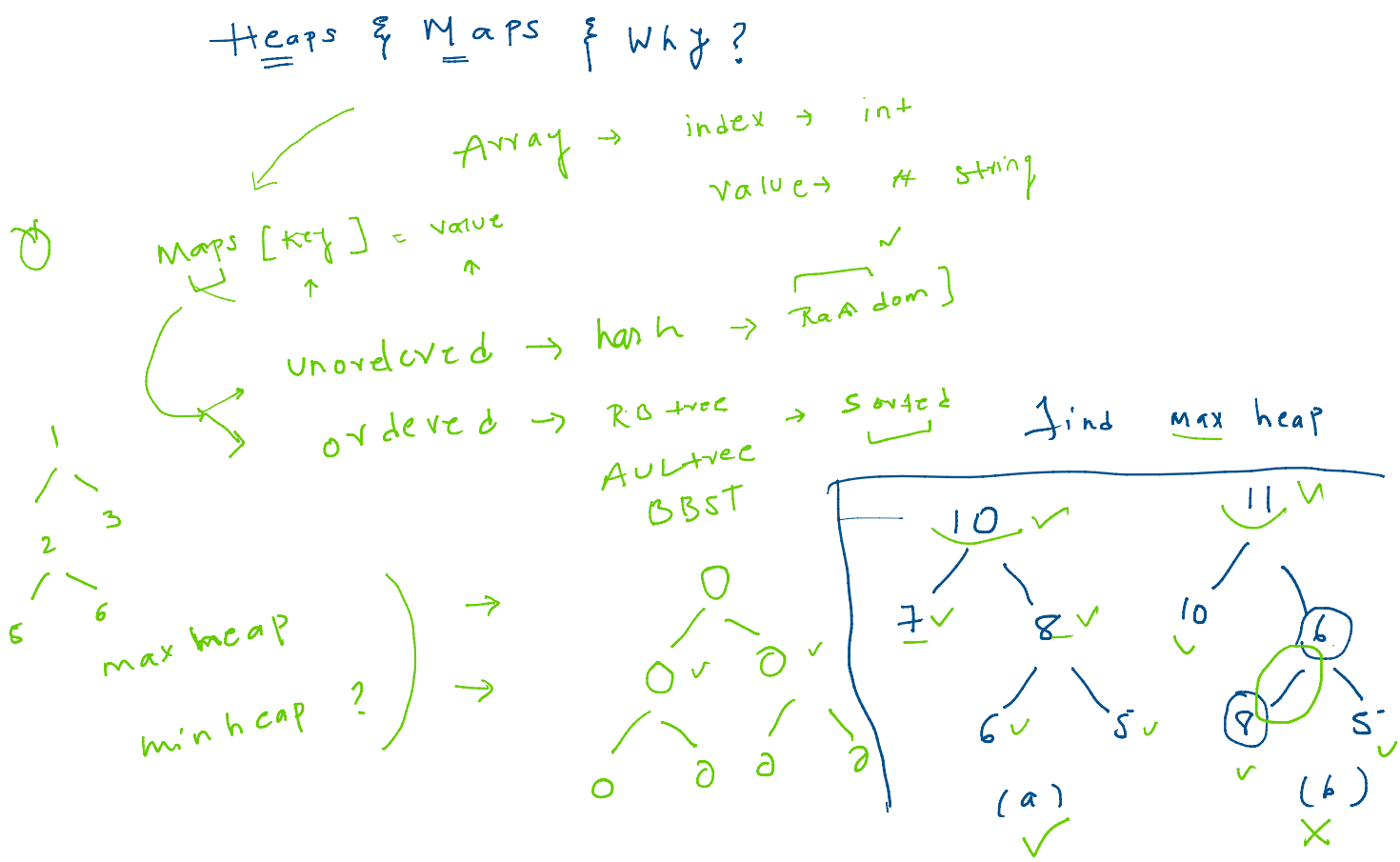
Heap 22

22 June 2022 18:40

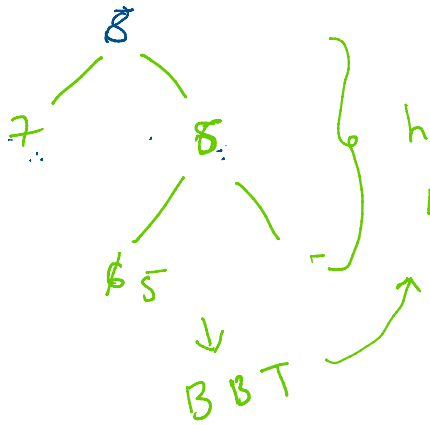


	Insertion (x)	Deletion (x)	search (x)
HEAP →	O(log(n))	log(n)	O(n * Logn)
MAP (Hash) unordered	O(1)	O(1)	O(1)
MAP (BBST) AVL	Olog(n)	log(n)	Log(n)

AVL Tree

{H.W 99% O(1) x O(n) 1% x → P.N

Delete



Search (x)

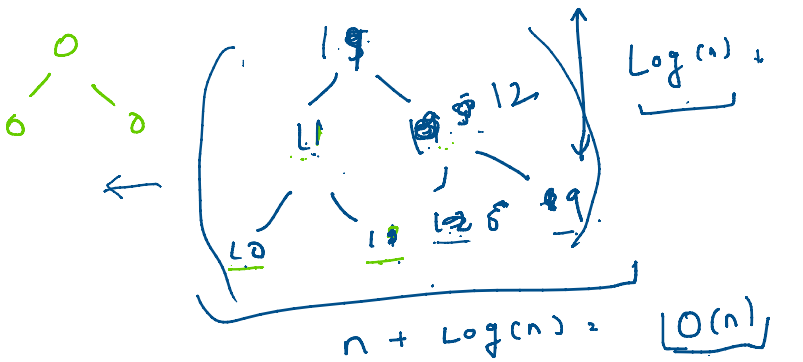
