CSCI 200

HW #2: Trees, heaps, heapsort, mergesort

Directions: *Complete your work on a separate sheet of paper. Submit the physical copy of your work at the beginning of class on the specified due date. Show your work. You may work in groups of up to 3 students provided that all students participate in each question. Provide a short preliminary explanation of how an algorithm works before running an algorithm or presenting a formal algorithm description, and use examples or diagrams if they are needed to make your presentation clear.*

1. Draw a single binary tree T such that all of the following properties hold for T:
   - each node of T stores a single character,
   - a preorder traversal of T yields MYSTERY, and
   - an inorder traversal of T yields TSEYRMY

2. Give the **pseudocode** for an algorithm called ComputeDepth(T) which runs in total $O(n)$-time and computes the depth of every node of a tree T, where n is the number of nodes of T. Assume every node has an attribute `depth` which has an initially null value, i.e., the purpose of the ComputeDepth algorithm is to appropriately set the depth attribute of every node with its correct depth value. *Hint: After you have set a particular node's depth attribute, you may access it later to help set another node's depth attribute.*

3. Use the table below to convert a character key to an integer. For each of the following questions, give the contents of the hash table that results when the following keys are inserted into an initially empty 13-item hash table, which has indices 0 to 12. The keys to insert, in this order, are: $(E_1, A, S_1, Y, Q, U, E_2, S_2, T, I, O, N)$. Use $h(k) = k \mod 13$ for the hash function for the k-th letter of the alphabet (see above table for converting letter keys to integer values). Use the provided method to resolve collisions. Note that the elements to insert into the hashtable are letters. not numbers.

Letter	A	B	C	D	E	F	G	H	I	J	K	L	M
Key	0	1	2	3	4	5	6	7	8	9	10	11	12
Letter	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Key	13	14	15	16	17	18	19	20	21	22	23	24	25

   (a) Illustrate the contents of the hashtable when collisions are resolved using **chaining.**

   (b) Illustrate the contents of the hashtable when collisions are resolved using **linear probing.**

   (c) Illustrate the contents of the hashtable when collisions are resolved using **double hashing**. Let the secondary hash function be $h'(k) = 1 + (k \mod 11)$.

4. Consider the following arrays. Indicate (yes/no) whether they are or are not a max-heap.

   (a) $[23, 17, 14, 6, 13, 10, 1, 5, 7, 12]$

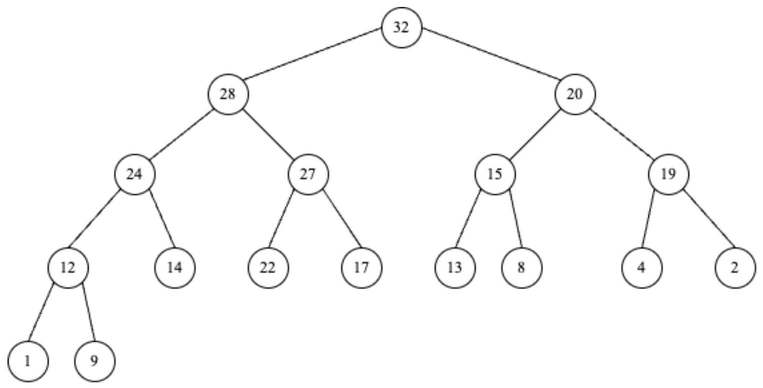
   (b) $[50, 25, 28, 8, 3, 17, 14, 6, 5, 1, 2]$

   (c) $[4, 8, 10, 9, 12, 16, 18, 22, 40]$

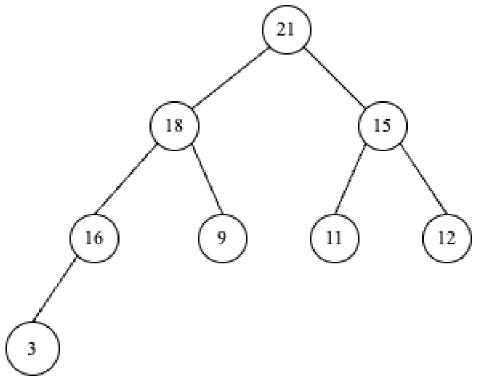
   (d) $[15, 14, 13, 10, 12, 9, 8, 5, 7, 3, 2]$

   (e) $[12, 5, 13, 4, 2, 6, 10, 1, 3]$

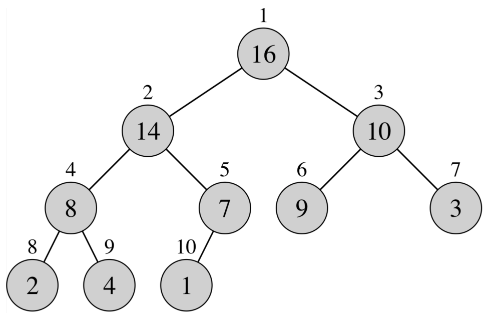
5. Consider the following max-heap. Insert in the following order: 10, 20, 30, 32. Show the resulting heap (as a tree) as each item is inserted.



6. Consider the following max-heap. Perform the RemoveMax operation three consecutive times. Show the resulting heap (as a tree) after each operation.



7. Let T be a (max) heap storing n keys. Give the **pseudocode** for an efficient algorithm for printing all the keys in T that are greater than or equal to a given query key x (which is not necessarily in T). You can assume the existence of a $O(1)$-time *print(key)* function. For example, given the heap below and query key $x = 6$, the algorithm should report $16, 14, 10, 8, 7, 9$. Note that the keys do not need to be reported in sorted order. **Analyze the run time of your algorithm**. It should run in $O(k)$ time, where k is the number of keys reported.



8. Illustrate the execution tree of mergesort on the following array: $[1, 5, 8, 10, 12, 3, 6, 13, 4, 2, 7, 9, 15, 10, 24, 28, 11]$.

9. Using the last element as pivot, illustrate the execution tree of quicksort on the following array: $[5, 8, 2, 4, 3, 7, 6, 9, 10]$.

10. Suppose we are given a sequence S of n elements, each of which is colored red or blue. Assuming S is represented by an array, give a linear-time **in-place** algorithm for ordering S so that all the blue elements are listed before all the red elements. What is the running time of your method?