/Connect ropes.txt

Eg: max heap (By default PriorityQueue in java is maintained as minheap)

*import java.util.PriorityQueue;*

*import java.util.Comparator;*

*public class MaxHeapExample {*

*public static void main(String[] args) {*

*// Create a priority queue with custom comparator for max-heap*

*PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Comparator.reverseOrder());*

*// Add elements*

*maxHeap.add(10);*

*maxHeap.add(5);*

*maxHeap.add(20);*

*// Remove elements (largest element comes first)*

*System.out.println(maxHeap.poll()); // Output: 20*

*System.out.println(maxHeap.poll()); // Output: 10*

*System.out.println(maxHeap.poll()); // Output: 5*

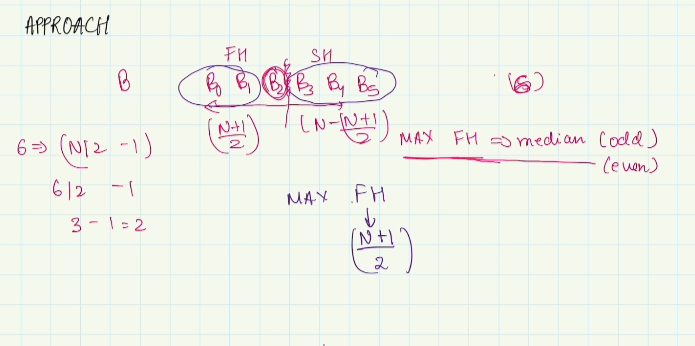
*}*

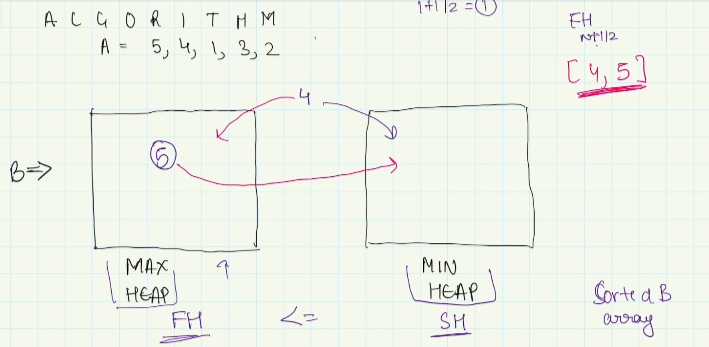
*}*

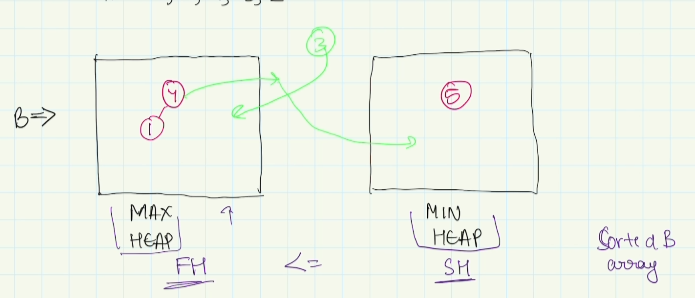
2. B closest points to origin

Advanced part 3/Trees/Heaps/answers/B Closest Points to Origin.txt

3. Running Median

  
Max of First Half of sorted array will be my median.   
So the FH will have (N+1/2) elements.

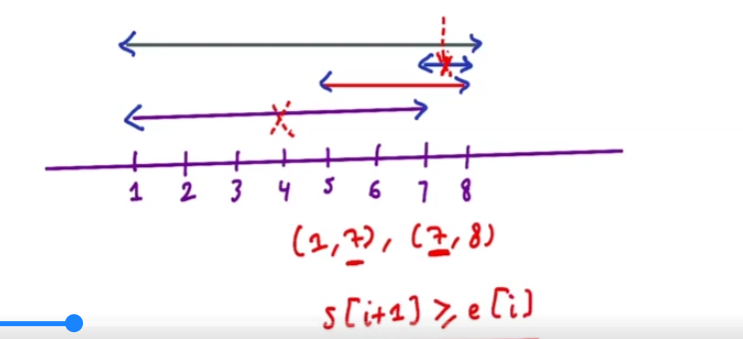




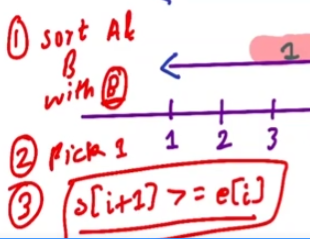
**Greedy**

1. Finish Maximum Jobs

Advanced part 4/Greedy/answers/Finish Maximum Jobs.txt

  
Condn. that has to be maintained, where start of the i+1th job should be greater than end of ith job.

