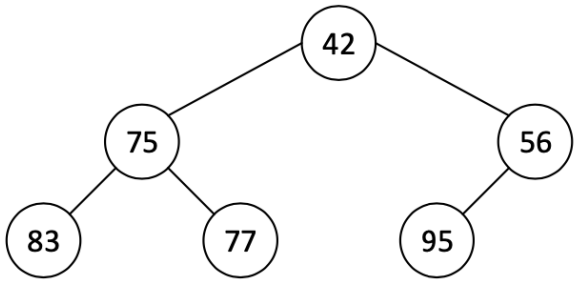
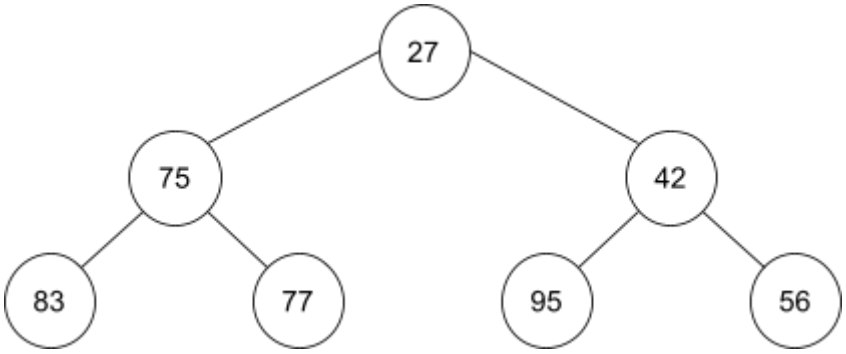
Problem 1 (10 points). Consider the following heap, which shows integer keys in the nodes:

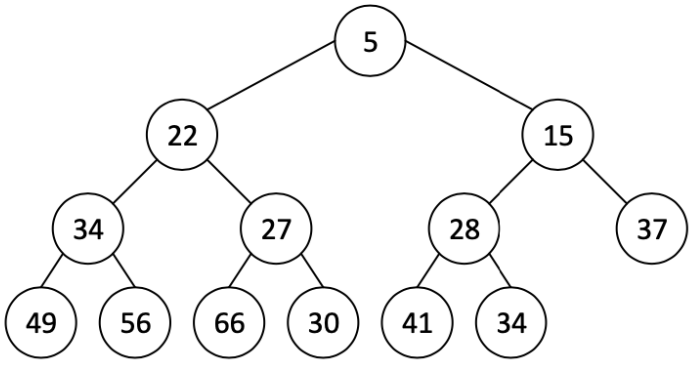


Show the resulting tree if you add an entry with key = 27 to the above tree? You need to describe, step by step, how the resulting tree is generated.

1. First of all, the new entry (27) is added to the end of the heap,
2. Since 27 < 56 , (27) is swapped with its parent (56)
3. Since 27 < 46 , (27) is swapped with its parent (46)
4. (27) reached the root. So, stop. This is the final heap.

