Name: Kamalnath. Polakam

Entry No: 2013CS10244

COL106: Data Structures. I semester, 2016-17.
Minor II
1 PM to 2 PM, 9th October 2016.

Attendance Sheet Serial Number: 67

Question	1 (11 marks)	2 (9 marks)	Total (20 marks)
Marks	8	8	16

0. The Dean has instructed us to give **no clarifications during the course of the exam** to prevent students from being disturbed. If you have any doubts, please state your assumptions and answer the question to the best of your ability.
1. Write your answers on the **printed question paper** in the space provided. **ROUGH SHEETS WILL NOT BE COLLECTED.**
2. Please write your name on every page and enter your serial number in the box above. **If you miss out any of these: -1 and no rechecking.**
3. **No pseudocode** means: For every loop you use, you must explain what it will do to the input. You cannot write things like "x ← x + y", you must explain the significance of every step in words. You cannot write "for i ← 1 to ..." or "while <condition>...", you must *explain* what the loop achieves. In summary: if we need to interpret how your description will treat a particular input then it is pseudocode.

Q1. Short answer questions (Total marks = 11)

Marks: 8

Q1.1 (2 marks) Consider the 2-4 tree given in Figure 1.

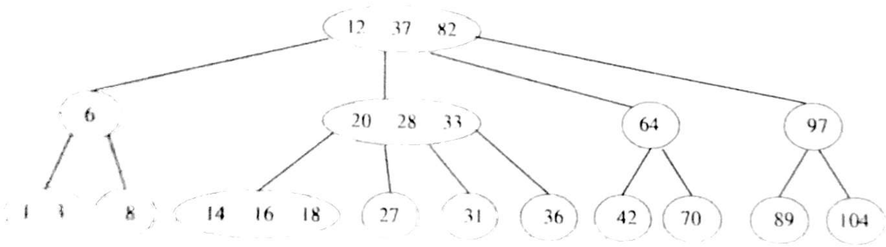
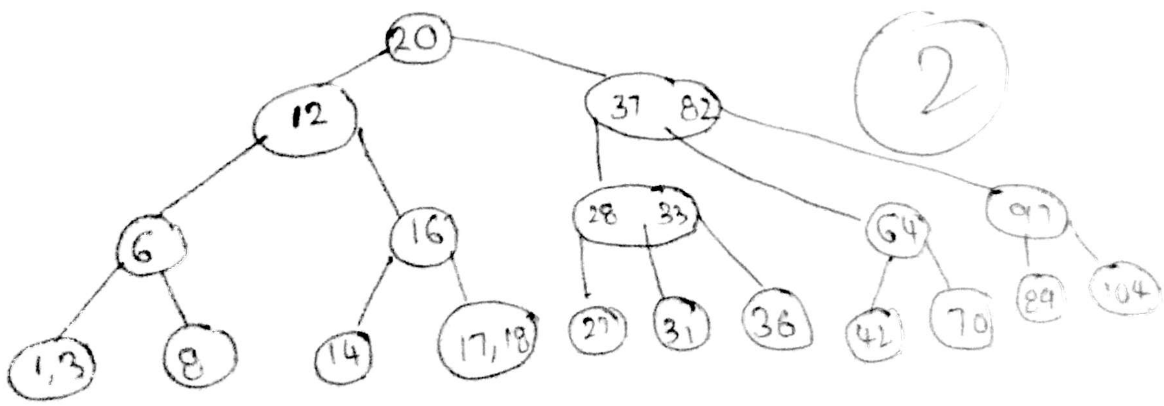


Figure 1: A 2-4 tree

Draw the state of the tree after inserting 17. You *do not* have to show the intermediate steps, only the final answer.



Scanned by CamScanner

Q1.2 (2 marks) Consider open addressing with *quadratic probing* i.e. if the hashtable A has size n and the hash function is $h(\cdot)$, then if we have to insert x and if the location $A[h(x) \mod n]$ is occupied, we try $A[h(x) + 1^2 \mod n]$, $A[h(x) + 2^2 \mod n]$, $A[h(x) + 3^2 \mod n]$, etc in sequence till we find an empty slot and insert x there. Consider a hashtable of size 13 and the hash function $h(x) = (3x + 11) \mod 13$. Assume that 21, 43 and 57 are already inserted into this hashtable. Where will 73 be inserted? To get full marks you must explain how you arrived at the answer. (1 mark for the correct answer and 1 mark for the correct explanation. Zero if only the answer is written with no explanation).

