HW 1

28 May 2025 12:29

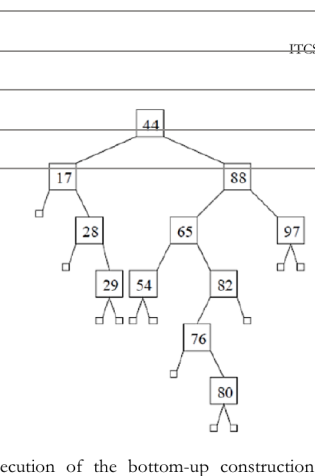
HW231+-+Summer+...

Homework 1: Time Complexity, Basic Data Structures, Binary Search Trees, and Hashing

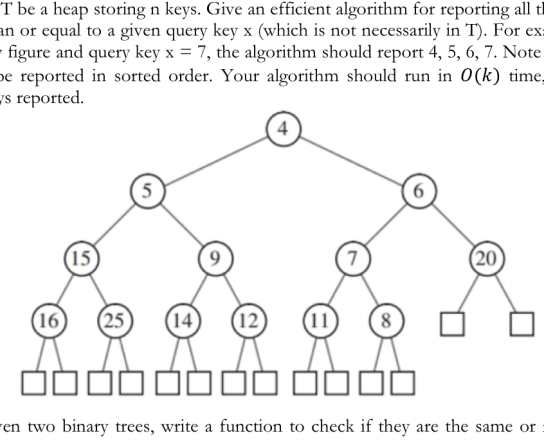
1. [2 + 3 = 5 points] What does the following algorithm do? Analyze its worst-case running time, and express it using "Big-Oh" notation.

   Algorithm Bar (a, n)
   Input: two integers, a and n
   Output: ?
   $k \leftarrow n$
   $b \leftarrow 1$
   $c \leftarrow a$
   while $k > 0$ **do**
       if $k \bmod 2 = 0$ **then**
           $k \leftarrow k/2$
           $c \leftarrow c * c$
       else
           $k \leftarrow k - 1$
           $b \leftarrow b * c$
   return b

2. [3 points] Algorithm A and B spend exactly $T_A(n) = 0.1n^2 \log_{10} n$ and $T_B(n) = 2.5n^2$ microseconds respectively, for a problem of size n. Choose the algorithm, which is better in the Big-O sense, and find out a problem size n_0 such that for any larger size $n > n_0$ the chosen algorithm outperforms the other. If your problems are of the size $n \leq 10^9$, which algorithm, will you recommend to use?
3. [3 points] Describe in pseudo-code a linear-time algorithm for reversing a queue Q. To access the queue, you are only allowed to use the methods of queue ADT. Hint: Consider using an auxiliary data structure.
4. [3 points] Draw a single binary tree T such that
   - each internal node of T stores a single character
   - a preorder traversal of T yields ABDGHEICFJ; and
   - an inorder traversal of T yields GDHBEIAFJC
5. [2 points] You are asked to implement a stack. You can use either a singly linked list or doubly linked list. Which one will you choose and why?
6. [3 + 1 = 4 points] Give pseudo-code of the algorithm least-common-ancestor that takes as input two nodes v and w in a tree T, and that gives as output the least common ancestor of v and w in T. What is the time complexity of your algorithm in terms of O?
7. [4 points] Remove from the given binary search tree the following keys (in this order): 65, 76, 88, 97. Draw the tree after each removal.



8. [5 points] Illustrate the execution of the bottom-up construction of a heap on the following sequence:(2, 5, 16, 4, 10, 23, 39, 18, 26, 15, 7, 9, 30, 31, 40).
9. [3 points] Let T be a heap storing n keys. Give an efficient algorithm for reporting all the keys in T that are smaller than or equal to a given query key x (which is not necessarily in T). For example, given the heap of below figure and query key x = 7, the algorithm should report 4, 5, 6, 7. Note that the keys do not need to be reported in sorted order. Your algorithm should run in $O(k)$ time, where k is the number of keys reported.



