Code archive

Sunday, 26 February 2023 7:39 AM

Practice programs:

- https://leetcode.com/problem-list/top-interview-questions/?page=1
- https://github.com/xizhengszhang/Leetcode_company_frequency/blob/master/100%20Problems%20-%20LeetCode.pdf
- https://takeuforward.org/data-structure/trapping-rainwater/
    - Maintain Prefix, suffix arrays values
    - Calculate leftMax, rightMax
- https://takeuforward.org/data-structure/remove-duplicates-in-place-from-sorted-arr
    - Use 2 pointer approach
    - Move 1 pointer slow
    - Fast pointer will identify the different elements
    - And using slow pointer it will be positioned next to different element
- https://takeuforward.org/data-structure/fractional-knapsack-problem-greedy-approa
- https://takeuforward.org/data-structure/count-maximum-consecutive-ones-in-the-ar
- https://www.geeksforgeeks.org/the-stock-span-problem/
- Next greater element(nge) in circular array ->
    - https://takeuforward.org/data-structure/next-greater-element-using-stack/
    - https://www.geeksforgeeks.org/find-the-next-greater-element-in-a-circular-arr
    - Youtube -> Next Greater Element | Two Variants | Leetcode
- Prefix/suffix sum arrays -> https://www.geeksforgeeks.org/suffix-sum-array/
- Maximum consecutive 1's -> https://takeuforward.org/data-structure/count-maximur array/
- Merge Sorted Array
    - https://zyrastory.com/en/coding-en/leetcode-en/leetcode-88-merge-sorted-arr explanation-en/
    - In place comparison and filling algorithm
    - Start from the last instead of beginning to fill the empty spaces first
    - Youtube -> Merge Sorted Array - Leetcode 88 - Python
- Celebrity find problem
    - https://leetcode.ca/2016-09-02-277-Find-the-Celebrity/
    - https://www.prepbytes.com/blog/stacks/the-celebrity-problem/
    - https://www.codingninjas.com/codestudio/library/celebrity-problem
    - Youtube -> Coding Interview Question: The Celebrity Problem [Logicmojo.com]
        - Approach 1:

%20Most%20Asked%

ay/

ch/
ray/

ay/?ref=rp

m-consecutive-ones-in-the-

ray-solution-and-

      - Using **stack data structure**
      - Insert all the members into the stack..order doesn't matter
      - Pick the top 2 elements J,I and set it as arr[I][J]
      - If the I knows J, then i is not a celebrity, so insert the j only back
      - If the I does not know J, then J is not a celebrity, so don't inert the J
      - If the only element left finally in stack, then we can assume him as c
      - Do row wise and column wise search to check whether everyone kn know anyone or not
    - Approach2:
      - 2 pointer approach
      - Start I=0, j=n-1
      - Move the pointers i++, j-- till the condition mees i<j
      - And pick the element as celebrity element
      - Do row wise and column wise search to check whether everyone kn know anyone or not
- **Bucket sort**

Values as index	0	1	2	3
count				

I as count	1	2	3
List of values	[1]	[2,3]	[4]

- K frequent elements in array
  - https://www.geeksforgeeks.org/find-k-numbers-occurrences-given-array/
  - Solve using hashmap, sort the array, pick the top k elements using iteration
  - Hashmap is required, Use max-heap(priority queue) instead of complete sort, po
  - Hashmap is required, use **bucket sort** to use indexes of the bucket array as frequ values which fits into that index..and sort the list internally within the index.
  - Bucket sort video -> Top K Frequent Elements - Bucket Sort - Leetcode 347 - Pyth

