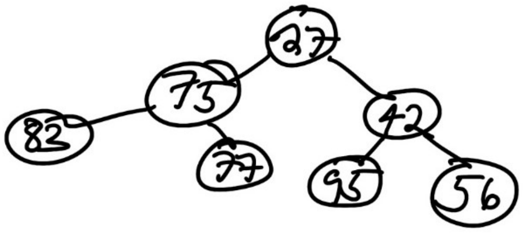
Aidan Duffy
Boston University
METCS 526

Module 4 Homework (Problems 1 through 5)

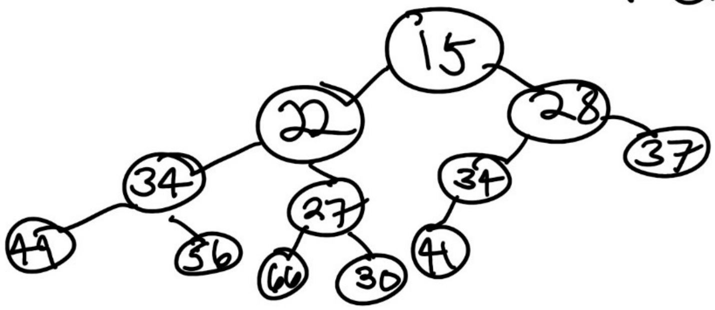
Problem 1 (10 points)

① This is a min heap be cause our root node is the smallest Key value, Since the new addition is smaller then the current root, it will replace it as the root. It is first placed as the right child of the 56 node, which it then swaps with since 56>27, then it swaps with the root b/c 42>27, which leads to the following tree:



Problem 2 (10 points).

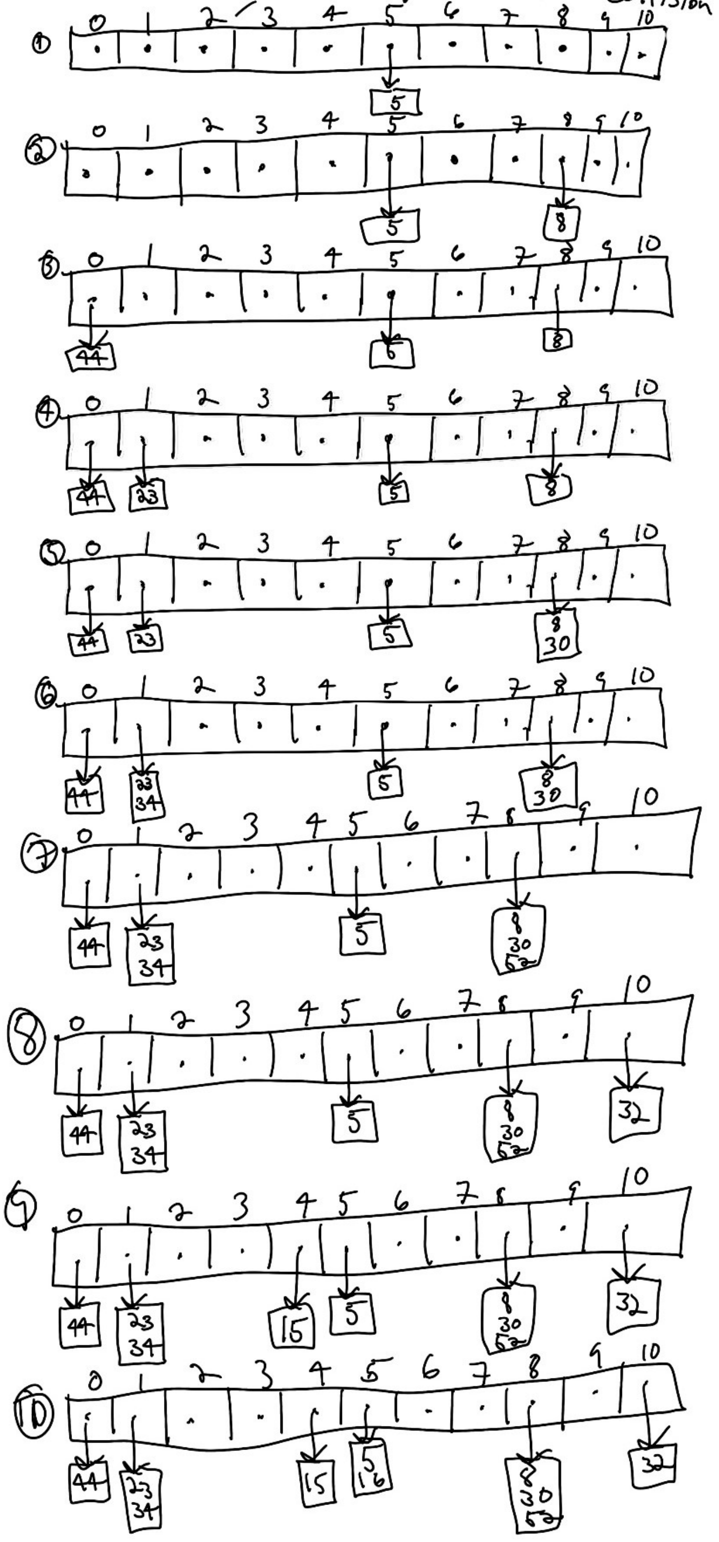
2 This is a min-heap again, so we will be removing the root & down heap bubbling. The furthest down elem., 34, goes and replaces the 5. Then 34 swaps with the right child (15) since it has the smaller key. Then it swaps with its left child as 34 is smaller than its right child but larger than its left child. This is the resulting tree:



Problems 3 (10 points each).

③ <5, 8, 44, 23, 30, 34, 52, 32, 15, 16>

size N=11 chaining used for collision



Problem 4 (10 points).

④ <5, 8, 44, 23, 30, 34, 52, 32, 15, 16>

N=11, linear probing used

①

0	1	2	3	4	5	6	7	8	9	10
					5					

②

0	1	2	3	4	5	6	7	8	9	10
					5			8		

③

0	1	2	3	4	5	6	7	8	9	10
44					5			8		

④

0	1	2	3	4	5	6	7	8	9	10
44	23				5			8		

⑤

0	1	2	3	4	5	6	7	8	9	10
44	23				5			8	30	

⑥

0	1	2	3	4	5	6	7	8	9	10
44	23	34			5			8	30	

⑦

0	1	2	3	4	5	6	7	8	9	10
44	23	34			5			8	30	52

⑧

0	1	2	3	4	5	6	7	8	9	10
44	23	34	32		5			8	30	52

⑨

0	1	2	3	4	5	6	7	8	9	10
44	23	34	32	15	5			8	30	52

⑩

0	1	2	3	4	5	6	7	8	9	10
44	23	34	32	15	5	16		8	30	52

Problem 5 (10 points).

5 $h(15) = 15 \% 13 = 2$, occupied

$h'(15) = (15 \% 11) + 1 = 4 + 1 = 5$

$h(15, 1) = (2+5) \% 13 = 7$, occupied

$h(15, 2) = (2 + 2 \cdot 5) \% 13 = 12$, occupied

$h(15, 3) = (2 + 3 \cdot 5) \% 13 = 4$, empty, so

$K = 15$ is stored at $\boxed{\text{index } 4}$

In this process, we check the index using the regular hash function, then since there was a collision, it uses the $h(k, i)$ function, or $(h(k) + i \cdot h'(k)) \bmod N$, until we find an empty cell, which is true for $i = 3$.

Problem 6 (10 points).

I ran the program several times, here are a couple of the outputs:

```
Number of keys = 100000

HashMap average total insert time = 7.4, ArrayList = 1.3, LinkedList = 1.5

HashMap average total search time = 2.4, ArrayList = 5940.9, LinkedList = 17547.2

Number of keys = 100000

HashMap average total insert time = 7.2, ArrayList = 1.2, LinkedList = 1.7

HashMap average total search time = 2.5, ArrayList = 5935.7, LinkedList = 17344.5
```

One thing that I observed almost instantly – mostly due to the fact that I had placed print statements throughout my program to signal progress – was that for the outermost loop (the one that runs 10 times), everything up until the array list search executed instantaneously to the human eye. After that, the linked list search loop also hung for quite a big of time. These were confirmed by the outputs of the program. Both insertion and search for the HashMap ran in constant time, despite the worst possible time being O(n). The insertion loops for both the ArrayList and LinkedList ran in constant time, as well. However, the search loops for those two ran in N time, though the ArrayList was significantly faster (~3x faster) than the LinkedList.

This experiment solidified the notion from lecture/readings that big-Oh analysis truly is the “worst case” and certainly does not happen always. Also, it reaffirmed the efficiency of hash data structures like HashMaps and Hash Tables.