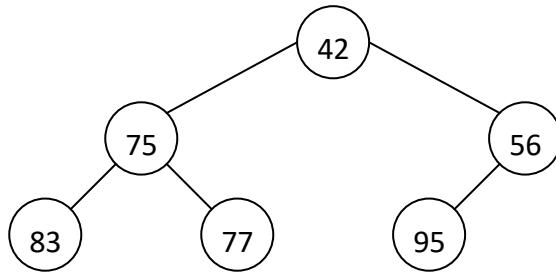


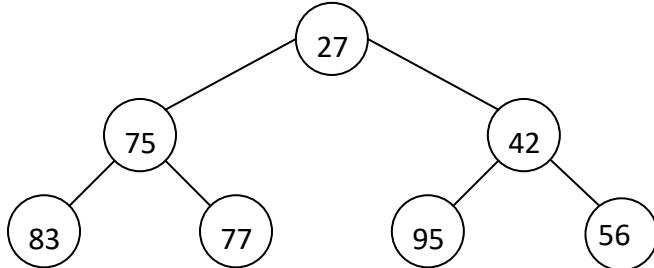
CS526 O2
Homework Assignment 4

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

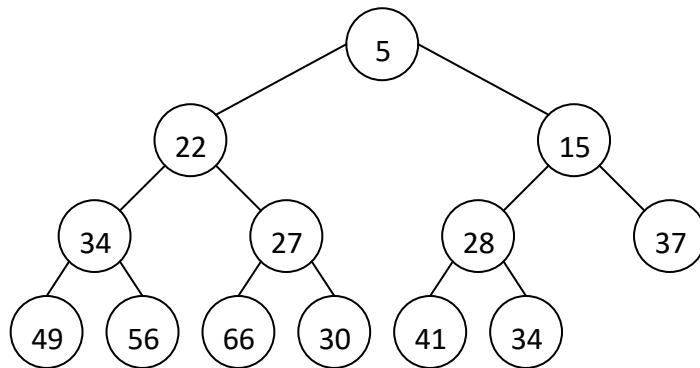


Show the resulting tree if you add an entry with key = 27 to the above tree? You need to describe, step by step, how the resulting tree is generated.

Answer: 27 goes in the next available spot, which is the right child node of 56. Then 27 is compared to 56, because 27 is smaller, they switch. Next 27 is compared to the root 42, because 27 is smaller than 42, they switch and 27 becomes the root.

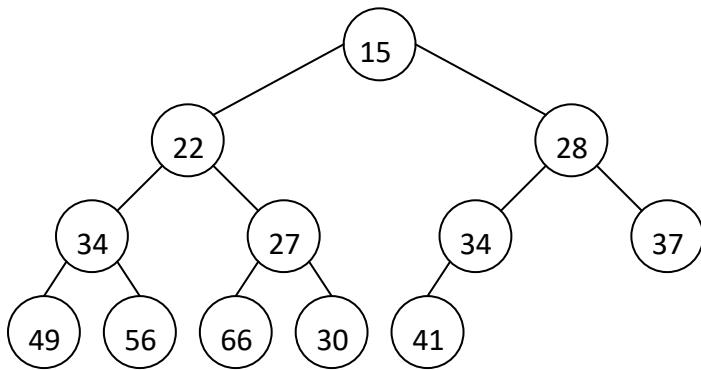


Problem 2 (10 points). Consider the following heap, which shows integer keys in the nodes:



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.