```
function (heap HP ) {
x = hp.top()
   hp.pop()
      if (x == search
}
```

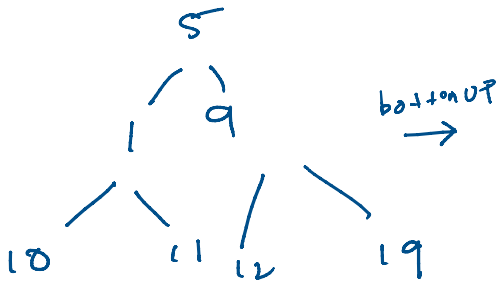
O(n)

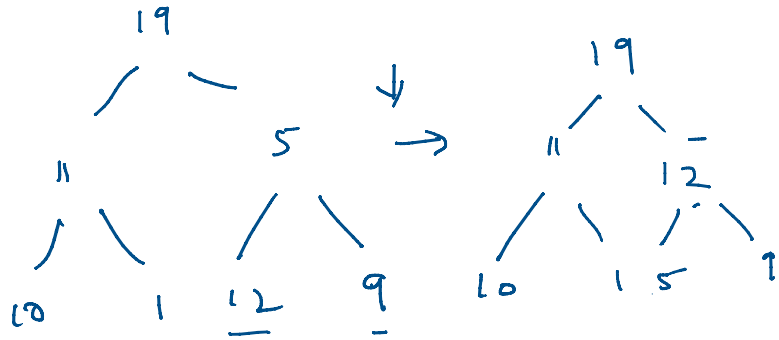
Insert 'n' elements in heap?

→ n * Log n

→







max-heap

(i)

→ hash

func [key] → store

K1

func [K1] → T O(1)

hash map unordered

chaining coillion

map [key] = value → O(1)

cat → 101 → O(1)