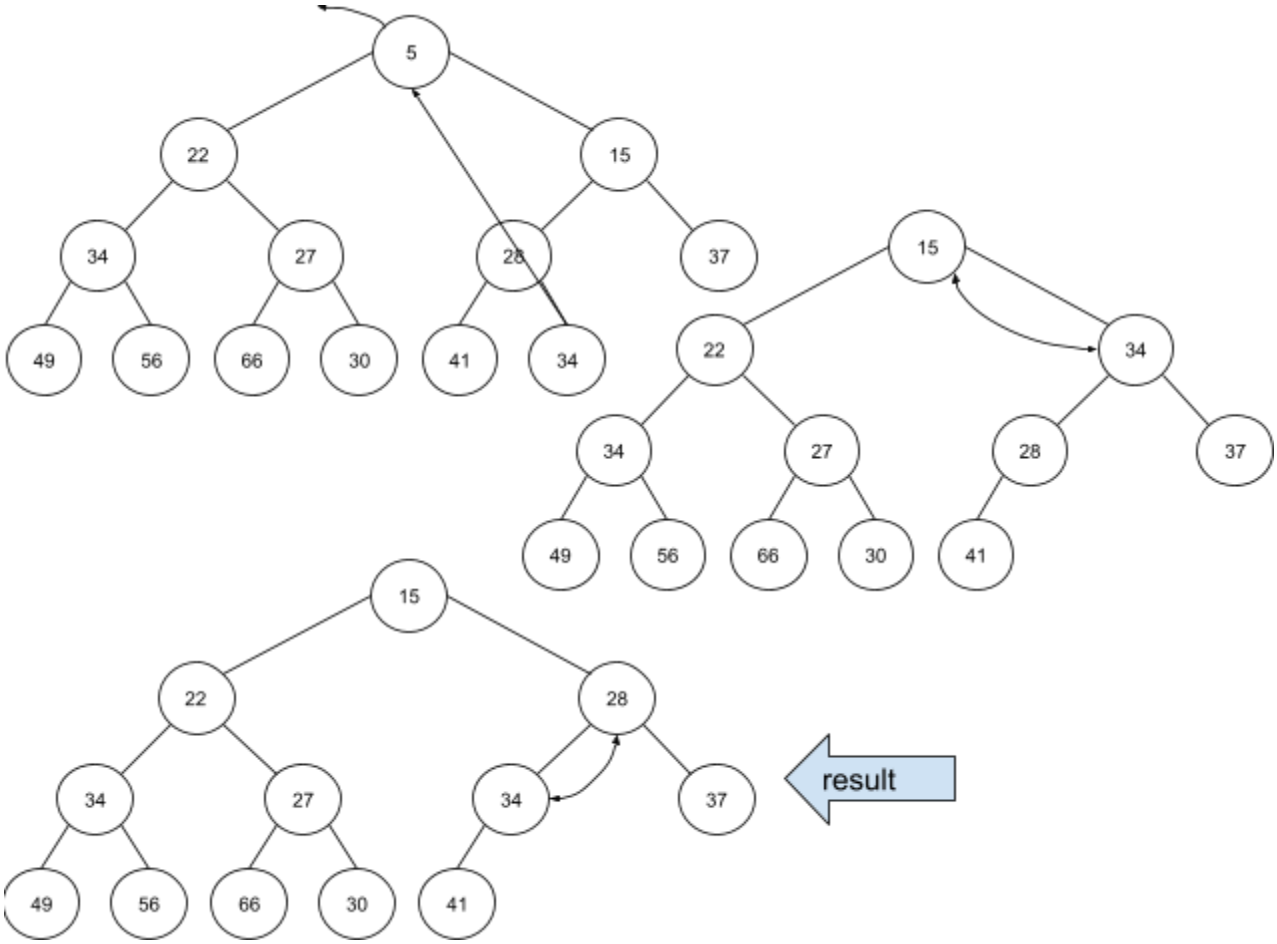


Problem 2 (10 points). Consider the following heap, which shows integer keys in the nodes:



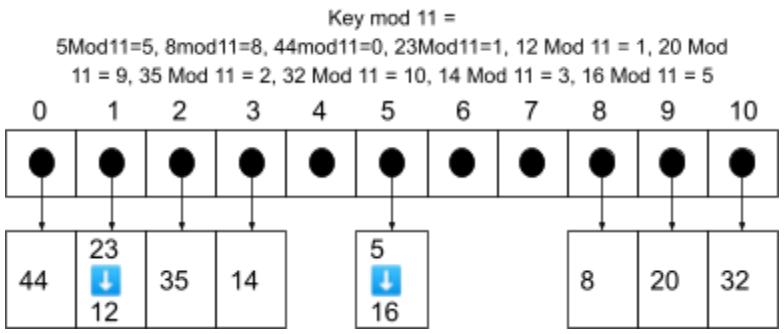
Suppose that you execute the *removeMin*( ) operation on the above tree. Show the resulting tree. You need to describe, step by step, how the resulting tree is generated.

Since an entry with a minimal key is at the root of a heap, The removal is also performed in two steps. In the **first step**, the root entry is removed, and the last node in the heap is moved up to the root position. Then, in the **second step,** the new heap with the new root is reorganized to maintain the heap-order property.



