Jorge Solis
jas2430
04/13/2017
Professor Paul Blaer

Problem Set 4 – Written Component

5.1 – Hash Tables

a. hash(x) = x mod 10

1 – 4371 3 – 6173, 1323 4 – 4344 9 – 1989, 9679, 4199

b. hash(x) = (x mod 10 + i) mod 10

0 – 9679 1 – 4371 2 – 1989 3 – 1323 4 – 6173 5 – 4344

9 – 4199

c. hash(x) = (x mod 10 + i^2) mod 10

0 – 9679 1 – 4371 3 – 1323 4 – 6173 5 – 4344 8 – 1989

9 – 4199

d. hash(x) = ((x mod 10 + i*(7 – x mod 7)) mod 10

1 – 4371 3 – 1323 4 – 6173 5 – 9679 7 – 4344 9 – 4199

1989 can not be inserted. The table size is not prime.

5.2 Rehashed Tables

a. hash(x) = x mod 19

0 – 4199 1 – 4371 8 – 9679 12 – 4344, 1323 13 – 1989 17 – 6173

b. hash(x) = (x mod 19 + i) mod 19

0 – 4199 1 – 4371 8 – 9679 12 – 1323 13 – 4344 14 – 1989

17 – 6173

c. hash(x) = (x mod 19 + i^2) mod 19

0 – 4199 1 – 4371 8 – 9679 12 – 1323 13 – 4344 14 – 1989

17 – 6173

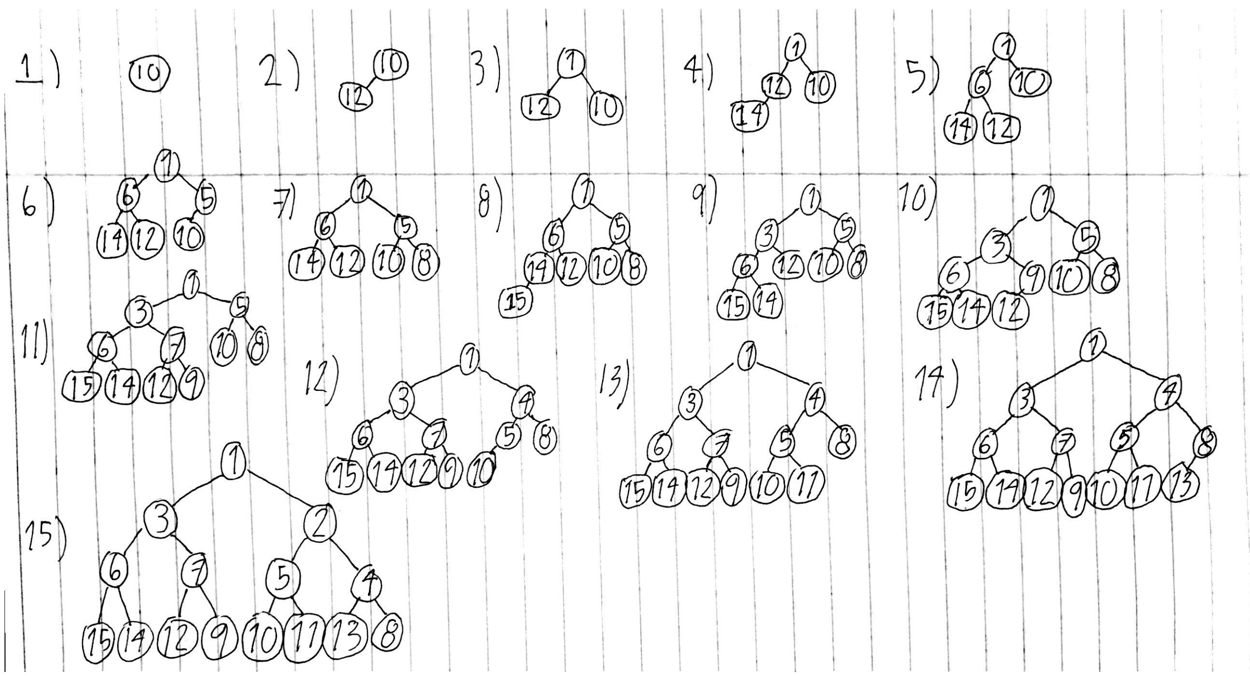
d. hash(x) = ((x mod 19 + i*(7 – x mod 7)) mod 19

0 – 4199 1 – 4371 8 – 9679 12 – 1323 13 – 1989 15 – 4344

17 – 6173

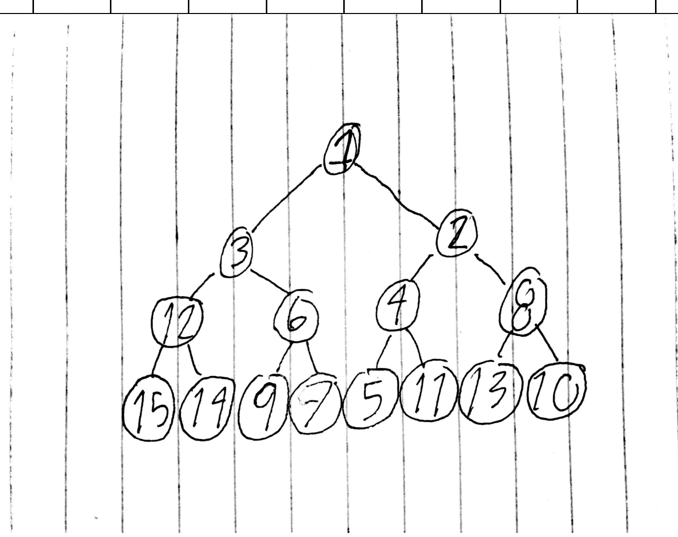
6.2 – Heaps

a.



b.