- Check bst or not -> https://www.geeksforgeeks.org/a-program-to-check-if-a-binary-tr
  - Recursive approach - O(N2)
    - Using min/max concept by passing (min, data-1) to left and (data+1, max)
    - Up-bottom approach
  - Inorder traversal - O(N)
    - By setting previous node while travesing from one node to another.
    - Bottom-up approach
- Implement queue using stack
  - https://takeuforward.org/data-structure/implement-queue-using-stack/
  - Approach1:
    - Maintain Stack1 and stack2

back, insert I only
celebrity
ows him and he doesn’t

ows him and he doesn’t

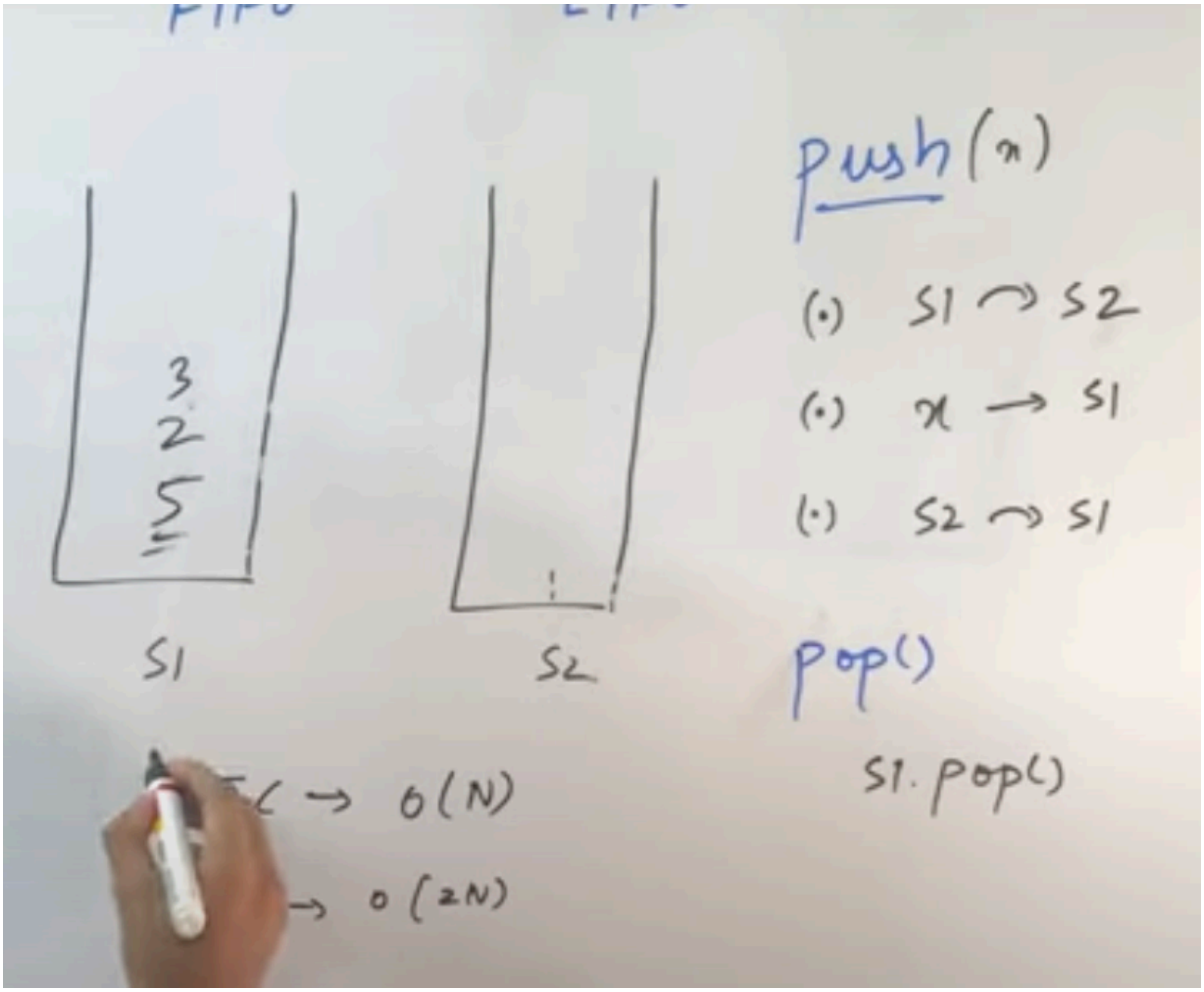
op the first k elements
uency and store the list of

