EECE.3220: Data Structures

Spring 2017

Exam 3
May 5, 2017

Name: _____

For this exam, you may use only one 8.5" x 11" double-sided page 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 4 questions for a total of 100 points, and 1 extra credit question worth 10 points. Please answer the 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. In particular, note that:

- Several questions refer to class or function definitions listed on the extra sheets (labeled "EECE.3220 Spring 2017: Exam 3 Class Definitions") provided with the exam.
- Questions 1a and 2a are multiple choice questions that may have more than one correct answer. If a question has more than one correct answer, you will receive partial credit for each correct answer you choose.
- Question 4a is a multiple choice question that only has one correct answer.

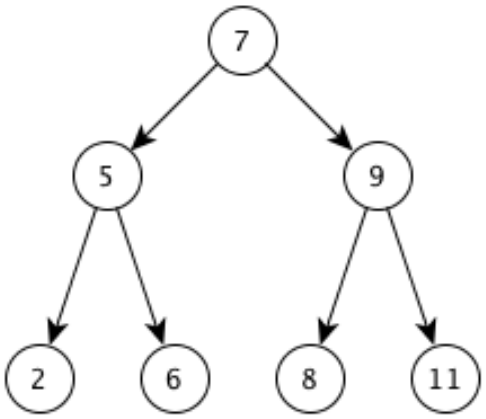
You will have three hours to complete this exam.

Q1: Binary search trees	/ 28
Q2: Heaps and priority queues	/ 27
Q3: Sorting	/ 20
Q4: Hash tables	/ 25
TOTAL SCORE	/ 100
EXTRA CREDIT	/ 10

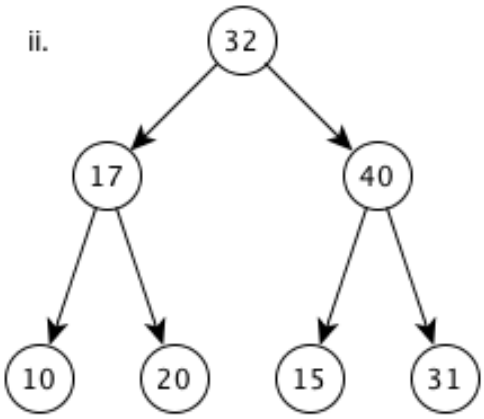
1. (28 points) ***Binary search trees***

a. (6 points) Which of the trees below are binary search trees? **Circle ALL correct answers—there may be more than one.**

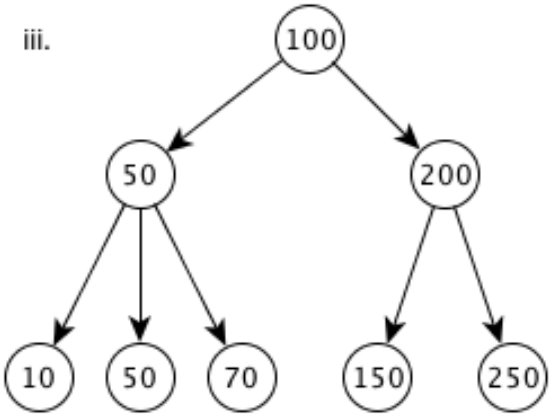
i.



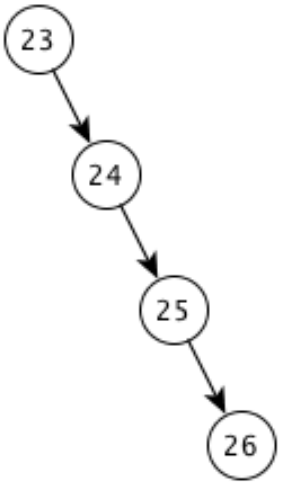
ii.



iii.



iv.