func [K1] → T O(1)
F

sai → ~~VSa~~ 101
sai → 101

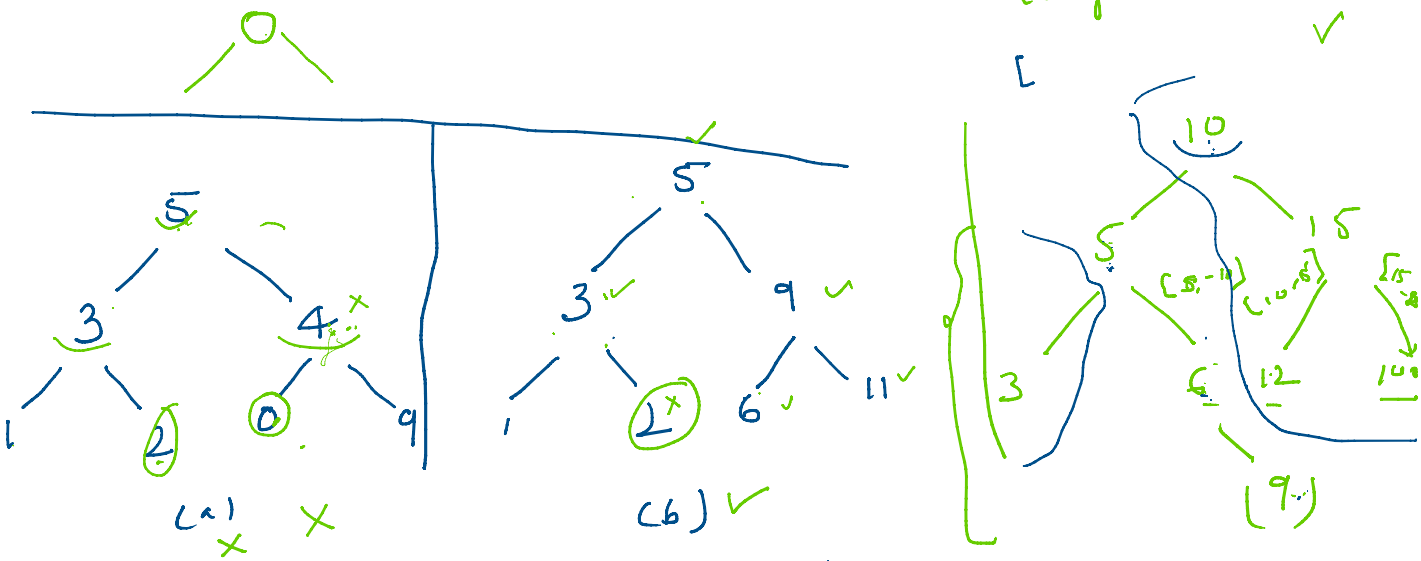
-O(1)

BBST

map

-1

log(n) (9)



(a) X X

(6) ✓

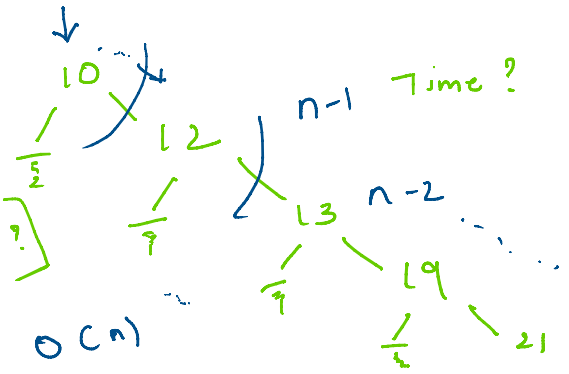
→ AVL Tree

→

BST ✓

is Balanced ?

X



n-1 Time ?

O(n)

→ Log(n) → height tree

Session 9

Topics: Application of Heaps & Maps

Background required for today's topics

- Basic Working of Heaps : Building of a Heap, Insertion, Deletion, Top
- How a map works [BBST], it's internals - insertion, deletion, search
- Time complexity associated with each operation of a heap & map

Question 1

Given a non-empty array of integers, return the *k* most frequent elements.

→ T.C] for unordered map ← freq] freq -> O(n)
step ①

max heap (val, freq)

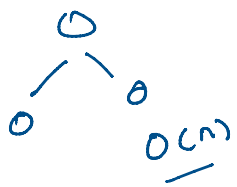
step ②

k times

n-k ?