hon

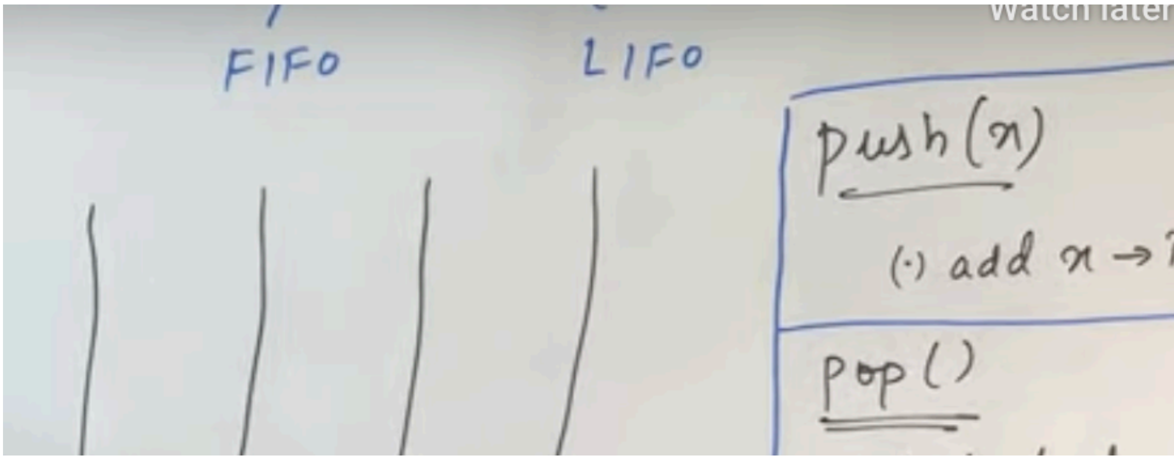
ee-is-bst-or-not/

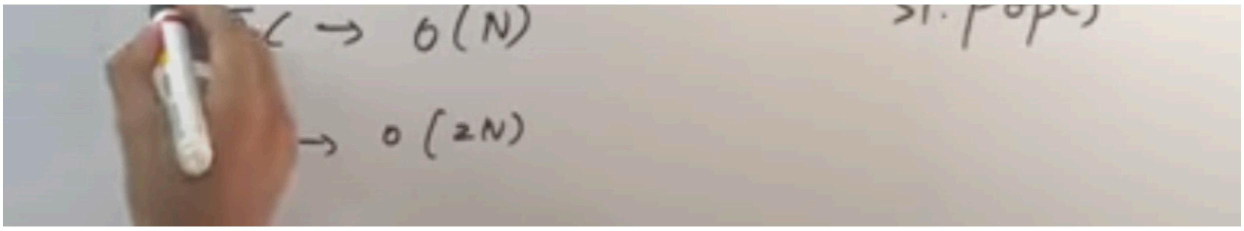
to right recusively

- Costly push() operation
- Efficient pop() operation
- Push
  - □ Before pushing element into stack1, move the elements from stack1
  - □ Once the element is inserted in stack1, shift the elements back from

