FINAL EXAM - May 2016

CSC 212 01: Programming with Data Structures

Instructor: Sara Sheehan

- This is a self-scheduled exam to be completed during one of the final exam periods.
- Please write all your work on these pages (DO NOT USE THE BLUE BOOK).
- It is closed notes, closed Internet, and closed technology, but you may use two "cheat sheets".
- Your cheat sheets must be hand-written, created by you, 8.5 × 11 inches, and double-sided, so 4 sides total.
- Do not discuss the exam with other students and respect the honor code of doing your own work.
- If you are unable to make progress on any part of the exam, tell me what you tried; describe your thought process.
- If something seems unclear, state how you interpreted the problem and try to make progress based on that.
- For ease of writing, I'll encourage using *uppercase, single-letter* variable names.

Name	Solutions (sketches)

Part 1	✓ /20
Part 2	✓ /25
Part 3	✓ /10
Part 4	✓ /25
Part 5	✓ /20
Total	/100

Part 1: Warmup and Vocab

Answer the following short questions. Justifications are not required, but may be given if you are unsure or want to explain your answer.

2 (a) To *declare* a new variable, the programmer *must* provide two pieces of information. These two pieces of information are the variable's __type__ and __name__.

6 (b) For the following data structures, what are the runtimes of the following operations, in terms of the number of elements n? For insertion, assume that we are already have access to the desired position.

	Getting an element	Inserting an element
Array	$O(1)$	$O(n)$
List	$O(n)$	$O(1)$ → gave credit for $O(n)$
Tree	$O(\log n)$	$O(1)$ → gave credit for $O(\log n)$

1 (c) Multiple choice, circle one: the heap *implementation* in this class made use of a

- [✓ array/vector]
- tree
- both of the above

1 (d) For trees, in-order, post-order, and pre-order all are forms of what type of traversal? Circle **BFT** (breadth-first traversal) or [✓ **DFT**] (depth-first traversal).

1 (e) Circle [✓ **True**] or **False**: trees are a subset of graphs.

3 (f) What is the main implementation difference between a BFT for a tree and a BFT for a graph? What features of graphs makes this implementation difference necessary?

- In BFT for a graph, we needed to mark nodes as visited so we didn't traverse a node twice.
- This is necessary because graphs can have cycles.

note { • Many people said a tree only has 2 children
[illegible]

3

(g) For the following two blocks of code, rewrite each to improve the style and conciseness of the code. The result should be the same.

```
• if (x > 0) {
      System.out.println("here");
  } else if (x <= 0) {
      System.out.println("there");
  }
```

not necessary + can cause confusion with more complicated code.

```
if (x >0) {
    System.out.println ("here");
} else {
    System.out.println("there");
}
```

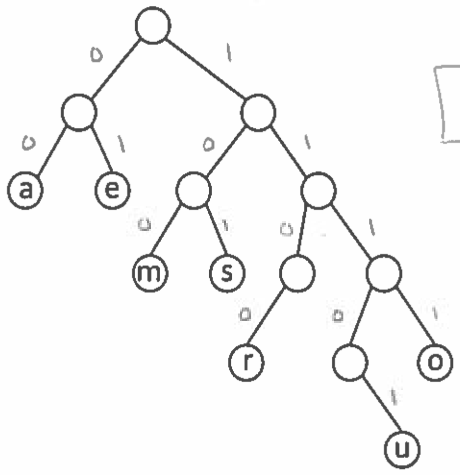
```
• int i = 0;
  while(true) {
      System.out.println(i);
      i = i + 1;
      if (i == 5) {
          break;
      }
  }
```

```
for (int i =0; i< 5 ; i++){
    System.out.println(i);
}
```

3

(h) Given the following HuffTree (left for 0, right for 1), decode the message below:

encoded message = 1 0 1 | 1 1 1 0 1 | 1 0 0 | 1 0 0 | 0 1 | 1 1 0 0



$\boxed{\text{s u m m e r}}$

Why are there no alphabet symbols at the *internal nodes* of a HuffTree?

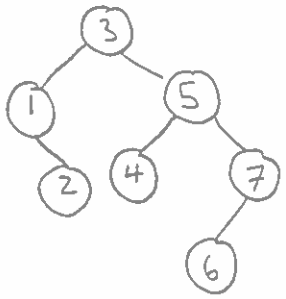
• If we had symbols at internal nodes, the code would be ambiguous, since we wouldn't know whether to stop at an internal node or keep going until a leaf.

Part 2: Trees, Heaps, and Sorting

Consider the unsorted list of numbers below:

3, 5, 1, 4, 7, 2, 6

5 (a) Create an *binary-search tree* from these elements, inserted one at a time in the order given (as we did in class to create a binary-search tree). Write down the *in-order* traversal of your tree to confirm your sorted result.