→ O(n + n * k.Logn)

O(n + k.Logn) ✓✓



for ordered map → freq → nLogn

→ traverse sorted order

→ O(n)

→

+ O(n) → O(n.Logn)

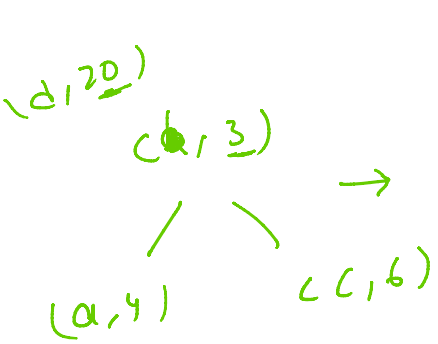
→

$n(\log n) + O(n) \rightarrow O(n \cdot \log n)$

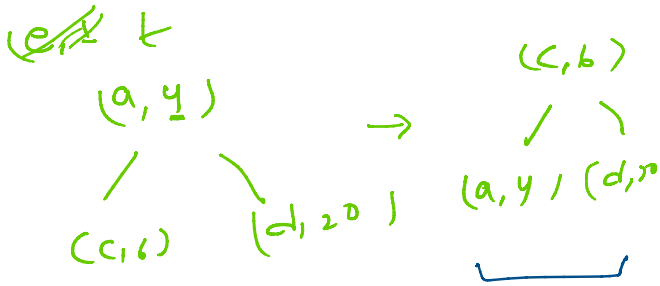
$K = 3$

a - 4
b - 3
c - 6
d - 20
e - 1
f - 10

$n - K$



→ $\log(K)$



→ $(O(n) + K + (n-K) * \log K)$

→ $n + K + n \cdot \log K - K \log K$

→ $n \log K$

A① → map + heap → $K \cdot \log n$

A② → ordered map →

A③ → hash + min heap $(n - K)$ $(1, 10)$

A④ → hash + min heap → tree size (K) → $n \cdot \log K$

Example

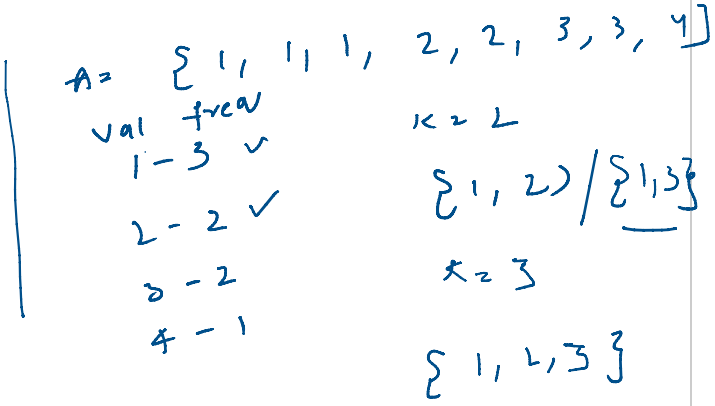
arr[] = {7, 10, 11, 5, 2, 5, 5, 7, 11, 8, 9},

k = 4

Output: 5 11 7 10

Input: nums = [1], k = 1

Output: [1]



Solution (Naive)

Count frequencies of each element using array.

Sort the array in decreasing order of frequencies.

Return the first k elements of the sorted array.

Solution - Using Max-Heap (Priority Queue)

`Use priority queue to implement a Max-Heap to store HashMap of the element-frequency pair.`

Priority queues are a type of container adapters, specifically designed such that the first element of the queue is the greatest of all elements in the queue and elements are in non increasing order.

Solution Cont.