2

first 21

$h(21) = (3\times21+11) \bmod 13$
$= 74 \bmod 13$
$= 9$

second 43
$h(43) = (140) \bmod 13$
$= 10$

Probing (1)
$9 + 1^2 \bmod 13$
$= 10$
$A[10] = 43$

10

third 57
$h(57) = 0$

Probing (2)
$(9+2^2) \bmod 13$

$A[0] = 57$

~~A[5] ... first~~

Now 73
$h(73) = (230) \bmod 13$
$= 9$
already filled

$A[9+$

0	1	2	3	4	5	6	7	8	9	10	11	12
57									21	43		

0	1	2	3	4	5	6	7	8	9	10	11
57				73					21	43	

Q1.3 (2 marks) Prove that the following statement is false: It is not possible to make an AVL tree on 10 or more nodes such that the left subtree of the root contains less than one third of the nodes that the right subtree contains.

Q1.4 (3 marks) Consider a probabilistic skip list with parameter p (i.e. a key in S_i is selected for S_{i+1} with probability p) containing the elements $x_1, x_2, \ldots, x_n$ numbered in ascending order (i.e. the order in which they appear in the base list S_0). Consider two elements x_i and x_j such that $i < j$. Under what condition (on the heights of the various elements in the skip list) is it true that the search path to x_j will go through x_i? What is the probability that the search path to x_j goes through x_i? (2 marks for the condition and 1 mark for the probability calculation).

Condition :- height$(x_i) \geq$ height(x_j) and
$\forall k \in (i, j)$ (k lies between i and
~~height(x_k)~~ height$(x_k) \leq$ height$(x_i$

2

probability calculation : $p($ $h(x_i) > h(x_j), \forall k \in (i,j)$
$h(x_i) > h(x$

$h(x_t) = h_t$
$\forall t \in [i+1, j$
$\max(h_{i+1}, h_{i+2}, \ldots, h_j) + 2$
(p)

$x_t) = h_t$
mean x_t was promoted h_t time

Q1.5 (2 marks) Any find path in a binary search tree can be written as a sequence of turns, some left turns and some right turns, e.g., if the root has key 12 and we are searching for 26, the first turn is Right and so on. Let this sequence of turns be $t_1, t_2, \ldots, t_k$ (each t_i is either L or R) and let $x_1, x_2, \ldots, x_k$ be the the nodes at which the turns were taken (e.g. x_1 is always the root). If t_i and t_j are both left turns, under what condition is $x_j > x_i$?

A) ~~let~~ ~~the~~ ~~new~~ node ~~be~~ ~~xN~~ ~~has Key xN~~ ~~as~~

As during traversal it is given both t_i and t_j are both left turns, The Node we are searching for is less than both the given N

case 1) If $i > j$. That means the node with key x_i is a child of x_j (since BST traversal path goes in this way) as t_j is left ~~x~~. Node(x_i) is in the left subtree of N

$\therefore x_j > x_i$, so for $x_j > x_i$ $i > j$ (In all other ca Node(x_i) is right c Node(x_j) so it g $x_j < x_i$

Marks: 8

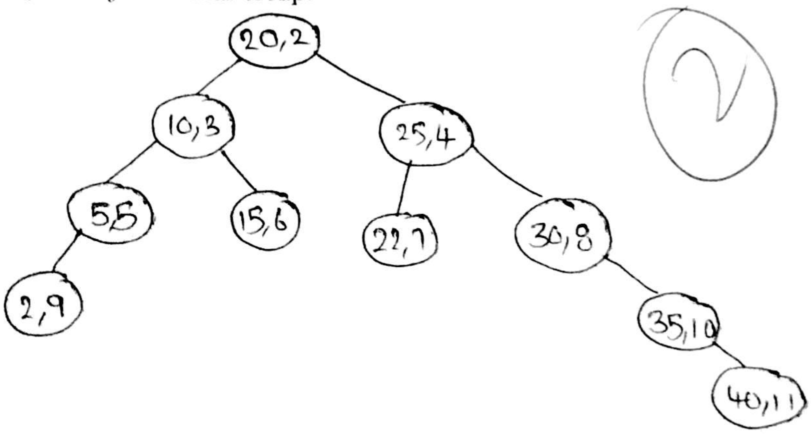
Q2. (Tree + heap = Treap. Total marks = 9).
A *Treap* is a special kind of binary search tree which also has the property of a heap. Each item in a treap is a tuple of two values i.e. $x_i = (k_i, p_i)$ where k_i is a key and p_i is a priority. A treap is a binary search tree on the keys of the items and maintains the *min-heap order* property on the priorities i.e. the priority value of a node is less than the priority value of any children it may have. Note that the Treap *does not* have to have the structural property of a heap i.e., it may not be a complete binary tree but must have the order property of a heap.

Q2.1 (2 marks). Construct a Treap on the set

$$S = \{(22,7), (20,2), (2,9), (10,3), (35,10), (40,11), (15,6), (30,8), (25,4), (5,5)\}.$$

8 0 7 1 8 9 4 6 2 3

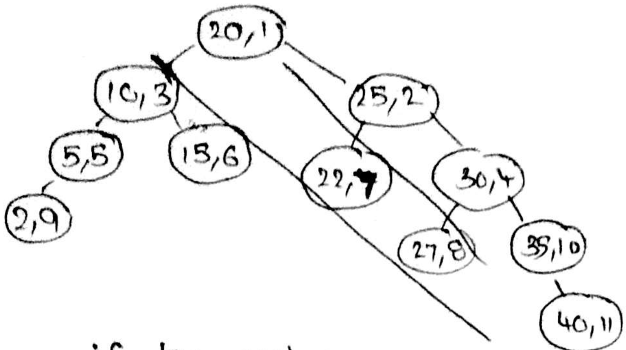
You do not need to show the steps. only the final treap.