10. [3 points] Given two binary trees, write a function to check if they are the same or not. Two binary trees are considered the same if they are structurally identical and the nodes have the same value.

```
Input:     1         1       Input:     1          1       Input:     1          1
          / \       / \                /            \                / \        / \
         2   3     2   3              2              2              2   1      1    2
        [1,2,3], [1,2,3]            [1,2], [1, null,2]             [1,2,1], [1,1,2]
Output: true                 Output: false                 Output: false
```


11. [4 + 4 + 7 = 15 points] Draw 11-item hash table that results from hash function $h(i) = i \bmod 11$ to keys 22, 1, 13, 11, 24, 33, 18, 42, and 31 for each of the following methods. The items are inserted in order mentioned above in the hash table.
    - Separate chaining
    - Open addressing/linear probing
    - Double hashing;
      a) Use $d(i) = [i \bmod (m-1)] + 1$ for the secondary hash function. In both hash functions, the argument i is the value of the key being hashed. Here, m = 11.
      b) Use $d(i) = 7 - (i \bmod 7)$ for the secondary hash function. In both hash functions, the argument i is the value of the key being hashed.

Items	h(x)	(a) d(x)	(b) d(x)
22			
1			
13			
11			
24			
33			
18			
42			
31			

Index	Separate chaining	Open addressing	(a) Double Hashing	(b) Double Hashing
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

Q1 1) The algorithm computes a^n using exponentiation by squaring, an efficient way to compute powers
2) It maintains variables b (accumulator) and c (base)
3) repeatedly squares c when K is even
4) multiplies 'b' by 'c' when 'K' is odd until $K=0$
5) Worst-case time complexity
Each iteration cuts 'K' roughly in half → Total time $= O(\log n)$

Algorithm Bar (a,n)	Worst Case Time
$K \leftarrow n$	1
$b \leftarrow 1$	1
$c \leftarrow a$	1
while $K>0$ do	$n-1$
if $K \bmod 2 = 0$ then	$\log n$
$K \leftarrow K/2$	1
$c \leftarrow c*c$	1
else	1
$K \leftarrow K-1$	1
$b \leftarrow b*c$	1
return b	1
	$O(\log n)$

Q2 Given: 1) $T_A(n) = 0.1n^2 \log n$
2) $T_B(n) = 2.5n^2$

Big O : 1) $T_A(n) = O(n^2 \log n)$
2) $T_B(n) = O(n^2)$

Therefore, asymptotically, Algorithm B is better

Find n_0:
Set
$0.1 n^2 \log n = 2.5n^2 \rightarrow \log n = 25 \rightarrow$
$n_0 = e^{25} \approx 7.2 \times 10^{10}$

For $n \leq 10^6$:
Check actual numbers

1) $T_A \approx 0.1 \times 10^{12} \times 14 = 1.4 \times 10^{12}$

2) $T_B \approx 2.5 \times 10^{12}$

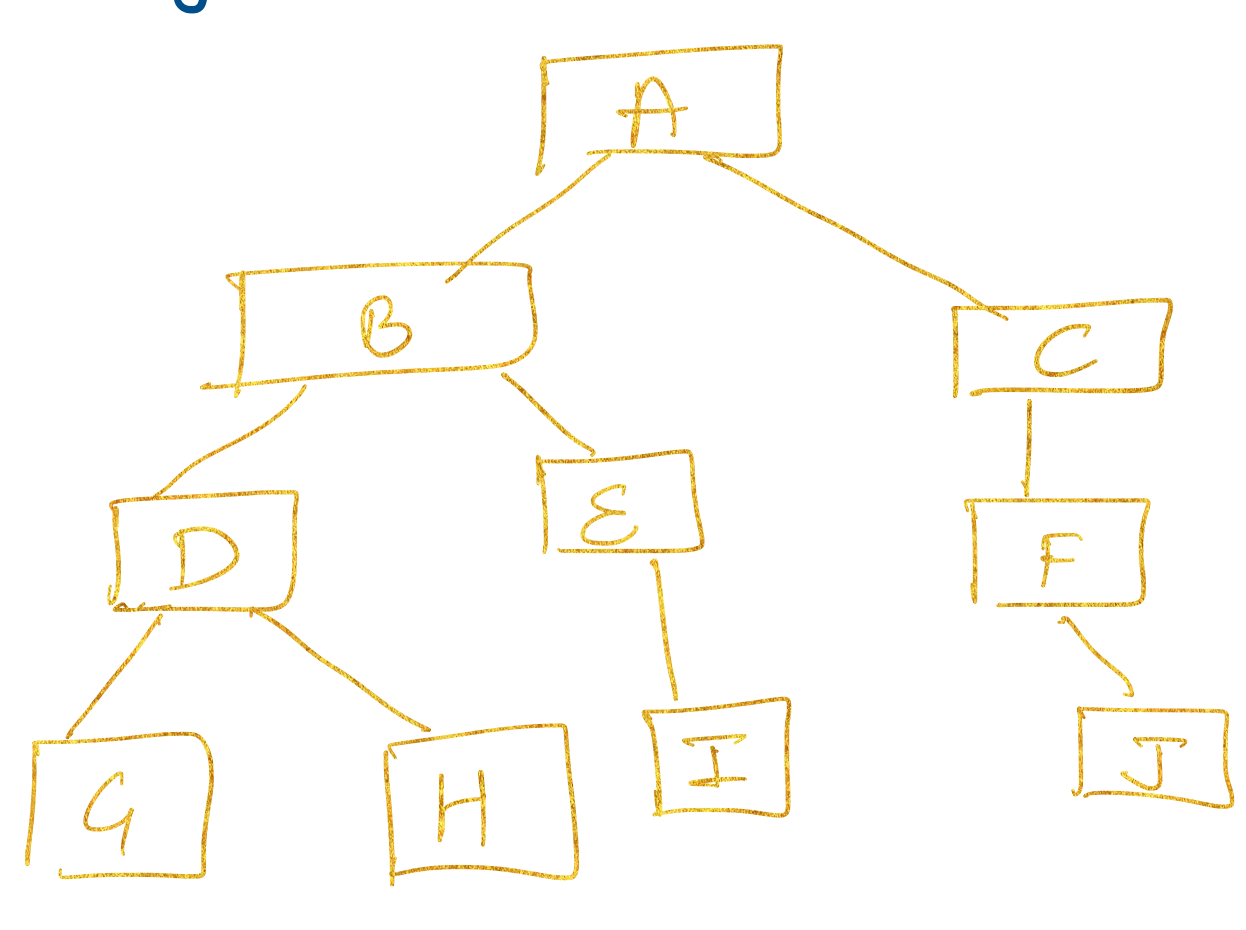
For Problem size $n \leq 10^9$, Algorithm T_A takes less time than Algorithm T_B. So, Algorithm T_A is recomonded for all $n \leq 10^9$