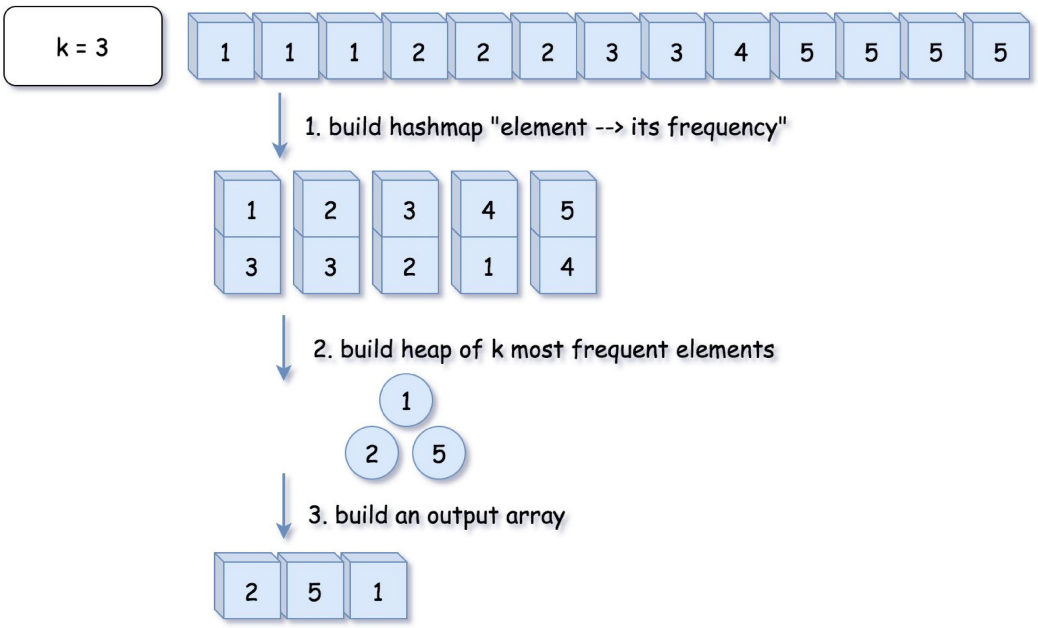
1. Create a Hashmap *hm*, to store key-value pair, i.e. element-frequency pair.
2. Traverse the array from start to end.
3. For every element in the array update *hm[array[i]]++*
4. Store the element-frequency pair in a Priority Queue and create the Priority Queue, this takes O(n) time.
5. Run a loop k times, and in each iteration remove the top of the priority queue and print the element.

Complexity Analysis

- **Time Complexity:** O(k log d + n), where **d** is the count of distinct elements in the array.
  To remove the top of priority queue O(log d) time is required, so if k elements are removed then O(k log d) time is required and to traverse the n elements O(n) time is required.
- **Space Complexity:** O(d), where **d** is the count of distinct elements in the array.
  To store the elements in HashMap O(d) space complexity is needed.

Alternate Solution

Complexity: O(n + k + (n-k)logk)



More Solutions ….

- Quickselect (HW) [O(n) in an average case]

Code

https://www.ideone.com/fork/jEtsjr

Question 2

Problem Statement

Find the kth largest element in an unsorted array.
Note that it is the kth largest element in the sorted order, not the kth distinct element.

n-k

k

Example

```
Input: [3,2,1,5,6,4] and k = 2

 Output: 5

Input: [3,2,3,1,2,4,5,5,6] and k = 4

 Output: 4
```

Solution

We can find k'th largest element in time complexity better than O(N Log N). A simple optimization is to create a Max Heap of the given n elements and call extractMax() k times.

Solution

```
BUILD-HEAP(A)

    heapsize := size(A);

    for i := floor(heapsize/2) downto 1

        do HEAPIFY(A, i);

    end for

 END
```

Can we do the same by using a Min Heap as well?

Complexity Analysis

Time complexity of this solution is O(n + kLogn). O(n) for building the heap and O(klogn) for calling the extractMax() k times.

Space Complexity : O(n)

Alternate Solutions to Explore

- Quickselect [O(n^2) in worst case, O(n) on an average]

- Randomized Quickselect [O(n) on average, performs the best than any other algorithm in the average case]

- Quickselect with partitioning based on Median of Medians [Worst case time complexity is O(n) but with high constants]

Question 3

Problem Statement

Given a char array representing tasks CPU need to do. It contains capital letters A to Z where different letters represent different tasks. Tasks could be done without original order. Each task could be done in one interval. For each interval, CPU could finish one task or just be idle.

However, there is a non-negative cooling interval n that means between two same tasks, there must be at least n intervals that CPU are doing different tasks or just be idle.

You need to return the least number of intervals the CPU will take to finish all the given tasks.