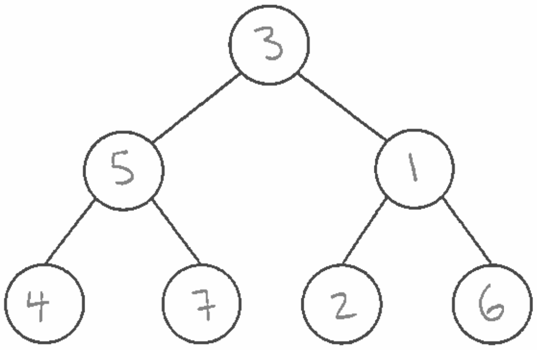


1, 2, 3, 4, 5, 6, 7 ✓

3 (b) Write down the *post-order* traversal of your tree.

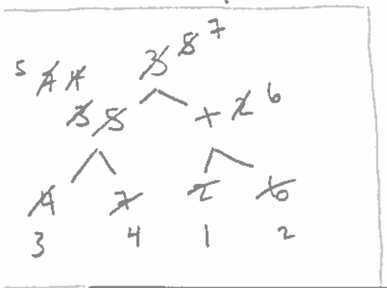
2, 1, 4, 6, 7, 5, 3

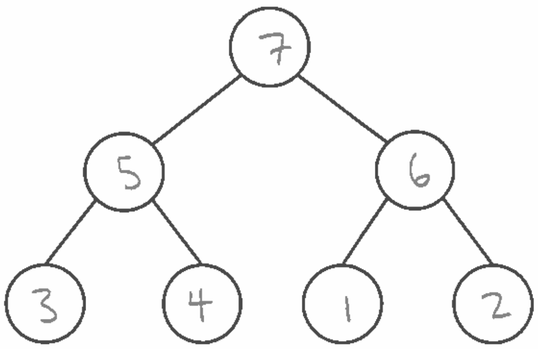
2 (c) Now consider how *heap sort* would sort these elements. First, show how this unsorted list of elements could viewed as a heap (*hint: BFT*), represented by the tree data structure below:



5 (d) Show the result of *heapifying* this list, making it into a *max heap* (Phase I). Show your work.

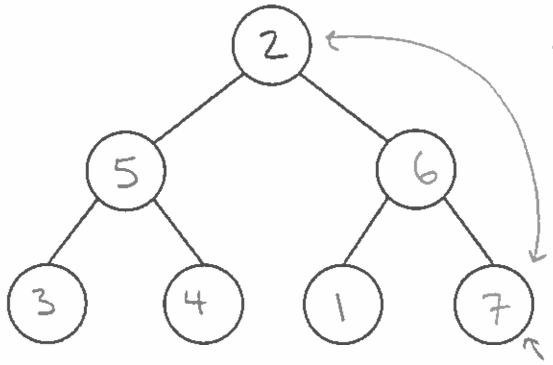
for $i = 0, 1, \ldots n$:
bubble up element i
until heapified





2 (e) Now show the result of the *first* swap during Phase II of heap sort. The result should be two elements swapped, but otherwise the same as part (d).

for i = n.... 1:
swap root + element i,
bubble down.

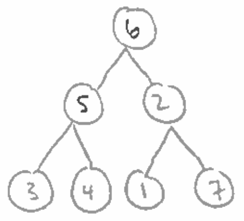


2 + 7 have been swapped.

7 is now in position and never moves.

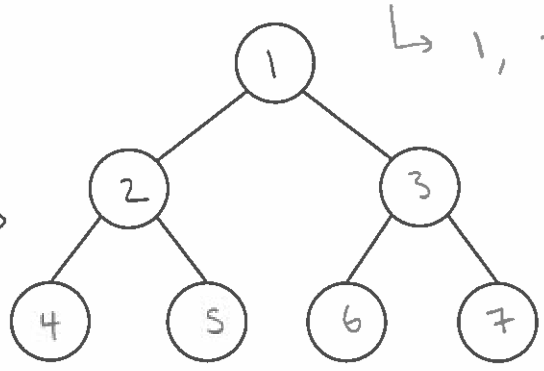
3 (f) Finally, show what the heap would look like after heap sort is complete (you don't need to show the intermediate steps). Write down the BFT of your tree to confirm your sorted result.

next swap: bubble down 2



after this, 1 + 6 would swap + 1 would be bubbled down

many steps later



↳ 1, 2, 3, 4, 5, 6, 7

no matter what the intermediate steps are, this should be the final result!

5 (g) Which of these two sorting algorithms is more efficient? Clearly justify your answer and discuss any worst-case scenarios.

- Both these algorithms (insertion sort with a binary tree + heap sort), have an asymptotic runtime of $\boxed{O(n\log n).}$ However, in the case of insertion sort with a binary tree, if the elements are mostly in order when inserted, insertion may creep up from log(n) to linear. However, heap sort will always retain its nlogn complexity, but insertion sort has a $\boxed{\text{worst-case of } O(n^2).}$

Part 3: Hash Tables

Consider the hash table below (built upon an array), which uses the hash function:

$$h(k) = k \mod 5$$

and handles collisions via linear probing as we did in lab.