Q3 Pseudocode [using Stack]

```
reverseQueue (Q):
    S ← empty stack
    while not Q.isEmpty ():
        S.push (Q.dequeue ())
    while not S.isEmpty ():
        Q.enqueue (S.pop ())
```

Q4 Given: 1) Preorder : ABDGHEICFJ
2) Inorder : GDHBEIAFJC

Steps : 1) Root A
2) Left Subtree (inorder) : GDHBEI
3) Right Subtree : FJC



Q5 Choice : Singly Linked List

Reason
1) Less memory (no need for prev pointers).

2) Stack operations only require one end (top).

3) Simpler and faster to implement.

4) The time complexity for inserting and deleting an element from the singly linked list is O(n)

5) Since primary functions of stack are PUSH and POP, therefore singly linked list would be more suitable.

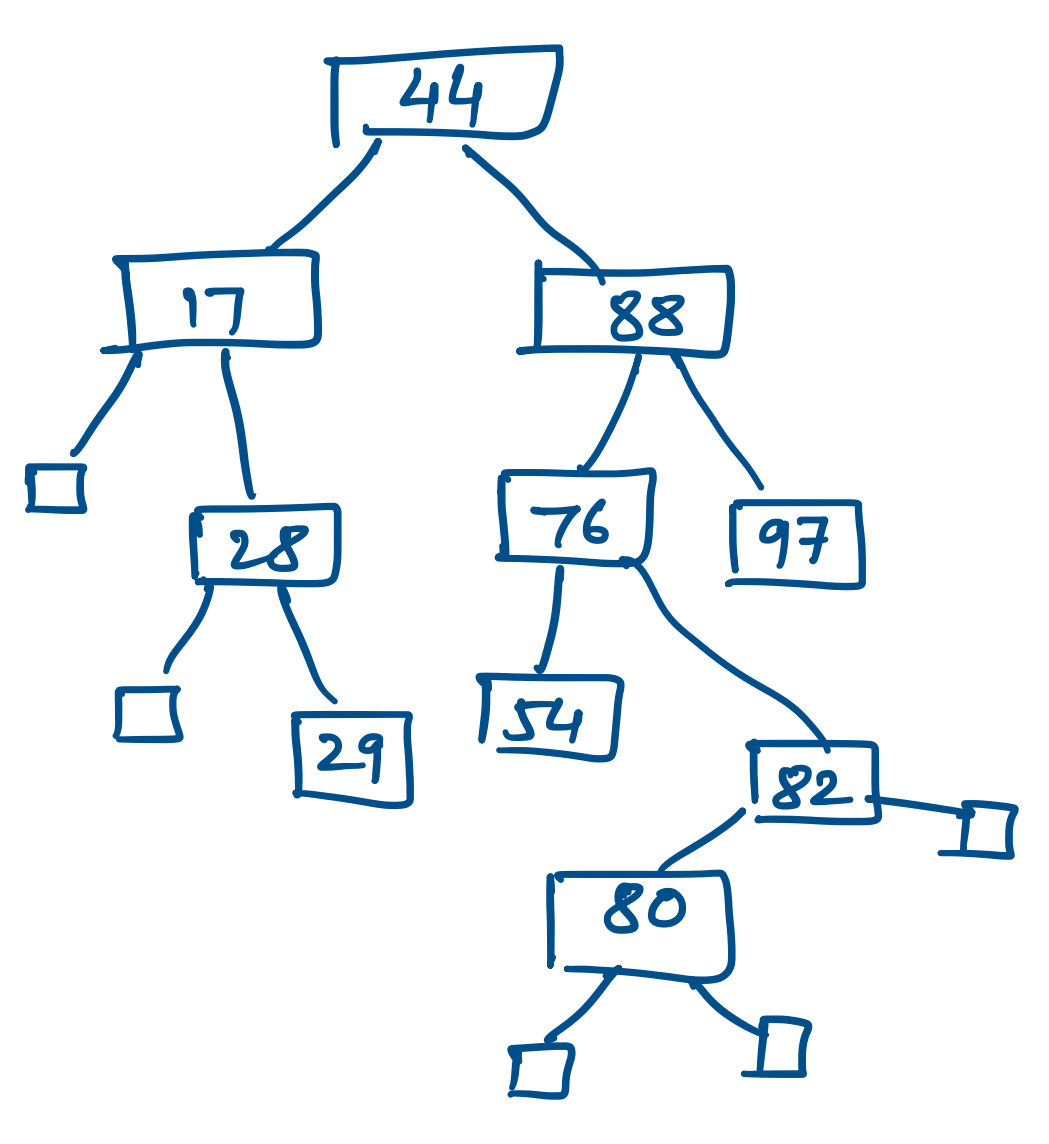
Q6 Least Common Ancestor

Pseudocode:

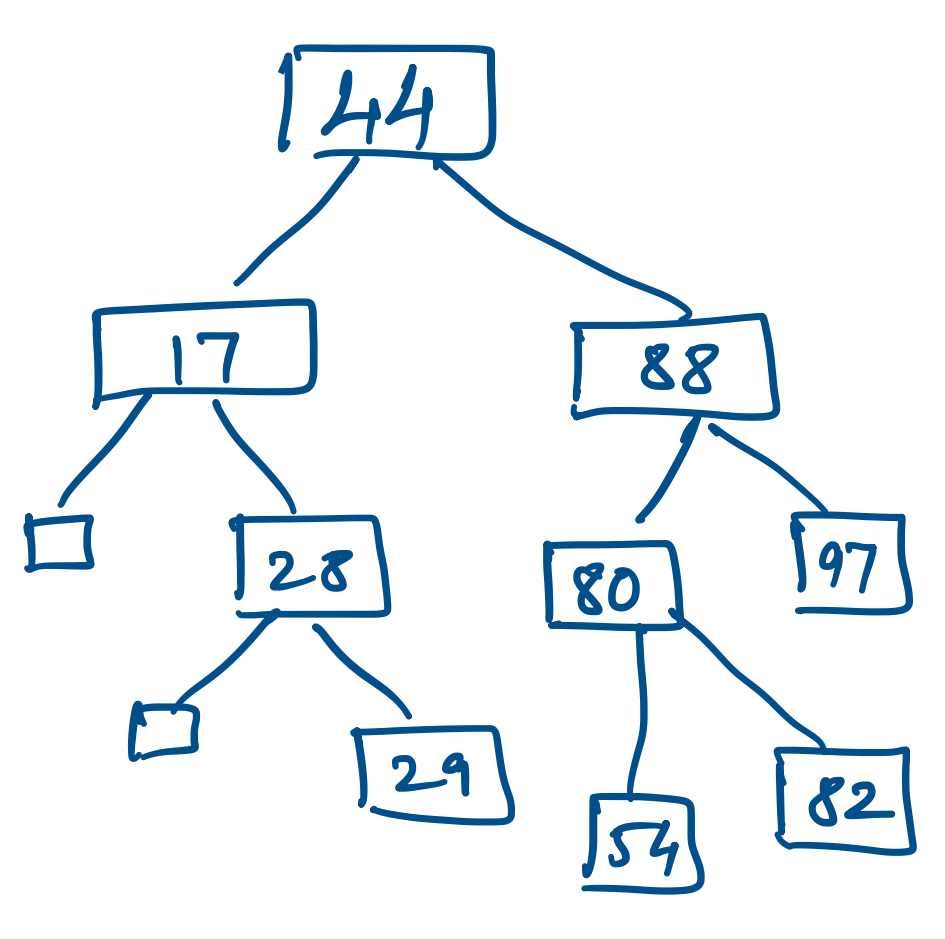
```
leastCommonAncestor (T, v, w):

    P_v  ← path from root to v
    P_w  ← path from root to w
    LCA  ← null

    for i in 0 to min (len(P_v), len(P_w)):
        if P_v [i] == P_w [i]:
            LCA ← P_v[i]
        else :
            break

    return LCA
```