T	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	10	12	1	14	6	5	8	15	3	9	7	4	11	13	2
8	10	12	1	14	6	5		15	3	9	7	4	11	13	2
8	10	12	1	14	6	5	2	15	3	9	7	4	11	13	
5	10	12	1	14	6		2	15	3	9	7	4	11	13	8
5	10	12	1	14	6	4	2	15	3	9	7		11	13	8
14	10	12	1		6	4	2	15	3	9	7	5	11	13	8
14	10	12	1	3	6	4	2	15		9	7	5	11	13	8
12	10		1	3	6	4	2	15	14	9	7	5	11	13	8
12	10	3	1		6	4	2	15	14	9	7	5	11	13	8
10		3	1	12	6	4	2	15	14	9	7	5	11	13	8
10	1	3		12	6	4	2	15	14	9	7	5	11	13	8
10	1	3	2	12	6	4		15	14	9	7	5	11	13	8
10	1	3	2	12	6	4	8	15	14	9	7	5	11	13	
	1	3	2	12	6	4	8	15	14	9	7	5	11	13	10



6.3

Three .deleteMin() operations on the heap from 6.2a

1.

T	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	1	3	2	6	7	5	4	15	14	12	9	10	11	13	8
8		3	2	6	7	5	4	15	14	12	9	10	11	13	
8	2	3		6	7	5	4	15	14	12	9	10	11	13	
8	2	3	4	6	7	5		15	14	12	9	10	11	13	
8	2	3	4	6	7	5	8	15	14	12	9	10	11	13	
	2	3	4	6	7	5	8	15	14	12	9	10	11	13	8

2.

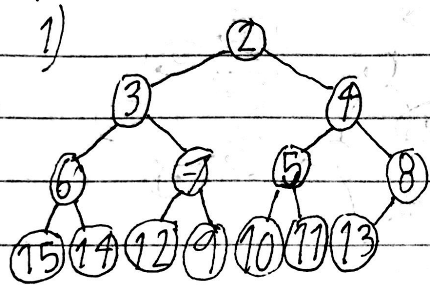
T	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	2	3	4	6	7	5	8	15	14	12	9	10	11	13	
13		3	4	6	7	5	8	15	14	12	9	10	11		
13	3		4	6	7	5	8	15	14	12	9	10	11		
13	3	6	4		7	5	8	15	14	12	9	10	11		
	3	6	4	13	7	5	8	15	14	12	9	10	11		

3.

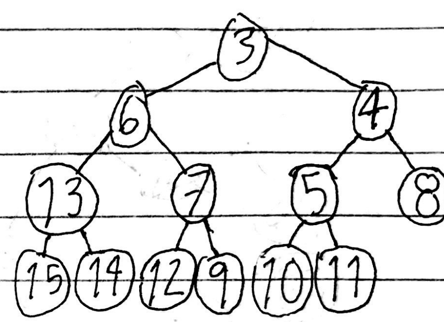
T	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	3	6	4	13	7	5	8	15	14	12	9	10			
11		6	4	13	7	5	8	15	14	12	9	10			
11	4	6		13	7	5	8	15	14	12	9	10			
11	4	6	5	13	7		8	15	14	12	9	10			
11	4	6	5	13	7	10	8	15	14	12	9				
	4	6	5	13	7	10	8	15	14	12	9	11			

Delete Min:

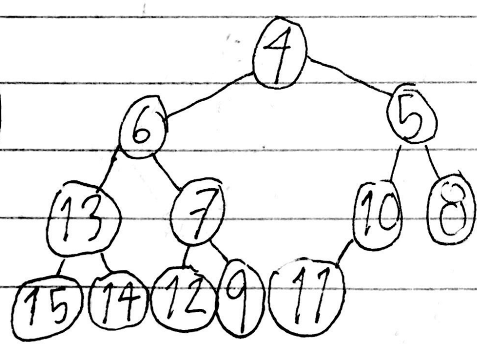
1)



2)



3)



6.8

a.

The value of every node in a MinHeap must be smaller than each of its children's values. This means that the node with the largest value, the maximum, cannot have any children, so it must be a leaf node.

b.

Assume you have a series of N values with which you will construct a binary heap. Beginning with an empty heap, each additional node *i* will increase the total number of nodes in the tree until it has N nodes. Starting with the first, every additional node *i* % 2 = 1 (every odd-numbered node) will increase the total number of leaf nodes by 1 while leaving the number of internal nodes unchanged. Every additional node *j* % 2 = 0 (every even-numbered node) will increase the total number of internal nodes by 1 while leaving the number of leaf nodes unchanged. Thus, the total number of leaf nodes *I* and total number of internal nodes *J* each equal N/2.

c.

The value of each node in a MinHeap must be larger than the value of its parent node, and smaller than each of its children's values. This means that the maximum value must be in a leaf node. This does not guarantee, however, that the maximum value will be found in any specific leaf node. Every node is only restricted relative to its parent and child nodes, therefore the maximum value can be found in any of the leaf nodes, given that every internal node is smaller than its children. This will require each leaf node to be evaluated when searching for the maximum node.