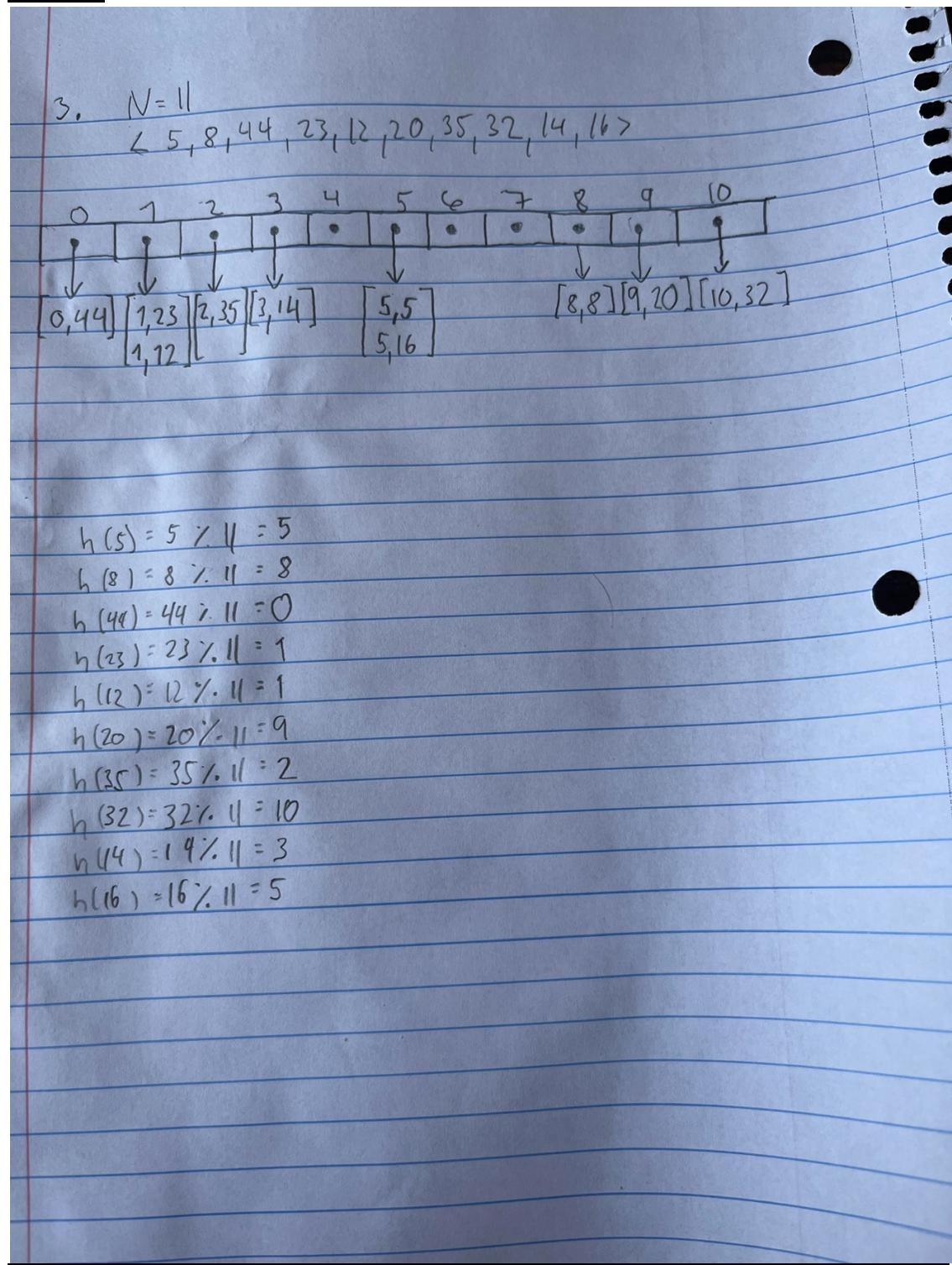
Answer: When the `removeMin()` is executed, 5 gets removed and returned. 34 then takes the place of 5, as the root of the tree. The 34 gets compared with 22 and 15, it's bigger than both but 15 is smaller, so 15 and 34 exchange positions. 34 is then compared to 28 and 37, it is bigger than 28 and smaller than 37, so it takes 28's place. 34 then gets compared to 41, because 41 is bigger, 41 and 34 exchange places.



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: $\langle 5, 8, 44, 23, 12, 20, 35, 32, 14, 16 \rangle$

Answer:



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:

4. $N = 11$, $h = k \bmod 11$

$\langle 5, 8, 44, 23, 12, 20, 35, 32, 14, 16 \rangle$

$$h(5) = 5 \% 11 = 5$$

$$h(8) = 8 \% 11 = 8$$

$$h(44) = 44 \% 11 = 0$$

$$h(23) = 23 \% 11 = 1$$

$$h(12) = 12 \% 11 = 1 \rightarrow \text{moves to } 2$$

$$h(20) = 20 \% 11 = 9$$

$$h(35) = 35 \% 11 = 2 \rightarrow \text{moves to } 3$$

$$h(32) = 32 \% 11 = 10$$

$$h(14) = 14 \% 11 = 3 \rightarrow \text{moves to } 4$$

$$h(16) = 16 \% 11 = 5 \rightarrow \text{moves to } 6$$

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:

5. $N = 13$, $h(k) = k \bmod 13$, $h'(k) = 1 + (k \bmod 11)$

1. $h(16) = 16 \bmod 13 = 3 \rightarrow \text{collision with } 29$

2. $h'(16) = 1 + (16 \bmod 11) = 6$

3. $i=1$, $h(16) + 1 \times h'(16) \bmod 13 = (3+1 \times 6) \bmod 13 = 9 \rightarrow \text{collision with } 48$

4. $i=2$, $(3+2 \times 6) \bmod 13 = 2 \rightarrow \text{collision with } 2$

5. $i=3$, $(3+3 \times 6) \bmod 13 = 8 \rightarrow \text{collision with } 21$

6. $i=4$, $(3+4 \times 6) \bmod 13 = 1$

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

Problem 6 (50 points). The goal of this problem is to give students an opportunity to observe differences among three data structures in Java – Java’s *HashMap*, *ArrayList*, *LinkedList* – in terms of insertion time and search time.