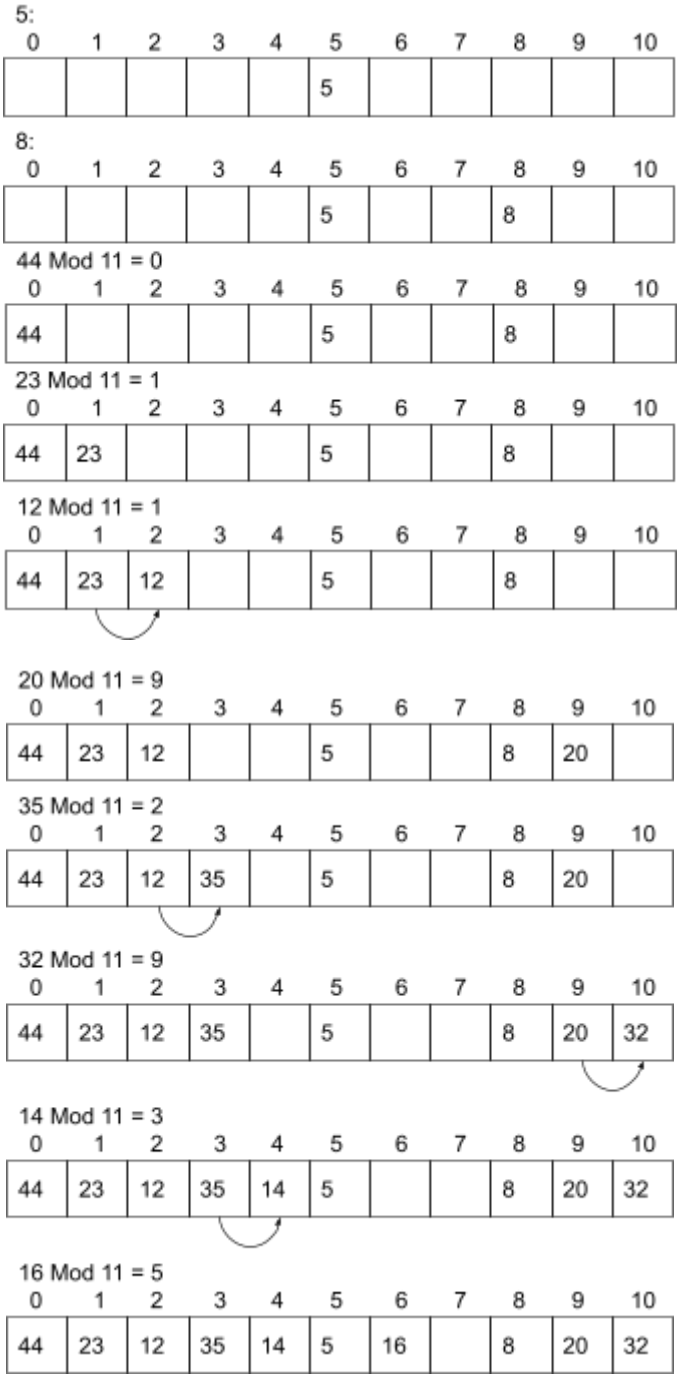
Problem 3 (10 points). This problem is about the chaining method we discussed in the class. Consider a hash table of size N = 11. Suppose that you insert the following sequence of keys to an initially empty hash table. Show, step by step, the content of the hash table.

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

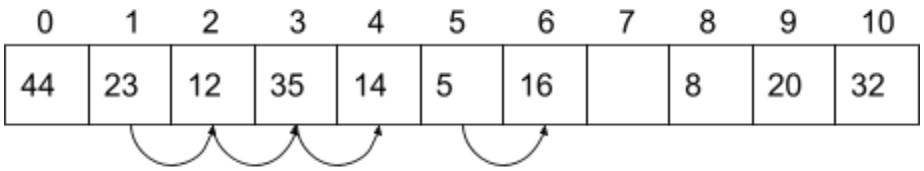


Problem 4 (10 points). This problem is about linear probing method we discussed in the class. Consider a hash table of size N = 11. Suppose that you insert the following sequence of keys to an initially empty hash table. Show, step by step, the content of the hash table.

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



Final result:



<5, 8, 44, 23, 12, 20, 35, 32, 14, 16>

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 mod 13, h'(k) = 1 + (k mod 11), and the current content of the hash table is:

0	1	2	3	4	5	6	7	8	9	10	11	12
	16	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.

1. Insert 16, h(16) = 3 and h'(k) = 1+ 5 = 6
2. h(16) = 3, occupied
3. i = 1, 3 + 6 = 9, occupied
4. i = 2, 3 + 2 * 6 = 15, occupied
5. i = 3, 3 + 3 * 6 = 21, occupied
6. i = 4, 3 + 4 * 6 = 27, empty, store 16 at position 1. 27 mod 13 = 1

Discussion/observation of Problem 6:
For the insert time I observed the arraylist usually takes the shortest time to insert, then its linked list, two lists have a very similar interesting time. Hash map usually takes longer to insert,

When they are dealing with keys two lists will have 100000 size when they are dealing with 100000 random numbered keys, but usually a hashmap will have less element than 100000 because when there is a duplication it won't be put again.

When we are dealing with searching hashmap outperforms lists, it takes a shortest time to search. The array list is much longer than a hashmap but still takes 3 times less time to search than the linked list. Linked list took the longest time.

Insert time = hashmap > array list > linked list
search time = hashmap < array list < linked list

```
Number of keys = 100000

HashMap average total insert time = 9482711ns
ArrayList average total insert time = 2317512ns
LinkedList average total insert time = 2626928ns

HashMap average total search time = 4644958ns
ArrayList average total search time = 6984995518ns
LinkedList average total search time = 20156649640ns

Process finished with exit code 0
```