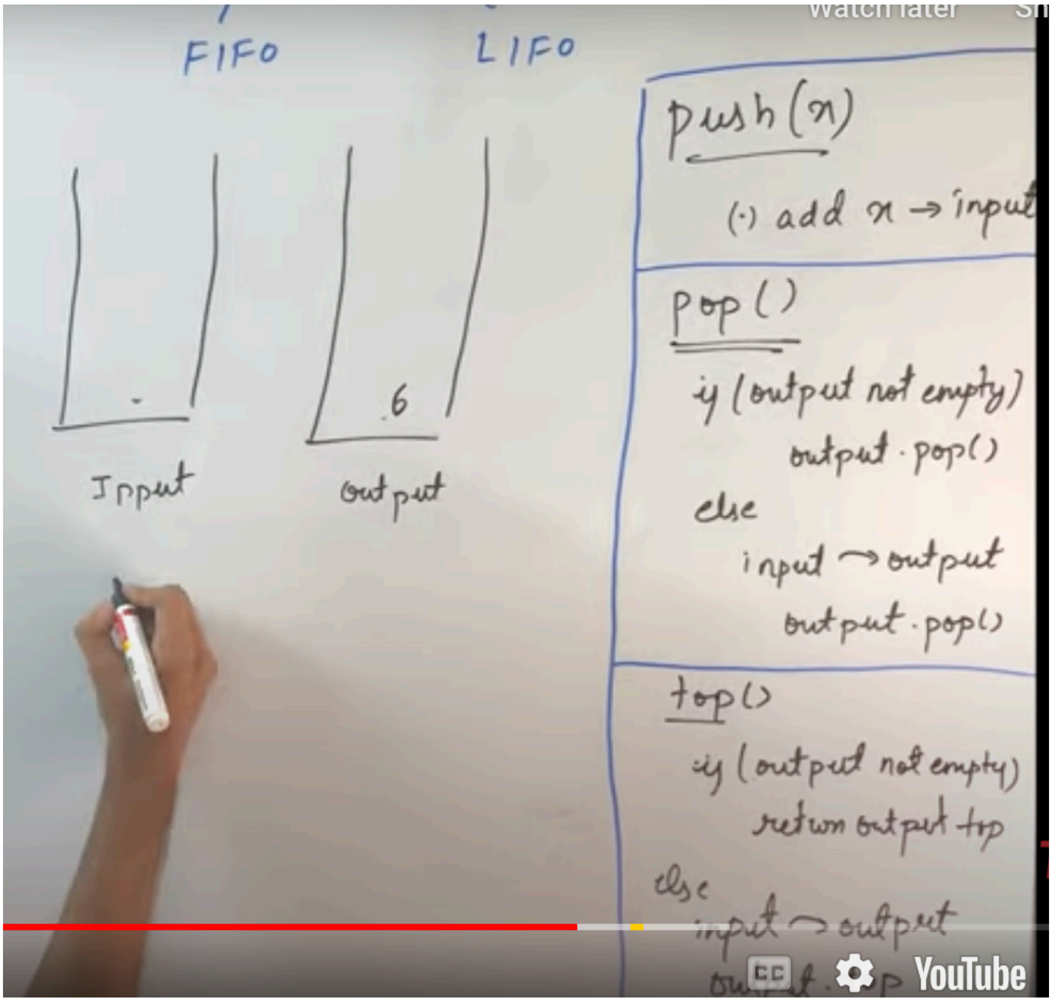
- 

  - □ Approach2:
  - □ Maintain input and output stacks
  - □ Motive is to optimize the push() and pop() operations, because in pr costly operation



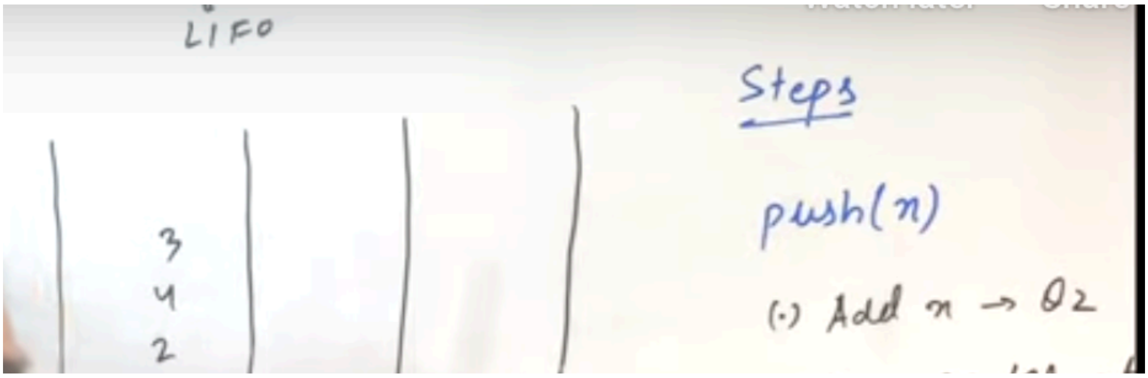


- □ Approach2:
- □ Maintain input and output stacks
- □ Motive is to optimize the push() and pop() operations, because in previous approach, it is a costly operation