Example

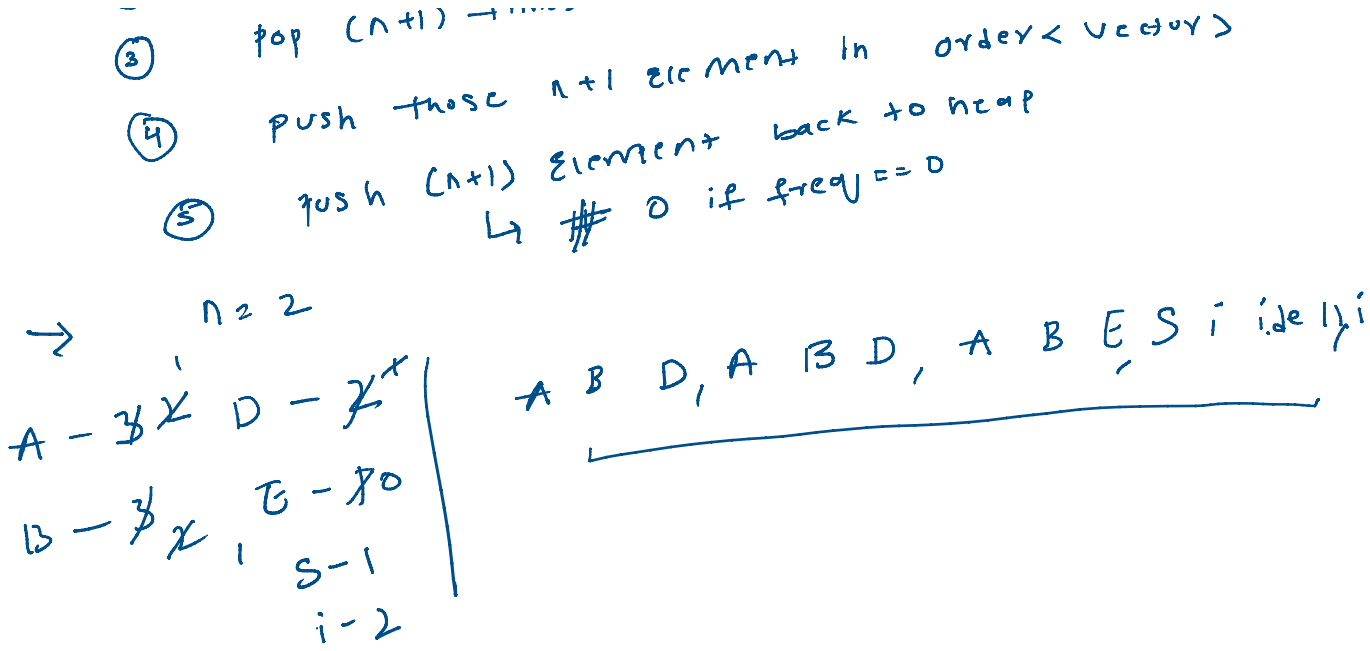
Input: tasks = ["A","A","A","B","B","B"], n = 2

Output: 8

Explanation: A -> B -> idle -> A -> B -> idle -> A -> B.

Constraints

- The number of tasks is in the range [1, 10000].
- The integer n is in the range [0, 100].



Solution 1

- Use sorting / hashmap counts of lowercase latin letters
- Among those letters not in cooldown, we do the task that has max count
- If all of them in cooldown, means we are in idle mode.

Solution cont.

We can use a Max-Heap (*Priority-queue*) to pick the order in which the tasks need to be executed. But we need to ensure that the heapification occurs only after the intervals of cooling time, n.

1. Put those elements from map into the priority queue which have non-zero number of instances.
2. Start picking up the largest task from the priority queue for current execution. (Again, at every instant, we update the current time as well.) We pop this element from the queue.
3. We also decrement its pending number of instances and if any more instances of the current task are pending, we store them(count) in a temporary temp list, to be added later on back into the queue.
4. We keep on doing so, till a cycle of cooling time has been finished. After every such cycle, we add the generated temp list back to the queue for considering the most critical task again.
5. We keep on doing so till the queue(and temp) becomes totally empty. At this instant, the

Complexity Analysis

Time complexity : O(n). Number of iterations will be equal to resultant time time.

Space complexity : O(1). Queue and temp size will not exceed O(26).