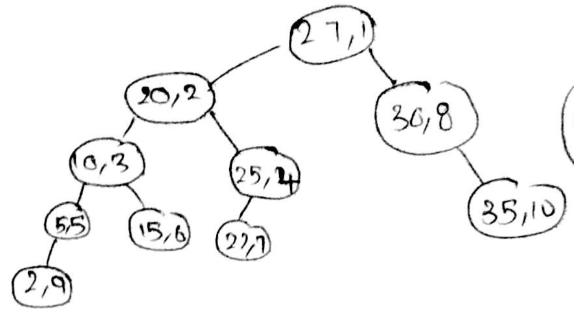
So time complexity is O

Q2.2 (2 marks). In the Treap constructed in the previous part insert (27, 1). The insertion algorithm proceeds by first inserting the node in its appropriate place as a binary search tree node and then adjusting the key correctly. Draw the heap after the insertion has completed. (Use the space given overleaf on page 4).

5A (Continued)) As it can be seen through rotations every sub BS can be successfully converted to new sub treap with N' as it new root. The rotations take constant time, B if the whole tree can't be corrected to a treap in 'phase then we must continue to Next phase'. This is recursive i then we must continue to ... nature and should be Continued until the whole tree becomes treap. In ... the 'N' becomes the new root so whole tree height is tra ... time



if key values can be swapped.



if key values can't be swapped.

2

Q2.3 (5 marks) Explain the Treap insertion algorithm. As mentioned before you must begin by doing a BST insertion of the new (key, priority) pair and then adjust the priorities. Explain how this adjustment phase proceeds. Note: (1) **No pseudocode**. (2) You must justify your steps, i.e. argue the for the correctness of your algorithm. Without an argument of correctness you will get a maximum of 2 marks even for a perfectly correct answer. (3) Your insertion mechanism should work in $O(h)$ time where h is the height of the Treap. (Hint: If the priorities and keys are all different, as they are in this case, then any set of keys gives a unique Treap, so in the previous part you can find the correct answer simply by constructing a Treap on $S \cup \{(27, 1)\}$ from scratch. The purpose of the previous part is to help you figure out the adjustment phase of the insertion algorithm. Since you can tell the final answer by constructing the final Treap from scratch, you can check using this example if you are doing the adjustment phase correctly.)

A) Treap insertion algorithm.

You also need to justify that the rotation maintains the heap order property

4 the two

1) Begin with root if root(Key) < New Node(Key) go to the right subtree of the root and repeat the same algorithm, if root(Key) > New Node(Key) go to the left subtree of the root and repeat the same algorithm. Repeat until you reach ~~a leaf and~~ the perfect Node attach the New Node there as a child, (This is BST insertion).

2) Now adjustment phase. If they newly created tree follows treap properties then well and good. otherwise we get two cases.

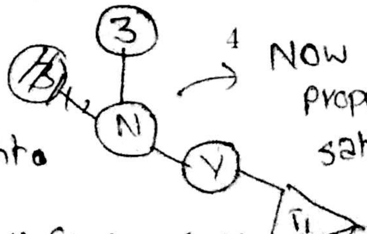
'N' is new node

(phase 1)

N(priority) < Y(priority)

phase (1), we must rotate this subtree such that BST properties are still valid.

Note:-



Now Treap property is satisfied.

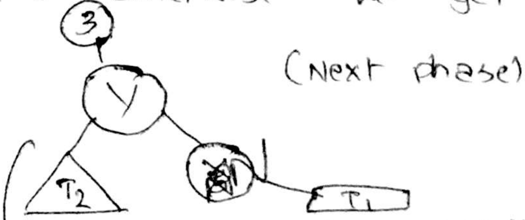
if Y was right child

similar if Y was left child).

Before, 3(Key) ≤ N(Key) ≤ Y(Key) ≤ Root(T1)(Key)

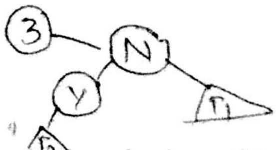
Now, 3(Key) ≤ N(Key) ≤ Y(Key) ≤ Root(T1)(Key).

(Next phase)



N(priority) < Y(priority)

Phase 1 already happend so now the subtree's T1 and 'N' are treaps but Y is just a BST.



if Y was right child.

Before:- 3(Key) ≤ N(Key) ≤ Y(Key)

Now:- 3(Key) ≤ N(Key ≤ Y(Key)