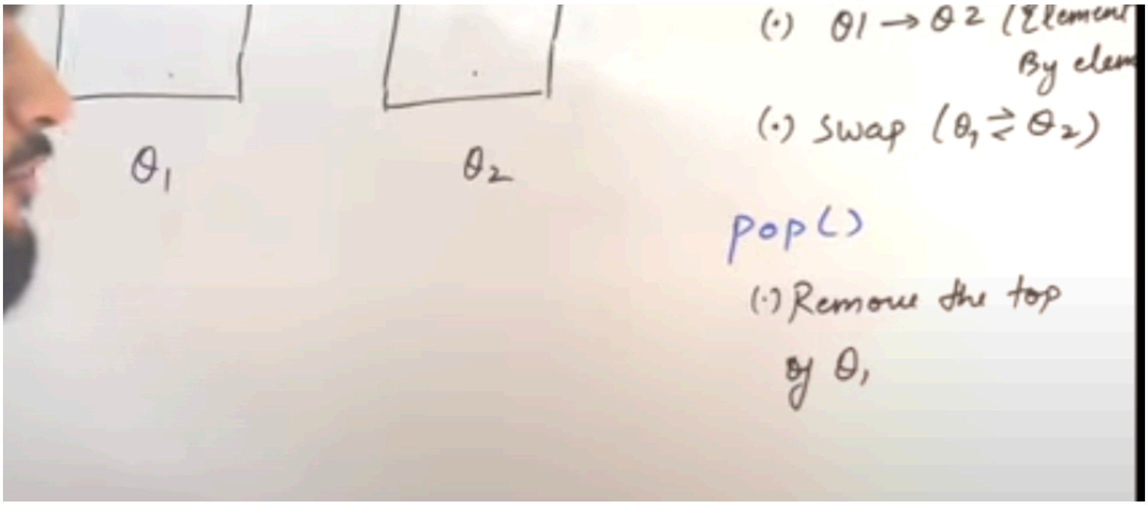


```
push(x)
  (·) add x → input
pop()
  if (output not empty)
      output.pop()
  else
      input → output
      output.pop()
top()
  if (output not empty)
      return output.top
  else
      input → output
```