Time Complexity: $O(h)$
h = height of the tree.

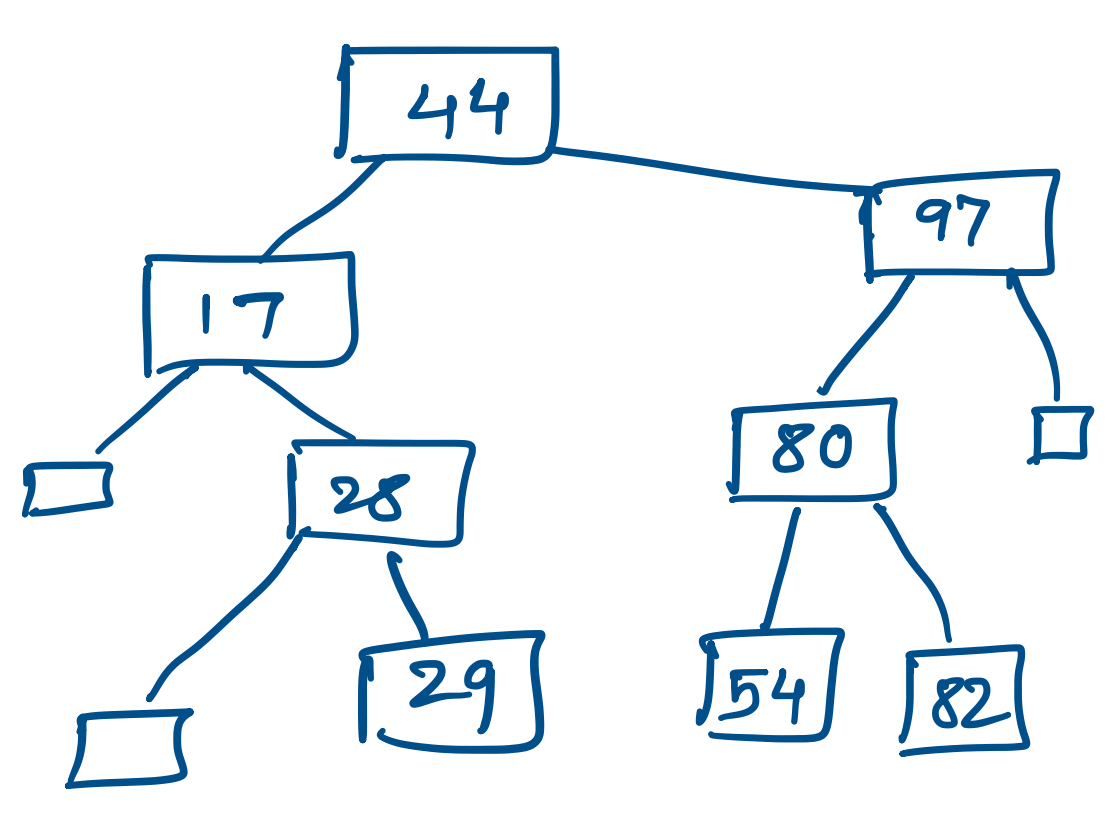
Q7 Removing 65



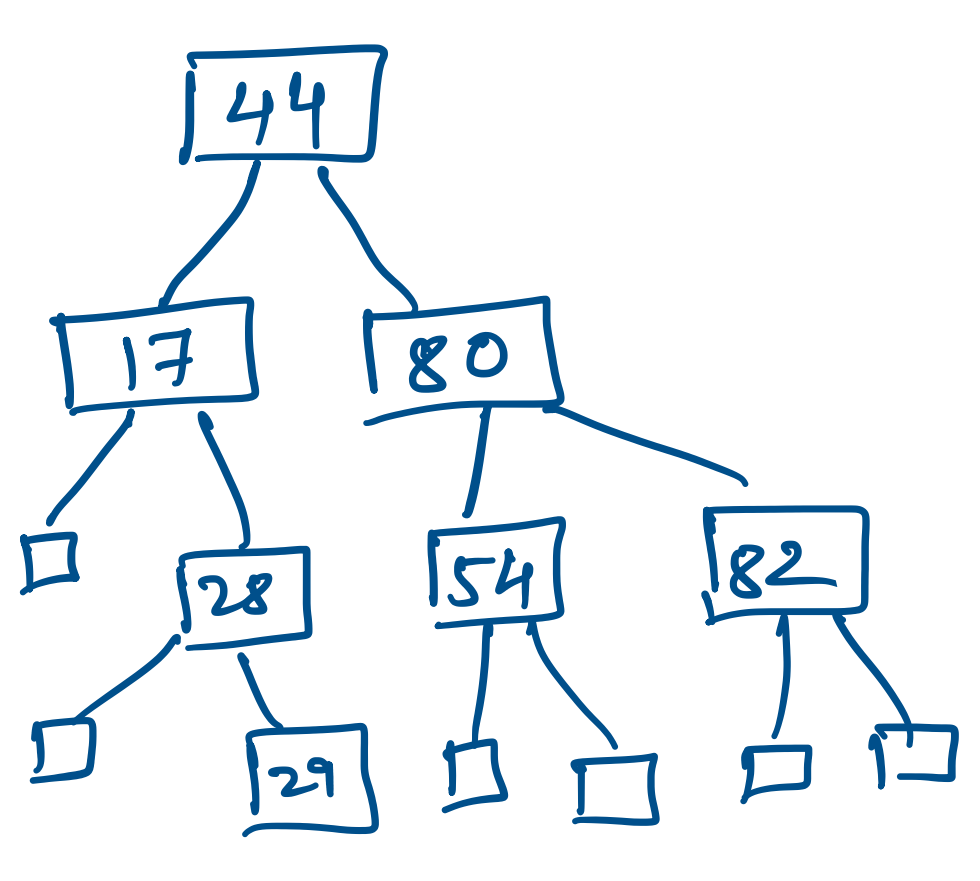
Removal of 76



Removal of 88



Removal of 97



Q8 Given sequence
2, 5, 16, 4, 10, 23, 39, 18, 26, 15, 7, 9, 30, 31, 40

Step 1: Place in array

Index	Value
1	2
2	5
3	16
4	4
5	10
6	23
7	39
8	18
9	26
10	15
11	7
12	9
13	30
14	31
15	40

