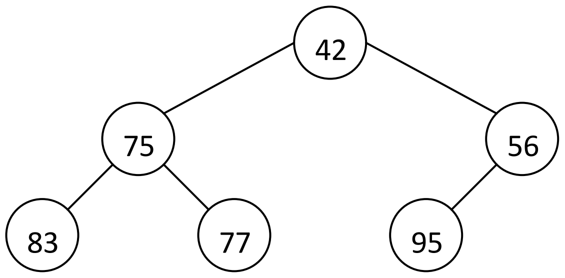
CS526 O2
Homework Assignment 4

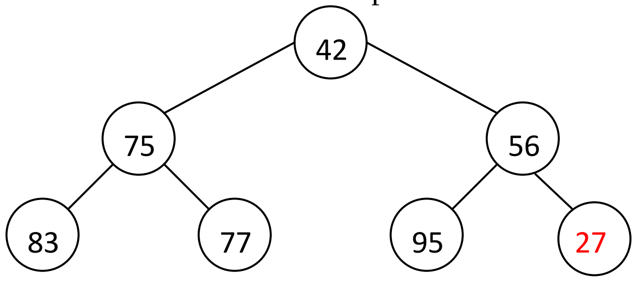
Problem 1 (10 points). Consider the following heap, which shows integer keys in the nodes:



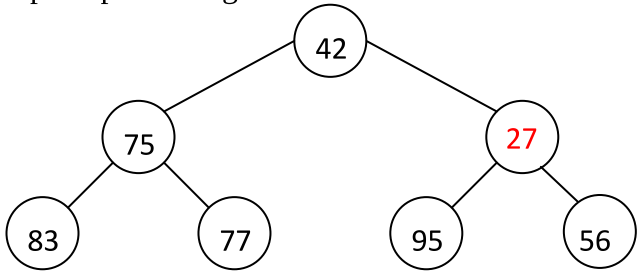
Show the resulting tree if you add an entry with key = 27 to the above tree? You need to describe, step by step, how the resulting tree is generated.

Answer:

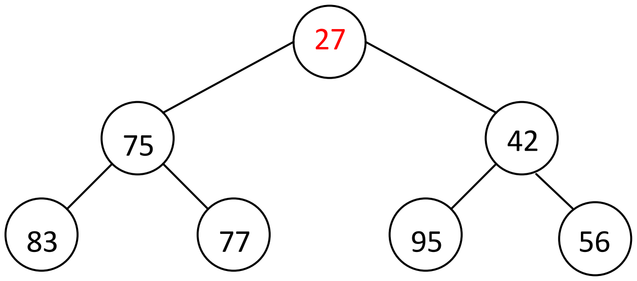
1. 27 is added to last leaf node position.



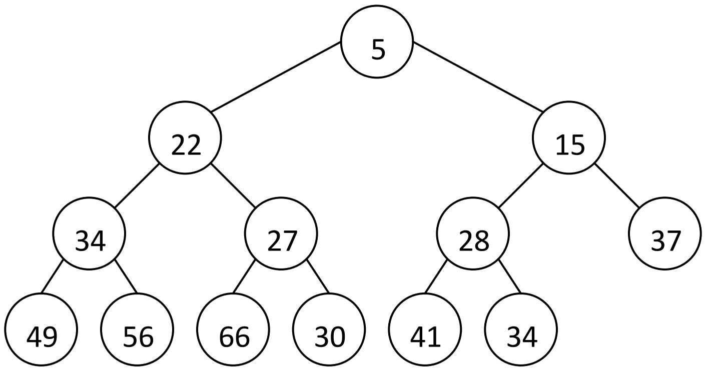
2. Up-heap bubbling with 56.



3. Up-heap bubbling with 42 and reach the root, which concludes with the resulting tree below.



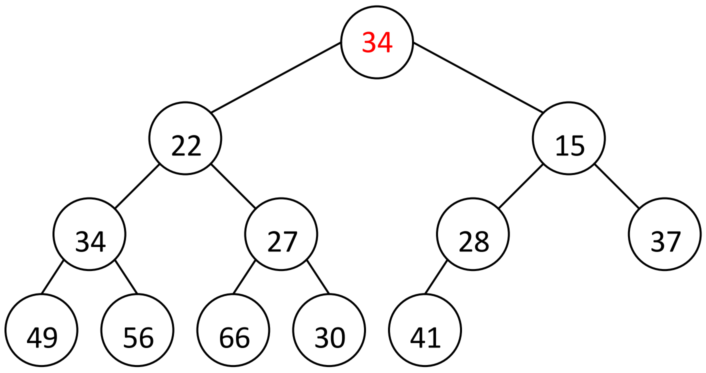
Problem 2 (10 points). Consider the following heap, which shows integer keys in the nodes:



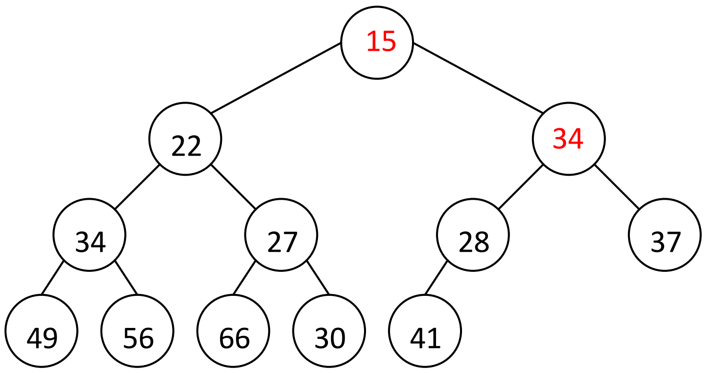
Suppose that you execute the *removeMin*( ) operation on the above tree. Show the resulting tree. You need to describe, step by step, how the resulting tree is generated.

Answer:

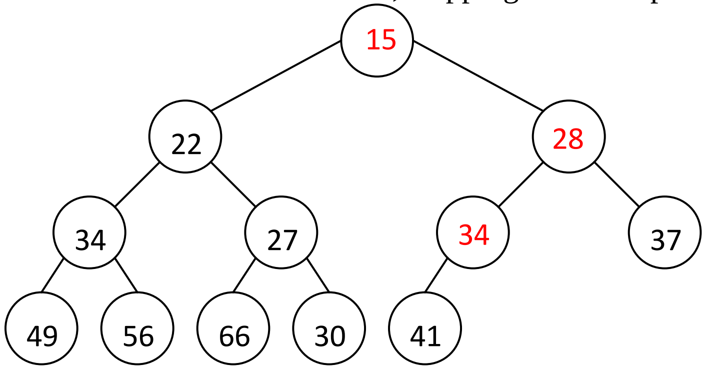
1. Remove root and then swap last node to the root position.



2. Down-heap bubbling with the smaller child.



3. Down-heap bubbling with the smaller child node, which concludes with the resulting tree below as 34 is smaller than 41, stopping down heap bubbling.



Problem 3 (10 points). This problem is about the chaining method we discussed in the class. Consider a hash table of size N = 11. Suppose that you insert the following sequence of keys to an initially empty hash table. Show, step by step, the content of the hash table.

Sequence of keys to be inserted: <5, 8, 44, 23, 12, 20, 35, 32, 14, 16>

Answer:

1. Using Division method to compute the position for every key: $h(k) = k \bmod 11$ (e.g. key 5 mod 11 = position 5), so the sequence of slots is <5, 8, 0, 1, 1, 9, 2, 10, 3, 5>
2. The chaining method will keep an unsorted list in each slot for collision handling.
3. Resulting in following hash table.

0	1	2	3	4	5	6	7	8	9	10
44	23, 12	35	14		5, 16			8	20	32

Problem 4 (10 points). This problem is about linear probing method we discussed in the class. Consider a hash table of size N = 11. Suppose that you insert the following sequence of keys to an initially empty hash table. Show, step by step, the content of the hash table.

Sequence of keys to be inserted: <5, 8, 44, 23, 12, 20, 35, 32, 14, 16>

Answer:

1. Using Division method to compute the position for every key: $h(k) = k \bmod 11$ (e.g. key 5 mod 11 = position 5), so the sequence of slots is <5, 8, 0, 1, 1, 9, 2, 10, 3, 5>
2. The linear probing method will probe the next slot if the current slot is occupied due to collision handling. the sequence of slots is updated as <5, 8, 0, 1, 2, 9, 3, 10, 4, 6>
3. Resulting in following hash table.

0	1	2	3	4	5	6	7	8	9	10
44	23	12	35	14	5	16		8	20	32

Problem 5 (10 points). Suppose that your hash function resolves collisions using open addressing with double hashing, which we discussed in the class. The double hashing method uses two hash functions h and h'.