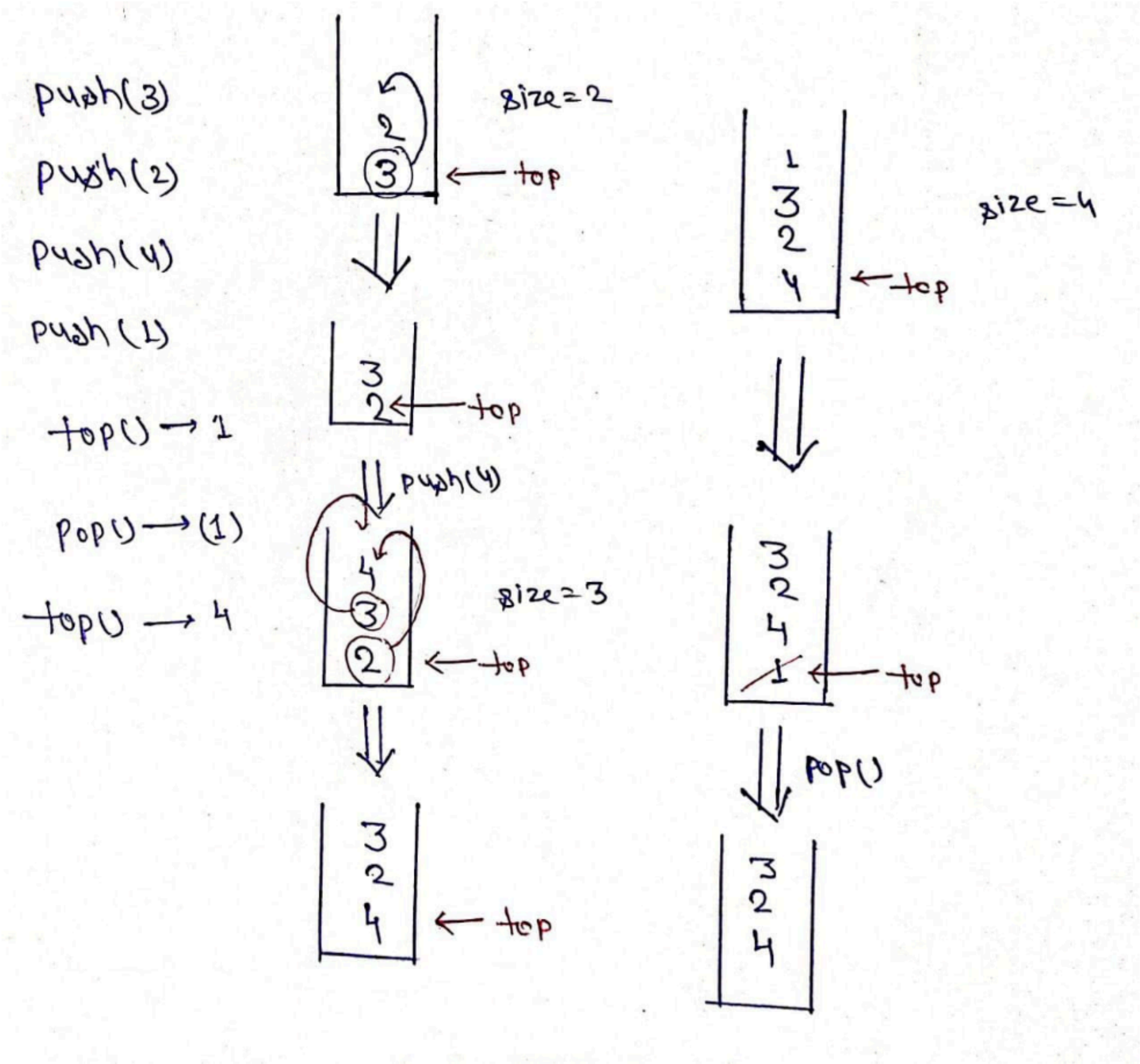
- Stack using queue
  - https://takeuforward.org/data-structure/implement-stack-using-single-queue/



```
Steps
push(x)
  (·) Add x → Q2
```

- ○ 

Repeat **step3** at every insertion of the element.

- 

Find maximum of minimums for every window size:
https://www.geeksforgeeks.org/find-the-maximum-of-minimums-for-every-

https://www.geeksforgeeks.org/find-the-maximum-of-minimums-for-every-window-size-in-a-given-array/

LFU cache

Since it is frequency based algorithm, frequency counter is required for each key

So 3 hashMaps are required

- Map<Key, Value> vals;
- Map<Key, Counter> counts;
- Map<Counter, LinkedHashSet<Integer>> counter and itemsList
- Totally 3 maps are required to manage LFU cache in O(1) complexity.
- https://medium.com/algorithm-and-datastructure/lfu-cache-in-o-1-in-java-4bac0892bdb3
- https://www.enjoyalgorithms.com/blog/least-frequently-used-cache

Symmetric Binary Tree

https://takeuforward.org/data-structure/check-for-symmetrical-binary-tree/

Compare

https://www.geeksforgeeks.org/compare-two-version-numbers/

Construct/build tree from inorder and preorder traversal

- https://www.geeksforgeeks.org/construct-tree-from-given-inorder-and-preorder-traversal/
- https://takeuforward.org/data-structure/construct-a-binary-tree-from-inorder-and-preorder-traversal/
- Use hashmap to store inorder data and its indexes
- Move index in preorder traversal
    - Divide into left and right subarrays based on the array index in inorder array
    - If leftPtr>rightPtr then
        - Return null
    - If leftPtr == rightPtr then
        - Return node.