Code

https://www.ideone.com/fork/91XoLd

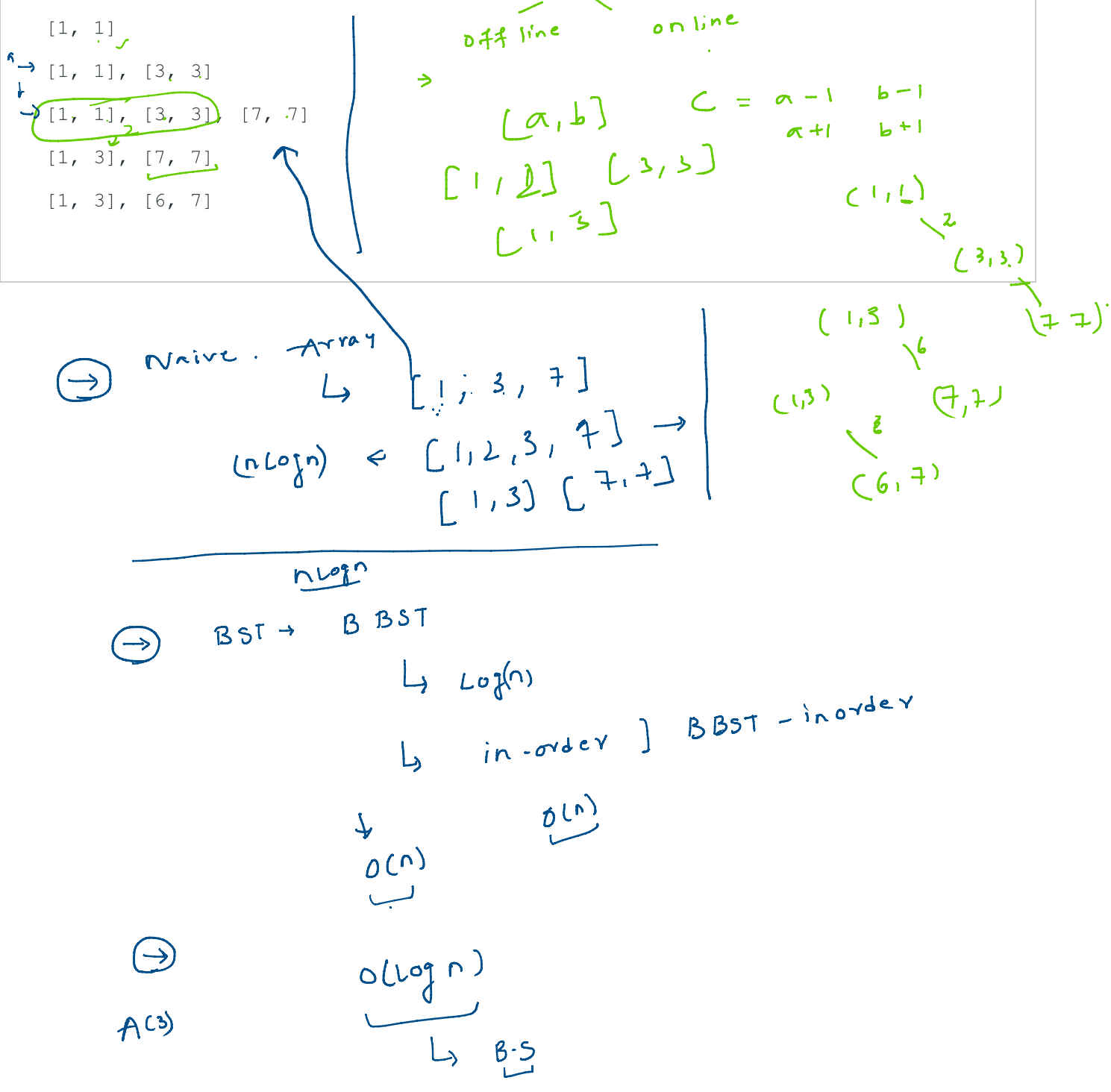
Question 4

Problem Statement

Given a data stream input of non-negative integers a1, a2, ..., an, ..., summarize the numbers seen so far as a list of disjoint intervals.

