EECE.3220: Data Structures

Fall 2019

Exam 3

December 21, 2019

Name: ____

For this exam, you may use two 8.5" x 11" double-sided pages of notes. All electronic devices (e.g., calculators, cell phones) are prohibited. If you have a cell phone, please turn off your ringer prior to the start of the exam to avoid distracting other students.

The exam contains 3 sections for a total of 100 points, as well as a 10-point extra credit section. Please answer all questions in the spaces provided. If you need additional space, use the back of the page on which the question is written and clearly indicate that you have done so.

Please read each question carefully before you answer, especially the multiple-choice questions. Question 4c has exactly one correct answer, while the remaining multiple-choice questions (4a, 4b, 4d, 4e, 4f) may have more than one correct answer.

You will be provided with a sheet containing class definitions that are necessary for Questions 2b and 3b. You do not have to submit this sheet with the exam.

You will have three hours to complete this exam.

S1: Recursion	/ 10
S2: Binary search trees	/ 44
S3: Heaps	/ 22
S4: Multiple choice (heapsort, priority queues, hash tables, C++ containers)	/ 24
TOTAL SCORE	/ 100
S5: EXTRA CREDIT	/ 10

1. (10 points) ***Recursion***

For the short program shown below, list the output exactly as it will appear on the screen. Be sure to clearly indicate spaces between characters when necessary.

You may use the available space to show your work as well as the output; just be sure to clearly mark where you show the output so that I can easily recognize your final answer.

```
int f1(int v1, int v2) {
   cout << "v1 = " << v1 << ", v2 = " << v2 << '\n';
   if (v1 == 0) {
      cout << "Returning 1\n";
      return 1;
   }
   else if (v2 == 0) {
      cout << "Returning 2\n";
      return 2;
   }
   else if (v1 > v2) {
      cout << "v1 > v2\n";
      return 1 + f1(v1 / 2, v2 / 2);
   }
   else {
      cout << "v1 <= v2\n";
      return 2 + f1(v2 / 3, v1 / 3);
   }
}


int main() {
   cout << "Final return value = " << f1(7, 12) << '\n';
   return 0;
}
```

2. (44 points) ***Binary search trees***

a. (12 points) Show the binary search tree built by inserting the integers below in the order shown (beginning with 10). Assume the tree is empty to start.

Integers: 10, 4, 8, 15, 25, 12, 6, 7

2 (continued)

b. (12 points) To print a binary search tree's contents, visit all the nodes and print their contents. The order in which the nodes are visited determines the order of the output values.

In this problem, the top-level function for the tree object (*the BST class defined on the extra sheet*) simply calls a function that takes a node pointer as an argument. That function does the work of visiting all the nodes and printing each of their contents.

The top-level print function is shown. You must write the `printtree()` function, which takes two arguments: an output stream reference, `os`, and a pointer to the root of a given subtree, `st`. `printtree()` prints the data in the node `st` points to and visits its left and right children to do the same.

I strongly suggest writing this function recursively, as an iterative solution is significantly more difficult to write without parent pointers in the tree.

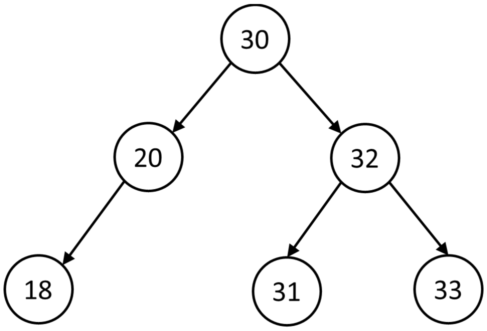
For full credit, write `printtree()` so all tree data are printed in order from lowest to highest. For example, given the data in Question 2a, the function prints: `4 6 7 8 10 12 15 25`

```
void BST::print(ostream &os) {
   printtree(os, root);
   os << "\n";
}

void BST::printtree(ostream &os, BNode *st) {

















}
```