1 (continued)

b. (10 points) Given the program below, show (i) the contents of the BST this program generates (drawn like the trees shown in Question 1a), and (ii) the program output. Assume:

- This program uses the BST class definition on the extra sheet provided with the exam. That sheet also contains the definition of the `print()` function (and its helper function `printtree()`).
- The `add()` function adds a node to the given BST object as a leaf node in the appropriate location within the tree. You should not need a detailed `add()` definition to understand how nodes are added to a binary search tree.

```
int main() {
    BST test;

    test.add(8);
    test.add(3);
    test.add(5);
    test.add(1);
    test.add(9);
    test.add(13);
    test.add(11);

    cout << "Tree contents: ";
    test.print(cout);

    return 0;
}
```

1 (continued)

c. (12 points) The method for deleting a node from a binary search tree depends on the number of children that node has. The trickiest case is deleting a node, N, with 2 children, which requires the following steps:

   A. Identify the immediate successor of N and copy the contents of that node to N.
   B. Delete the immediate successor.

Complete the partial `delete()` function below. Your code should check if N has two children and, if so, identify its immediate successor and copy the contents of the successor to N. You do not have to write any other part of the `delete()` function.

This problem uses the BST class definition shown on the extra sheet provided with the exam.

```
void BST::delete(int v) {
   BNode *N;    // Pointer to node to be deleted
   BNode *S;    // Pointer to immediate successor of N

   ...      // Code to find node with value v in BST

   // START WRITING CODE BELOW THIS COMMENT
   // N points to node to delete; check if N has 2 children

   if (_____________________________________________________) {

      // Write code to set S = address of N's immediate successor









      // Copy data from immediate successor to node N points to




   } // End of if statement
   // STOP WRITING CODE AT THIS COMMENT

   ...      // Code for remainder of delete function
}
```

2. (27 points) ***Heaps and priority queues***

a. (6 points) Which of the following statements about a priority queue are true? **Circle all true statements—there may be more than one correct answer.**

i. All valid priority queue implementations require that all items in the priority queue are sorted by priority.

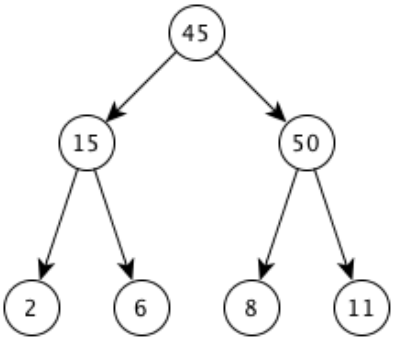
ii. If a priority queue is implemented using a heap, finding the highest priority item in the queue takes O(1) time.

iii. If a priority queue is implemented using a heap, removing the highest priority item in the queue takes O(1) time.

iv. Priority queues may be implemented using data structures other than a heap.

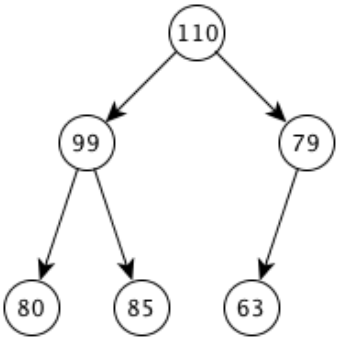
2 (continued)

b. (9 points) For each of the binary trees below, indicate (i) whether it is a heap (by circling "Yes" or "No"), and (ii) if not, which heap property it violates.



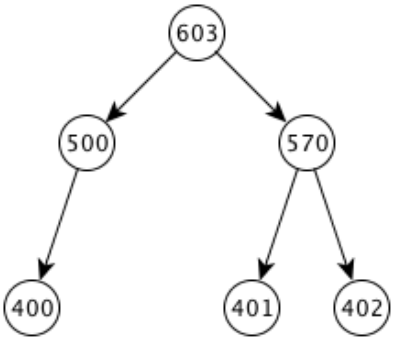
(i) Is this tree a heap? Yes No

(ii) If no, which heap property does it violate?



(i) Is this tree a heap? Yes No

(ii) If no, which heap property does it violate?



(i) Is this tree a heap? Yes No

(ii) If no, which heap property does it violate?

2 (continued)

c. (12 points) Inserting a new value into a heap involves two steps:

   A. Place the new value at the end of the array.
   B. "Percolate up" the new value by exchanging it with its parent until the new value is either (i) less than or equal to its parent, or (ii) moved to the root of the heap.