For example, suppose the integers from the data stream are 1, 3, 7, 2, 6, ..., then the summary will be:

```
[1, 1]
[1, 1], [3, 3]
[1, 1], [3, 3], [7, 7]
[1, 3], [7, 7]
[1, 3], [6, 7]
```



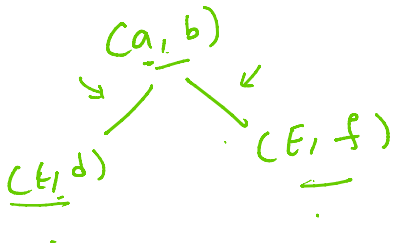
[x, y] d [a, b]

y = d+1 — y − 1
        \ N − 2

a = d−1 — y − 3
        \ N − 4

d

- 2, 4 → N.N
- 1, 3 → merge
- 1, 4 → ~~cap~~ update y
- 3, 2 → update a



O → pair<int, int>

→

BBST

Data Streams ……. ? Fav. Goto of Big N

- Many questions in interview involve ***data streams***

- What does this mean?

- In simple terms , it means your solution/code should work ***online*** (as the data comes) instead of ***offline***.

- For eg: How is an algorithm like this for data streams: storing in vector and sorting?

Naive Solutions

- Store in vector, sort, find ranges.

- Sorting take NlogN everytime.

- **Solution**: Use ordered map

- Now, find ranges.

- Complexity: O(logN) for insertion in ordered set, O(N) for iteration through ordered set and allocating ranges.

- Code: https://ideone.com/dXfcSh

Solution cont ….

- Note that this will be the best case solution esp. when there are lots of ranges for data such as 1,3,5,7,..... etc etc, as the worst case time complexity is anyways O(N).

- However, one follow up can be: ***What if there are lots of merges and the number of disjoint intervals are small compared to the data stream's size?***

Hint

- We certainly can do better than O(N) in getting intervals as number of intervals << N.

- Thus, the idea we get is to directly store the intervals instead of single numbers and merging later.

- But then, how do we maintain this interval when a new number is added?

Solution

- We can use binary search (upper bound) on our ordered map to decide the suitable interval which will be added, modified or merged.

- Refer to the code.

Complexity Analysis

Worst Case: O(logN) per number insertion, O(N) for every query [all separate ranges]

Many Merges, less no of disjoint intervals: O(logN) per number insertion [more constant than previous solution], O(m) where m << N, m is no. of disjoint intervals.

Code

https://ideone.com/mJC7Gp

Question 5

Asked in Google intw in last 1 year

Problem Statement

Alice has a hand of cards, given as an array of integers.

Now she wants to rearrange the cards into groups so that each group is size W, and consists of W consecutive cards.

Return true if and only if she can.

Example

Example 1:

```
Input: hand = [1,2,3,6,2,3,4,7,8], W = 3

Output: true

 Explanation: Alice's hand can be rearranged as [1,2,3],[2,3,4],[6,7,8].
```

Example 2:

```
Input: hand = [1,2,3,4,5], W = 4

Output: false

 Explanation: Alice's hand can't be rearranged into groups of 4.
```

Solution

- Store the count of the numbers in an ordered map.
- Extract the smallest number that has count >= 1
- Check whether next consecutive W-1 elements are present or not and decrease the count by 1 accordingly.
- Repeat the procedure
- If all operations happen successfully and map is empty, return true
- Else return false

Complexity

O(NlogN) -> Can you answer how?

Question 6 [Another Data Stream Question]

Asked in Google intw in last 1 year

Problem Statement

Median is the middle value in an ordered integer list. If the size of the list is even, there is no middle value. So the median is the mean of the two middle value.

For example,
`[2,3,4]`, the median is `3`

`[2,3]`, the median is `(2 + 3) / 2 = 2.5`

Design a data structure that supports the following two operations:

- void addNum(int num) - Add a integer number from the data stream to the data structure.
- double findMedian() - Return the median of all elements so far.

Example

Example:

```
addNum(1)

addNum(2)

findMedian() -> 1.5

addNum(3)

 findMedian() -> 2
```

Solution

- Multiple solutions are possible for this problem.
- We will discuss the one with two heaps.
- We have to keep a max heap and min heap.
- Max Heap for first half of elements
- Min Heap for the other half
- Why ? How will we find median with the help of this?

Complexity

Addition of number : O(logN)

Median Query: O(1)