Step 2: Heapify bottom-up

start from last non-leaf node
index (7) upto root

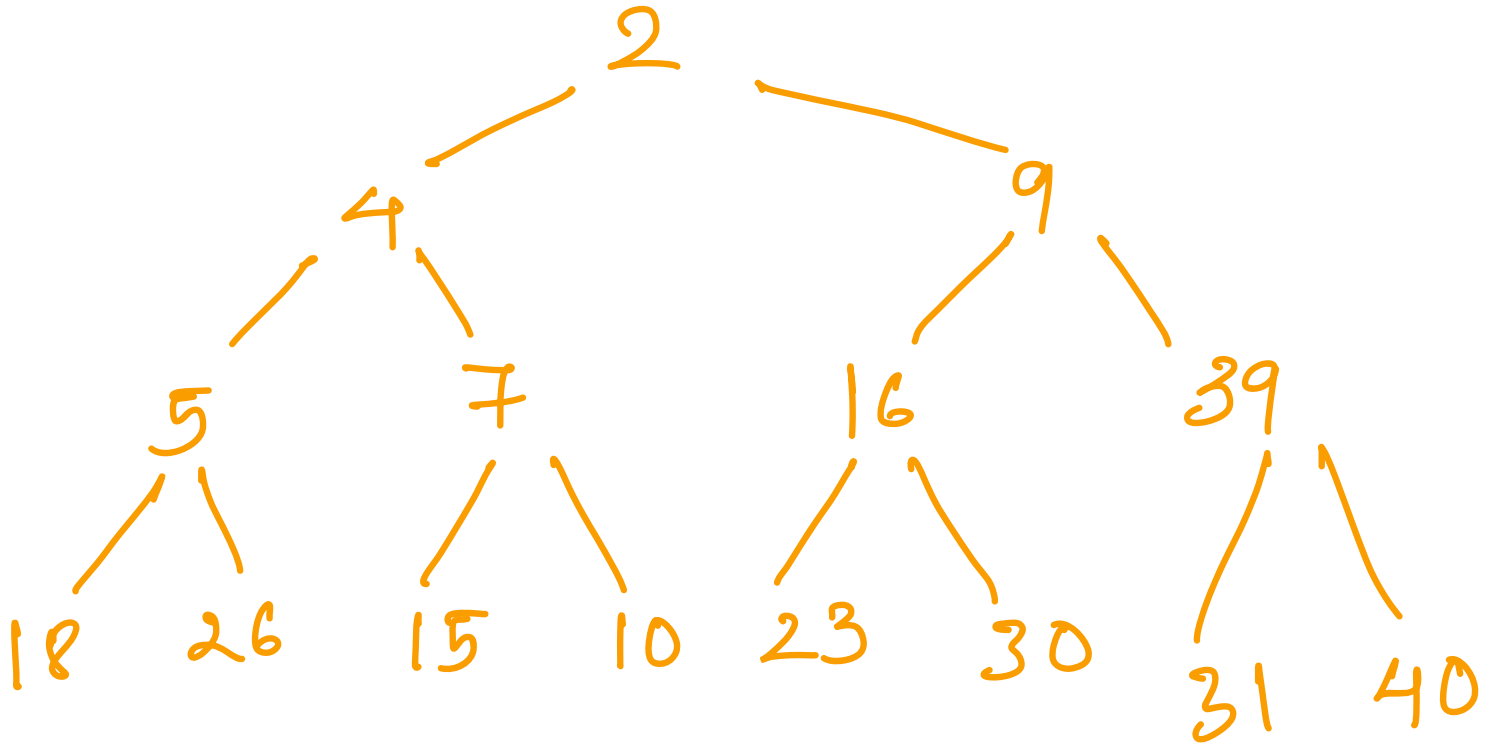
Node 7 : ok (39 < 31, 40)
Node 6 : Swap 23 ⟷ 9
Node 5 : Swap 10 ⟷ 7
Node 4 : ok (4 < 18, 26)
Node 3 : Swap 16 ⟷ 9
Node 2 : Swap 5 ⟷ 4
Node 1 : ok (2 < 4, 9)

Final Min-Heap Array.

Index	Value
1	2
2	4
3	9
4	5
5	7
6	16
7	39

8	18
9	26
10	15
11	10
12	23
13	30
14	31
15	40

Final Min-Heap Tree



Q9 Algorithm: smallerkeys (currentNode, x)