Use the space below to write the `percolate_up()` function as described above. The function takes a single argument, `pos`, that is the initial position of the value to be moved up the heap.

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 first entry in the array that stores the actual data (`heaparr[0]`) is left empty and can be used as swap space if necessary.
- The root of the heap is stored in the second array location (`heaparr[1]`).

```
void Heap::percolate_up(unsigned pos) {




}
```

3. (20 points) ***Sorting***

Given the list of values below, show **all** of the steps required to sort the list using **either** quicksort or merge sort. **You do not have to show the results of both sorting algorithms.**

The following page is also blank to allow you extra space to show your work if necessary. The list of values to be sorted is reprinted on that page for your reference.

List of values to be sorted: {7, 9, 3, 1, 5, 10, 2, 12, 6, 4}

3 (continued)
Additional space to solve problem 3 (sort the list below using **either** quicksort or merge sort)

List of values to be sorted: {7, 9, 3, 1, 5, 10, 2, 12, 6, 4}

4. (25 points) ***Hash tables***

a. (5 points) You want to store data with the following key values in a hash table with 8 locations: {3, 6, 8, 12, 15, 16, 20}

If `k` refers to a key value and `L` refers to a location in the hash table ($0 \leq L \leq 7$), which of the following hash functions (some of which are written in pseudocode) causes the fewest collisions for this data set?

i.
```
L = k % 8;
```

ii.
```
s = 0.25 * k;
x = fractional part of s;      // i.e, if s = 5.25, x = 0.25
L = integer part of (x * 8);   // i.e., if x*8 = 1.75, L = 1
```

iii.
```
M = maximum key value;         // In this data set, M = 20
L = k / M;
```

iv. All three hash functions (i-iii) cause the same number of collisions for this data set

4 (continued)

b. (10 points) Show the output of the program below, which uses the HashTable class definition and function definitions shown on the extra sheet provided with the exam.

```
int main() {
    HashTable <int>HT;

    HT.add(3);
    HT.add(13);
    HT.add(1);
    HT.add(34);
    HT.add(29);
    HT.add(11);
    HT.add(56);
    HT.add(22);

    HT.print(cout);

    return 0;
}
```

4 (continued)

c. (10 points) Assume you have the same program as in Question 4b (reproduced below), but your HashTable class implementation (and the associated functions) has changed as follows:

- The table uses chaining (placing multiple items that map to the same table index in a linked list) to resolve collisions
- Rather than the two arrays shown in the original `HashTable` class definition, the hash table with chaining uses a single array, `Node *tab[5];` where each entry in `tab[]` is a pointer to the head of a linked list. *(Note that this array only holds 5 pointers.)*
- Given a value, `v`, to be stored in the table, the hash function is simply `v % 5`.

How many values will be in the <u>longest</u> linked list in your hash table, and what will those values be? <u>Show all work for full credit.</u>

```
int main() {
    HashTable <int>HT;

    HT.add(3);
    HT.add(13);
    HT.add(1);
    HT.add(34);
    HT.add(29);
    HT.add(11);
    HT.add(56);
    HT.add(22);

    HT.print(cout);

    return 0;
}
```

EXTRA CREDIT

a. (5 points) In the space below, write the function `removeStar()`, which works as follows: given an input string, `s`, which contains a single `'*'` character, return a string with the `'*'` removed.

For example, `removeStar("*one") = "one"`, `removeStar("t*wo") = "two"`, and `removeStar("three*") = "three"`

```
string removeStar(string s) {



}
```

EXTRA CREDIT

b. (5 points) In a game of war, the two players start with the decks shown below. Each card is listed as in Program 5, with a single character each for the rank and suit (rank is shown first).

Player A

```
4c Th 8c 7c 8s Jd 5h Qc 2c 5d 6c 4h Kd
3s Td 2s 7h Ks Qd 9d Ah Ts Js Tc Jc 7d
```

Player B

```
4s Ac 5c 6s 8h 4d 3c Qh 2d 6h 3h 2h Kh
Ad Kc 3d 7s 9c Jh 5s As 9h 6d 8d Qs 9s
```

Which player wins? For full credit, explain (briefly) how the game plays out.