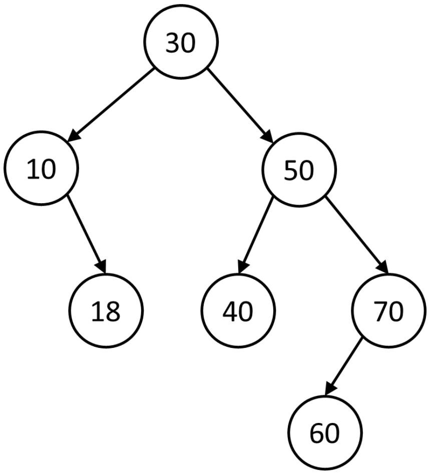
2 (continued)

c. (8 points) For each part of this problem, you are given an AVL tree and an operation (add or remove a specific value) to perform. Show the modified tree after the specified operation, which will require at least one rotation to maintain the properties of an AVL tree.

i. Add 19 to the tree below

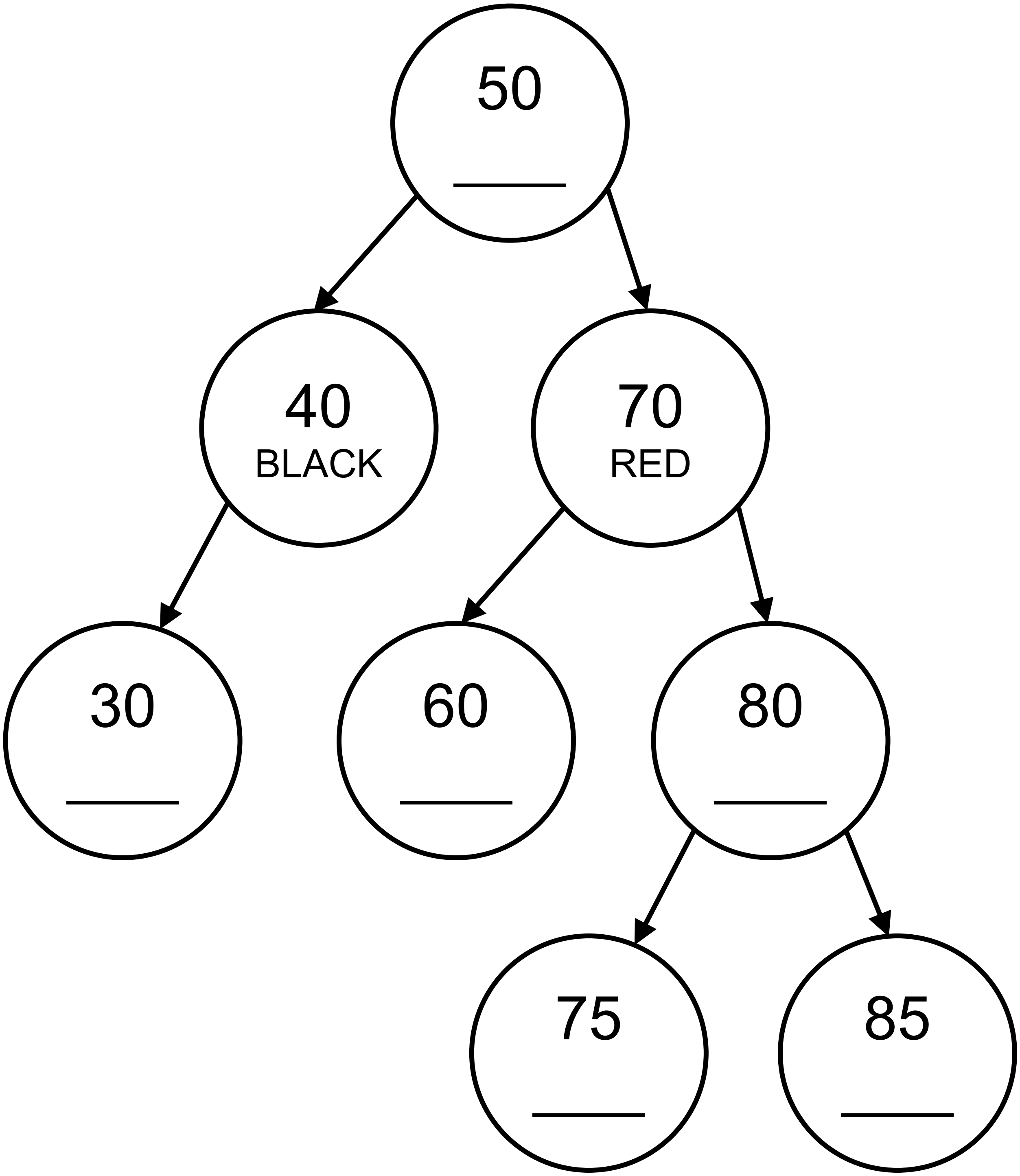


ii. Remove 30 from the tree below



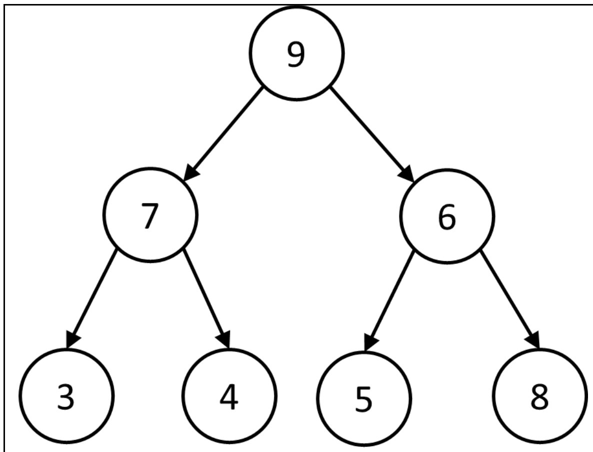
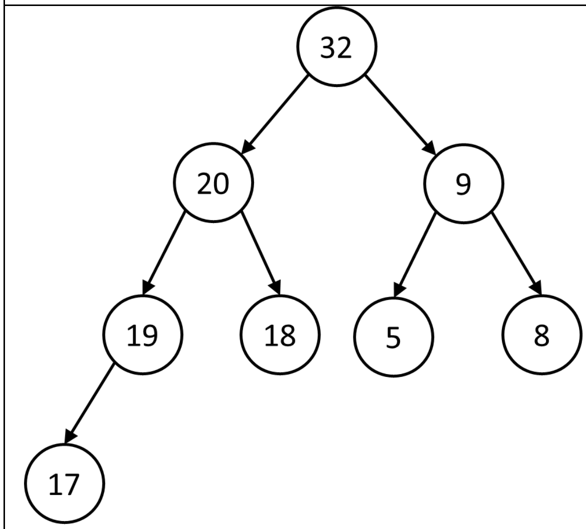
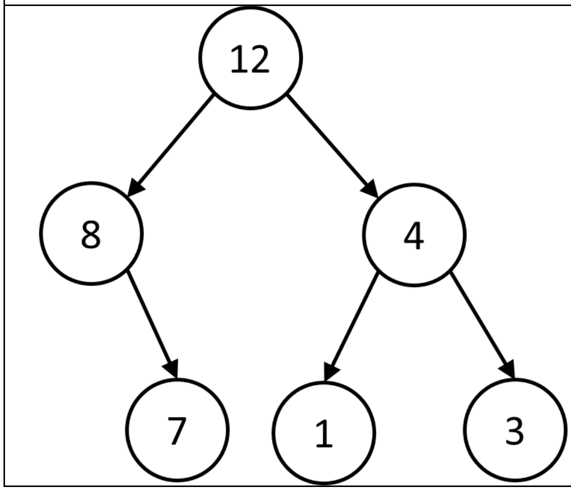
2 (continued)

d. (12 points) The tree below represents a red-black tree, but the colors of only two nodes are shown. The node storing the value 40 is black, while the node storing 70 is red. Use the rules for red-black trees we discussed in class to determine the correct color for each remaining node and write that color on the blank line in the node.



3. (22 points) ***Heaps***

a. (10 points) For each of the binary trees below, indicate (i) whether it is a max-heap (by circling "Yes" or "No"), and (ii) if not, which heap property it violates.

	(i) Is this tree a max-heap? Yes No (ii) If no, which heap property does it violate?
	(i) Is this tree a max-heap? Yes No (ii) If no, which heap property does it violate?
	(i) Is this tree a max-heap? Yes No (ii) If no, which heap property does it violate?

3 (continued)

b. (12 points) For some heap operations, the number of non-leaf nodes in the heap is a useful piece of information. Use the blank space below to write a function, `countNonLeaves()`, that returns the number of non-leaf nodes in a `Heap` object. Note that:

- The number of non-leaf nodes is a function of the heap size.
- The number of nodes in each complete level is a power of 2.
    - The top level (root) has 1 node, the 2nd level has 2, the 3rd level has 4, and so on.
    - So, a heap with 2 levels has 1 non-leaf node (since the bottom level is composed solely of leaf nodes); a heap with 3 levels has 1 + 2 = 3 non-leaf nodes, and so on.