Input : A query key x

Output : Array of keys smaller than input x

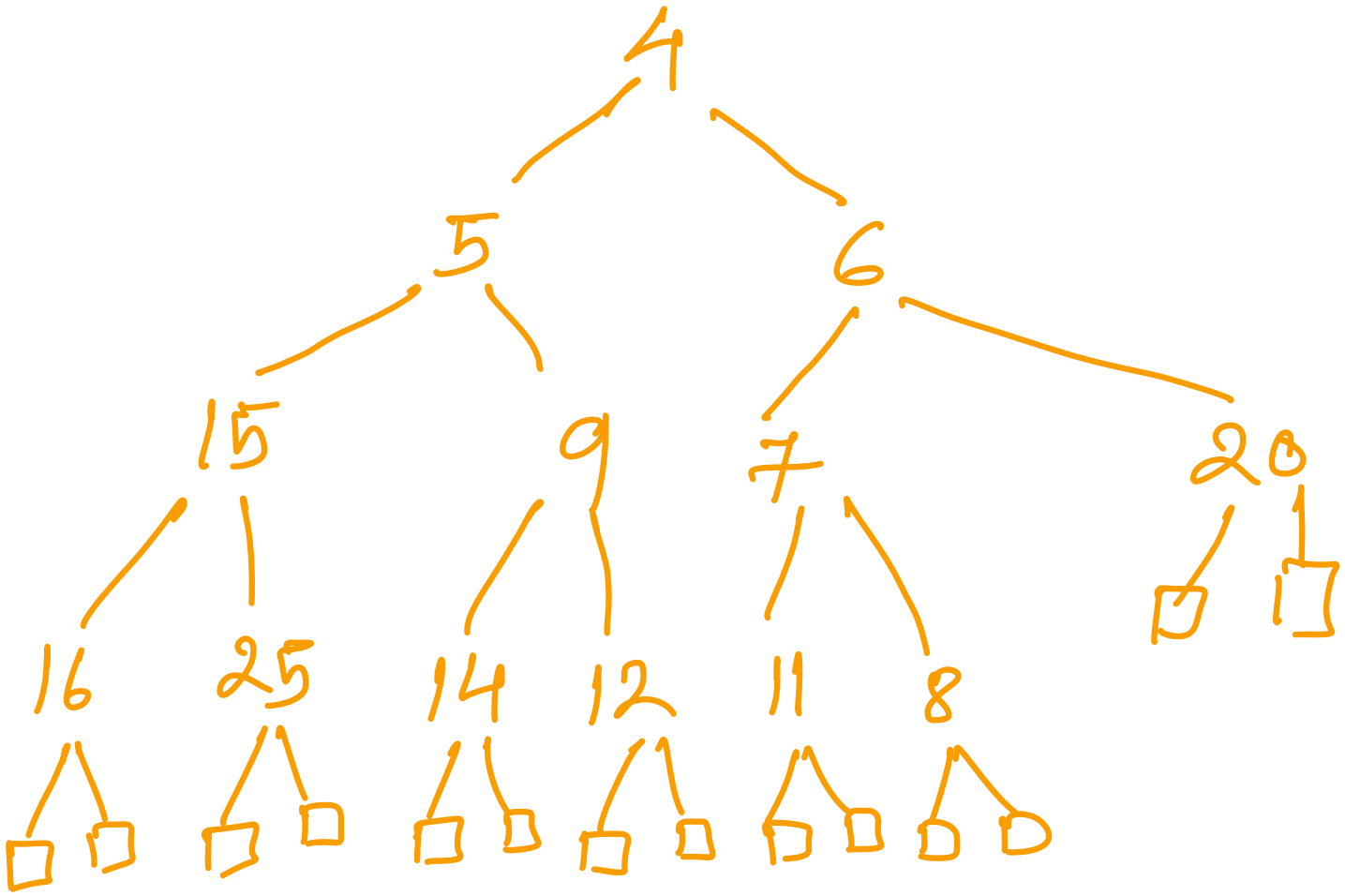
```
int [] result
if (currentNode → value ≥ x) then
        return result

else
    result.push(currentNode → Value)
  if (currentNode → left != NULL) then
      smallerkeys (currentNode → left, x)
 if (currentNode → right != NULL) then
      smallerkeys (currentNode → right, x)
```

pseudocode :

```
reportkeys (T, i, x):
```

if $i \geq$ size of T or $T[i] > x$:
    return

output $T[i]$
report keys $(T, \text{leftChild}(i), x)$
report keys $(T, \text{rightChild}(i), x)$



Runs in : $O(k)$
where k = number of reported keys.

Q10

Algorithm / Pseudocode

```
isIdenticalTree (T1, T2):

    if T1 is null and T2 is null:
        return True

    if T1 is null or T2 is null:
        return False

    if T1.val ≠ T2.val:
        return False

    return isIdenticalTree (T1.left,
    T2.left) and isIdenticalTree(
    T1.right , T2.right)
```

Q11 Given: 1) $m = 11$

2) Keys (in order): 22, 1, 13, 11, 24, 33, 18, 42, 31

$h(x) = x \bmod 11$

Item	h(x)	a) d(x)	b) d(x)
22	0	3	6
1	1	2	6
13	2	4	1
11	0	2	3
24	2	5	4
33	0	4	2
18	7	9	3
42	9	3	7
31	9	2	4

Index	Separate Chaining	Open Addressing (Linear Probing)	(a) Double Hashing	(b) Double Hashing
0	22 → 11 → 33	22	22	22
1	1	1	1	1
2	13 → 24	13	13	13
3		11	11	11
4		24	4 → moved keys → 33	33
5		33	5 → 18	
6			31	24
7	18	7	24	18
8			33	

9	42 → 31	9 → 42	42	42
10		31		31

Separate Chaining → all collisions go in the same bucket (linked list)

Open addressing → probe next available slot

Double hashing (a) → jump using $d(x) = (x \bmod 10) + 1$

Double hashing (b) → jump using $d(x) = 7 - (x \bmod 7)$

[A submission by Hritika Kucheriya
ID: 801456028
hkucheri@uncc.edu]