Fix Children Sum property:

https://www.techiedelight.com/fix-children-sum-property-binary-tree/

https://takeuforward.org/data-structure/check-for-children-sum-property-in-a-binary-tree/

Inorder prdecessor successor

https://www.geeksforgeeks.org/inorder-predecessor-successor-given-key-bst/

Build binary tree from preorder traversal

- https://www.techiedelight.com/build-binary-search-tree-from-preorder-sequence/
- https://www.geeksforgeeks.org/construct-bst-from-given-preorder-traversa/
    - Approach:
    - Construct elements on the go and attach it, so that time complexity of

        - Construct elements on the go and attach it, so that time complexity of O(N) can be achieved
        - Using min/max concept of validating BST, we can

Kth largest element in data streams

https://www.geeksforgeeks.org/kth-largest-element-in-a-stream/

Serialize and deserialize a tree:

https://takeuforward.org/data-structure/serialize-and-deserialize-a-binary-tree/

    - Approach:
    - Level order traversal
    - Using queue to do serialize and deserialize
    - Store as {1,2,3,4,null, null, 5,7,null}

- Connect nodes at same level:
    - https://workat.tech/problem-solving/approach/pnrpien/populating-next-right-pointers-in-each-node
    - https://www.geeksforgeeks.org/connect-nodes-at-same-level/
        - BFS traversal
        - Using queue data structure
        - By pushing nodes left and rigt nodes into queue together
        - Approach 2 - (**PreOrder traversal**) without extra auxiliary space
        - Traverse in pre order by root-> left -> right…-> nextRight
        - Again…traverse down… root -> left ->
- https://www.geeksforgeeks.org/connect-nodes-at-same-level-with-o1-extra-space/
    - Same logic of complete binary tree but since there might be missing of nodes inbetween, the using next of current level..we should keep on checking left and right child nodes to assign right.
    - 2 while loops required
    - Outer loop…will set the take the node pointer to next level extreme left/right
    - Inner loop will keep on moving to nextRight
        - Q.left
        - Q.right
- https://www.geeksforgeeks.org/largest-bst-binary-tree-set-2/
    - Bottom-up approach, eliminate non bst sub trees using isBST flag
    - https://www.geeksforgeeks.org/find-the-largest-subtree-in-a-tree-that-is-also-a-bst/

Median from Data streams:

```
importjava.util.PriorityQueue;
importjava.util.Random;
importjava.util.stream.Collectors;
importjava.util.stream.IntStream;
```

```
importjava.util.stream.IntStream;

publicclassRunningMedian{
publicstaticvoidmain(Stringargs[]){
RunningMedianUtilityrunningMedianUtility=newRunningMedianUtility(
);
IntStream.generate(()->
newRandom().nextInt(50)).limit(newRandom().nextInt(50)).forEach(run
ningMedianUtility::add);
runningMedianUtility.display();
System.out.println(runningMedianUtility.getMedian());
}
}

classRunningMedianUtility{
privatefinalPriorityQueue<Integer>maxHeap;
privatefinalPriorityQueue<Integer>minHeap;

publicRunningMedianUtility(){
this.maxHeap=newPriorityQueue<>((o1,o2)->{
return-1*o1.compareTo(o2);
});
this.minHeap=newPriorityQueue<>();
}

publicvoidadd(Integervalue){
if(maxHeap.isEmpty()||maxHeap.peek()>value){
maxHeap.add(value);
}else{
minHeap.add(value);
}
if(maxHeap.size()>minHeap.size()+1){
minHeap.add(maxHeap.poll());
}elseif(minHeap.size()>maxHeap.size()+1){
maxHeap.add(minHeap.poll());
}
}
```