Students are required to write a program that implements the following pseudocode:

```
create a HashMap instance myMap  
create an ArrayList instance myArrayList  
create a LinkedList instanc myLinkedList
```

Repeat the following 10 times and calculate average total insertion time and average total search time for each data structure

```
generate 100,000 distinct random integers in the range [1, 1,000,000] and store them in  
the array of integers insertKeys[ ]
```

```
// begin with empty myHap, myArrayList, and myLinkedList each time
```

```
// Insert keys one at a time but measure only the total time (not individual insert //  
time)  
// Use put method for HashMap, e.g., myMap.put(insertKeys[i], i)  
// Use add method for ArrayList and LinkedList
```

```
insert all keys in insertKeys [ ] into myMap and measure the total insert time  
insert all keys in insertKeys [ ] into myArrayList and measure the total insert time  
insert all keys in insertKeys [ ] into myLinkedList and measure the total insert time
```

```
generate 100,000 distinct random integers in the range [1, 2,000,000] and store them in  
the array searchKeys[ ].
```

```
// Search keys one at a time but measure only total time (not individual search //  
time)  
// Use containsKey method for HashMap  
// Use contains method for ArrayList and Linked List
```

```
search myMap for all keys in searchKeys[ ] and measure the total search time  
myArrayList for all keys in searchKeys[ ] and measure the total search time search  
myLinkedList for all keys in searchKeys[ ] and measure the total search time
```

Print your output on the screen using the following format:

```
Number of keys = 100000
```

```
HashMap average total insert time = xxxxx  
ArrayList average total insert time = xxxxx  
LinkedList average total insert time = xxxxx
```

```
HashMap average total search time = xxxxx  
ArrayList average total search time = xxxxx  
LinkedList average total search time = xxxxx
```

You can generate n random integers between 1 and N in the following way:

```
Random r = new Random(System.currentTimeMillis());  
for i = 0 to n - 1  
    a[i] = r.nextInt(N) + 1
```

When you generate random numbers, it is a good practice to reset the seed. When you first create an instance of the Random class, you can pass a seed as an argument, as shown below:

```
Random r = new Random(System.currentTimeMillis());
```

You can pass any long integer as an argument. The above example uses the current time as a seed.

Later, when you want to generate another sequence of random numbers using the same Random instance, you can reset the seed as follows:

```
r.setSeed(System.currentTimeMillis());
```

You can also use the *Math.random()* method. Refer to a Java tutorial or reference manual on how to use this method.

We cannot accurately measure the execution time of a code segment. However, we can estimate it by measuring an elapsed time, as shown below:

```
long startTime, endTime, elapsedTime; startTime  
= System.currentTimeMillis();  
// code segment  
endTime = System.currentTimeMillis(); elapsedTime  
= endTime - startTime;
```

We can use the *elapsedTime* as an estimate of the execution time of the code segment. Note that if the elapsed time is similar for the three data structures, you may need a more precise measure using `System.nanoTime()` or something similar.

Name the program *Hw4_P6.java*.

Deliverable

You need to submit the following files:

- *Hw4_p1_p5pdf*: This file must include:
- Answers to problems 1 through 5.
- Discussion/observation of Problem 6: This part must include what you observed and learned from this experiment and it must be “substantive.”
- *Hw4_p6.java*
- Other files, if any.

Combine all files into a single archive file and name it *LastName_FirstName_hw4.EXT*, where *EXT* is an appropriate archive file extension, such as *zip* or *rar*.

Grading

Problem 1 through Problem 5:

- For each problem, up to 6 points will be deducted if your answer is wrong.

Problem 6:

- There is no one correct output. As far as your output is consistent with generally expected output, no point will be deducted. Otherwise, up to 20 points will be deducted.
- If your conclusion/observation/discussion is not substantive, points will be deducted up to 5 points.
- If there are no sufficient inline comments in your program, points will be deducted up to 5 points.