Assume that the table size $N = 13$, $h(k) = k \bmod 13$, $h'(k) = 1 + (k \bmod 11)$, and the current content of the hash table is:

0	1	2	3	4	5	6	7	8	9	10	11	12
		2	29		18			21	48	15		

If you insert $k = 16$ to this hash table, where will it be placed in the hash table? You must describe, step by step, how the location of the key is determined.

Answer:

1. Inserting k = 16 will compute h(k) = 16 mod 13 = 3, h'(k) = 1 + (16 mod 11) = 6
2. Probe sequence h(k, i) = (h(k) + i*h'(k)) mod N). When i = 1, h(k, i) = (3 + 1 * 6) mod 13 = 9, which is occupied. Continue.
3. When i = 2, h(k, i) = (3 + 2 * 6) mod 13 = 2, which is occupied. Continue.
4. When i = 3, h(k, i) = (3 + 3 * 6) mod 13 = 8, which is occupied. Continue.
5. When i = 4, h(k, i) = (3 + 4 * 6) mod 13 = 1. Found the empty slot, which concludes the resulting hash table below.

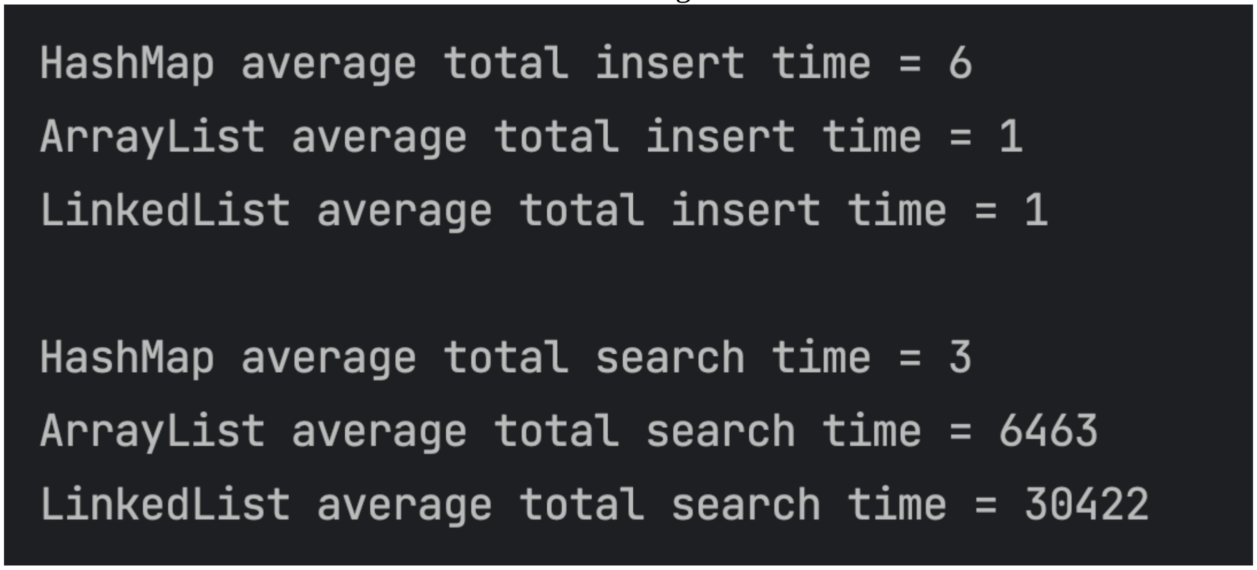
0	1	2	3	4	5	6	7	8	9	10	11	12
	16	2	29		18			21	48	15		

Problem 6

Observation:

I did several tests and found significant differences among HashMap, ArrayList and LinkedList: while HashMap took longer time for insertion than ArrayList and LinkedList, with the latter two took identical time for insertion; HashMap took much shorter time for search than ArrayList, and

LinkedList is much slower than former too in that regard. One test result is shown below:



```
HashMap average total insert time = 6
ArrayList average total insert time = 1
LinkedList average total insert time = 1

HashMap average total search time = 3
ArrayList average total search time = 6463
LinkedList average total search time = 30422
```

Discussion:
This is expected as all 3 data structures have O(1) time complexity for insertion, which means they take a constant amount of time. The small difference for HashMap is due to slightly higher constant amount of time.
For search, HashMap still takes O(1) but both ArrayList and LinkedList take O(n). The difference of search time between ArrayList and LinkedList may be because ArrayList stores all values in a continuous memory location, whereas LinkedList stores each node at a random memory location. Iterating through continuous memory location is more performance efficient than random memory location.