This function uses the `Heap` class definition shown on the extra sheet provided with the exam. You should assume that this `Heap` class is implemented similarly to the heaps discussed in lecture. Specifically:

- The root of the heap is stored in the first array location (`heaparr[0]`).
- No array locations are left empty for swap space.

```
unsigned Heap::countNonLeaves() {
































}
```

4. (24 points, 4 points each) ***Multiple choice (heapsort, priority queues, hash tables, C++ containers)***

a. Which of the following statements about heapsort are true? **Circle ALL correct answers—there may be more than one.**

   i. Heapsort has a worst-case execution time of O(n $\log_2$(n)).

   ii. The heapsort algorithm can only be used on arrays whose contents represent a heap before the sorting begins.

   iii. After running heapsort on an array to sort the array from lowest to highest value, the array's contents represent a min-heap.

   iv. The heapify algorithm, which is part of heapsort, has a worst-case execution time of O(n $\log_2$(n))

b. Which of the following statements about priority queues are true? **Circle ALL correct answers—there may be more than one.**

   i. A priority queue must be implemented using a heap.

   ii. If a priority queue is implemented using a heap, all priority queue operations can be written in terms of heap operations.

   iii. The contents of a priority queue must be ordered from highest priority to lowest priority.

   iv. Storing data in a priority queue allows you to access the highest-priority item in the queue in a constant amount of time (O(1)).

4 (continued)

c. Say you have a hash table containing 10 buckets, and you want to store a total of 20 items in the table. What is the minimum and maximum possible number of collisions (instances in which a new item maps to the same bucket as another item) in this scenario? **This question has exactly one correct answer.**

  i. Minimum = 0 collisions; maximum = 10 collisions

  ii. Minimum = 0 collisions; maximum = 19 collisions

  iii. Minimum = 0 collsions; maximum = 20 collisions

  iv. Minimum = 10 collisions; maximum = 19 collisions

  v. Minimum = 10 collisions; maximum = 20 collisions

d. Which of the following statements about the C++ sequence containers and sequence container adapters are true? **Circle ALL correct answers—there may be more than one.**

  i. All sequence containers must be implemented as linked data structures.

  ii. The key difference between the `array` and `vector` types is that a `vector` can be resized, while an `array` can not.

  iii. The basic `list` can be traversed in either direction, while the `forward_list` can only be traversed from first to last element.

  iv. A sequence container adapter always stores its data in a `vector`, then includes rules about how the data in the vector is accessed to create a stack, queue, or priority queue.

4 (continued)

e. Which of the following statements about the C++ associative containers are true? **Circle ALL correct answers—there may be more than one.**

i. Say you have the same data set stored in two different associative containers—one an ordered container (for example, set or map), and the other one an unordered version of the same type of container. Searching for an item in the ordered container will always be faster than searching for the same item in the unordered container.

ii. Ordered associative containers are typically implemented using binary search trees to ensure worst-case search times of O(log n).

iii. In a map, the data being stored represents the key value used to order that data. In a set, the data being stored is associated with a separate key used to order the data.

iv. Unordered associative containers are often implemented using hash tables, where the key is fed into a hash function to determine the bucket used to store the data.

f. Circle one (or more) of the choices below that you feel best "answers" this "question."

i. "Thanks for the free points."

ii. "This is the best final exam I've taken today."

iii. "At least we're getting this out of the way early in the day."

iv. "I would have preferred to take this on Tuesday."

v. None of the above.

5. (10 points) ***EXTRA CREDIT***

You are given the postfix expression shown below:

```
    5 3 + 6 8 - * 1 3 + /
```

Complete the table below to show how a stack can be used to evaluate this expression, asyour Program 3 solution would do. I've filled in all tokens; you show what operation is performed for each token (it's pushed on the stack or it causes data to be popped and a value to be computed) and the stack state once that operation is done. I've completed the first row.

Note that a postfix expression of the form `v1 v2 op` is evaluated as `v1 op v2`. So, for example, `8 2 -` represents the operation `8 - 2`.

Token	**Operation** (if data popped & result computed, list values removed and result of operation)	**Stack state** (show state as list of values with top of stack listed first)
5	5 pushed on stack	5
3		
+		
6		
8		
-		
*		
1		
3		
+		
/		