Lecture 21

Some Questions I Deleted from the Final Exam

1. I need to store n numbers in a data structure and since n is very large so I don't have much extra space for these numbers. I need to continuously search and delete $k \ll n$ numbers from this data structure. Which data structure should I use for a good time complexity?
   a. A sorted ArrayList
   b. A sorted LinkedList
   c. A Binary Heap
   d. An AVL Tree

	Search	Deletion
Sorted ArrayList	$O(\lg n)$	$O(n)$
Sorted LinkedList	$O(n)$	$O(1)$
Binary Heap	$O(n)$	$O(\lg n)$
AVL Tree	$O(\lg n)$	$O(\lg n)$

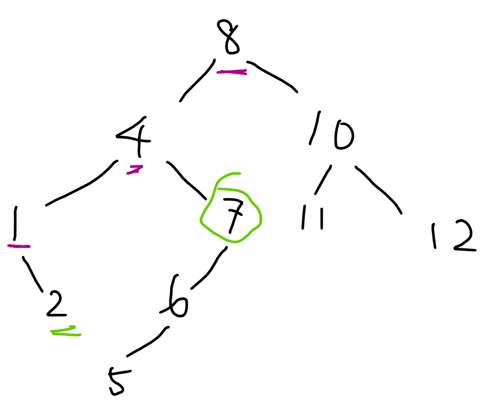
2. In an AVL tree with n nodes, among all paths from the root to leaves, what is the maximum possible ratio between the lengths of the longest path and the shortest path?
   a. 3
   b. 2
   c. 1.5
   d. $\lg n$

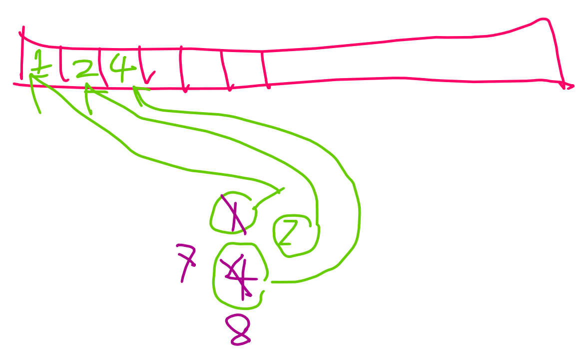
   - If the root has height h, then the longest root-to-leaf path has length h. So, let's focus on the length of a shortest possible path:
     If root has height h, then one of its child has height $h-1$ and the other child has height at least $h-2$.
   - The child of height $h-2$ has a child of height $h-3$ and the other child has height at least $h-4$.
   - The child of height $h-4$ has a child of height $h-5$ and the other child has height at least $h-6$.

   …

   In the worst case, there can be a branch in the AVL tree such that, for each level down on this branch, its height reduced by 2. Thus, the shortest possible root-to-leaf path has length $h/2$.

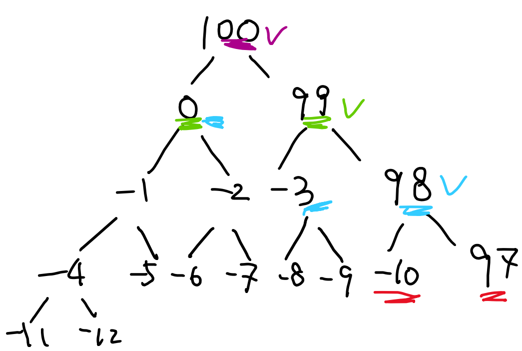
3. Present an implementation of **inorder_tree_walk**(T) for binary search tree T without using recursions.

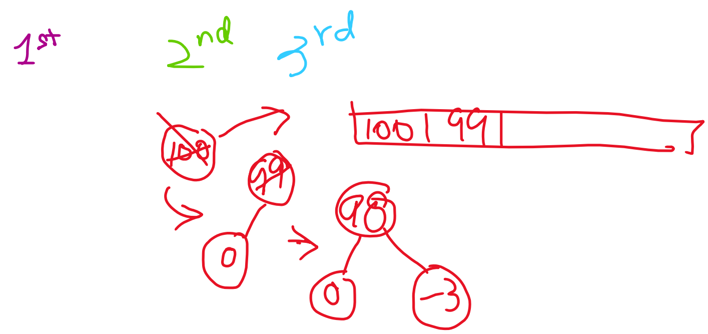




```
inorder_tree_walk(T)
output = [ ]
stack = Stack()
x = T.root
while True:
        if x is not None:
                stack.push(x)
                x = x.left
        elif stack is not empty:
                x = stack.pop()
                output.append(x)
                x = x.right
        else: break
return output
```

4. In a binary max heap BH with n unique numbers, how to return the largest $k \ll n$ numbers? Can you solve this problem within a running time that's independent from n?





- If we call pop max k times, then the running time will be $O(k \cdot \lg n)$
- We have the following observation: when the i^{th} largest number is found, there are $i+1$ candidates for the next largest number. With this observation, we have the following $O(k \lg k)$ algorithm:

```
k − largest(BH)
Let bh be an empty new binary heap      1
output = []      1
MAX = BH.max ()      1
bh.add(MAX)      1
for x in range(k)      k
        max = bh.pop_max ()      log k
        output.append(max)      1
        bh.add(max's left child in BH)   # this can be done in constant time   1
                                          with the help of HashTable(number, index_in_BH)
        bh.add(max's right child in BH)      1
return output
```