Soln:

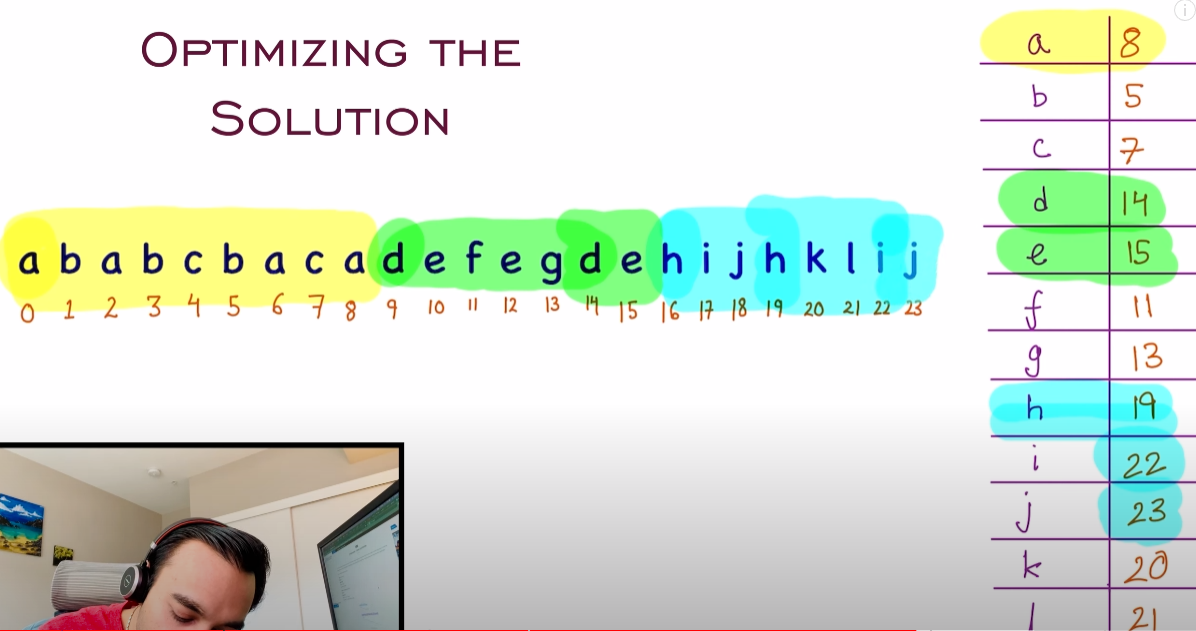
  
  
First sort A & B pair based on earliest end time. Next traverse them and count combinations

2. **Flipkart's Challenge in Effective Inventory Management**Advanced part 4/Greedy/answers/Flipkart's Challenge in Effective Inventory Management.txt

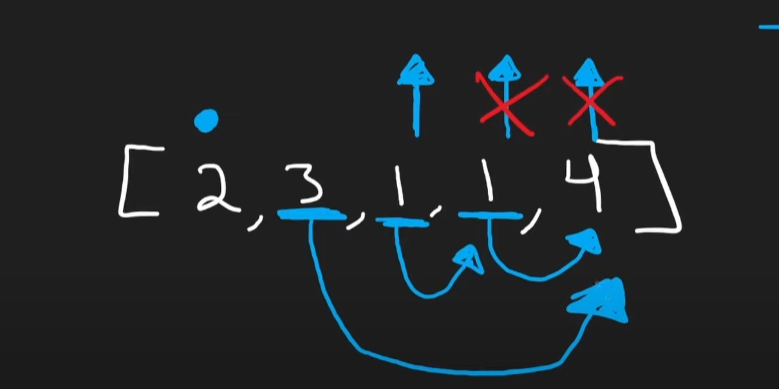
First sort the pairs based on time

Next iterate to add up the profits

3. Partition Label

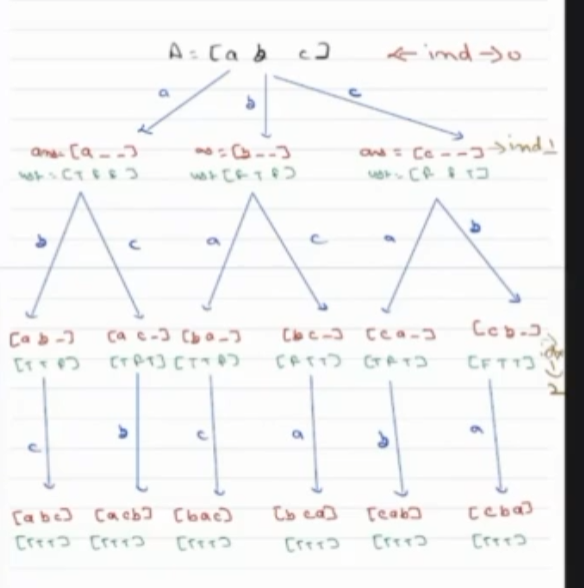
  
get the last index of each character  
  
Start with the starting Index and get its corresponding last index. Now using concept of sliding window search for elements betw the start and last index if any element has a greater end index then update it.

4. Jump Game



Keep updating the goal. Start from right most and check if the goal can be reached from a given pos ie: I, if so update goal posn with i. At last if goal is at 0 then return true

**BackTracking**

1. Permutations  


Advanced part 4/BackTracking/Basic/answers/All Unique Permutations.txt

2. Combination Sum  
Advanced part 4/BackTracking/Advanced/answers/Combination Sum.txt  
  
Note:  
 finalResult.add(new ArrayList<>(current));

Vs  
 finalResult.add((current));  
  
In the second way of implementation we would be storing a reference of the obje in finalResult which might get updated later