Index	Key	Value
0	18	Ada
1	57	Anita
2	72	Lixia
3	23	Grace
4	78	Shafi

4 (a) First fill in the array index on the left. Based on the (key, value) pairs currently in the hash table, what can we determine about the order in which elements were inserted?

$$\left.\begin{array}{l} 18 \bmod 5 = 3 \\ 23 \bmod 5 = 3 \\ 78 \bmod 5 = 3 \end{array}\right\} \text{all } 3 \Rightarrow$$

order must have been:

(23, Grace)

(78, Shafi)

(18, Ada)

4 (b) Now we want to insert the following two elements, in this order: (72, Lixia) and (57, Anita). Write in each pair in its final position. How many steps does it take to insert (57, Anita)?

$$\left.\begin{array}{l} 72 \bmod 5 = 2 \\ 57 \bmod 5 = 2 \end{array}\right\}$$

it takes $\boxed{5 \text{ steps}}$ to insert (57, Anita), since it has to go all the way through the table.

2 (c) (Open-ended) if we now wanted to insert more elements, what is the problem? Propose a solution to this problem.

- Now our hash table is full, so we cannot insert any more elements.
- One solution (which Java does) is to automatically resize the array once the hash table gets above a certain % full.

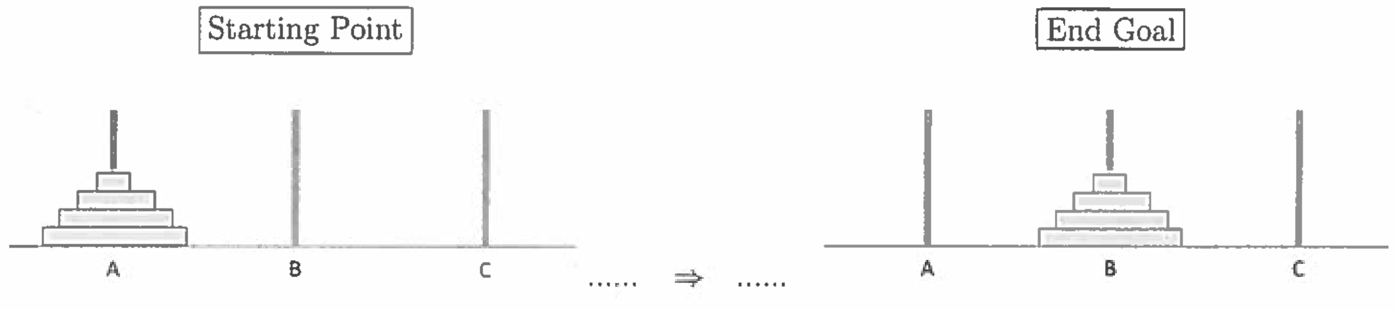
(No credit question) What do these names represent? They are all influential female computer scientists:

Ada Lovelace, Anita Borg, Lixia Zhang, Grace Hopper, Shafi Goldwasser

Part 4: Recursion

The following picture and pseudocode demonstrates the Towers of Hanoi problem. The problem is that a tower of disks sits at A, and we want to move it to B, but we have two constraints:

- only one disk may be moved at a time
- a larger disk may not be placed on top of a smaller disk

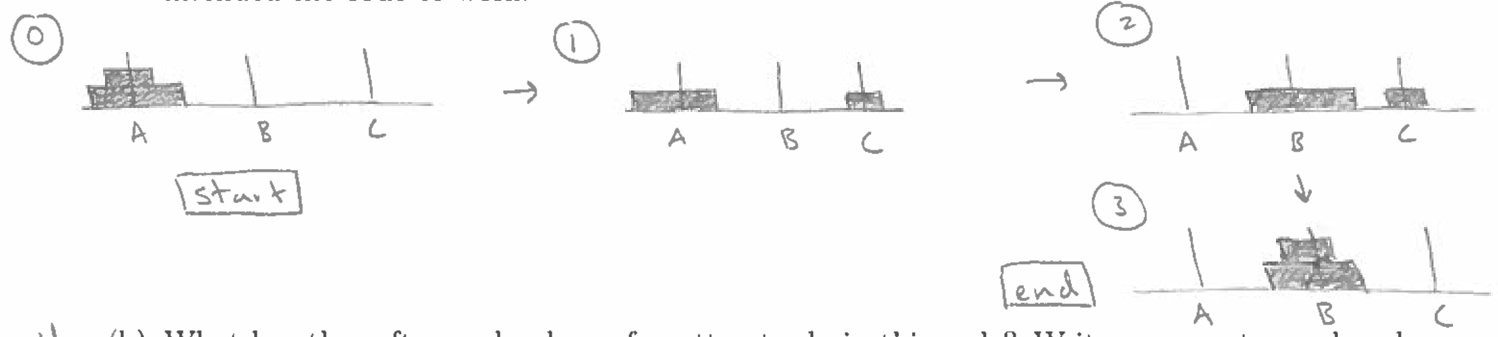


```
procedure moveTower(height, source, dest, temp):
    if height > 0 :
        moveTower(height-1, source, temp, dest)
        move disk from source to dest
        moveTower(height-1, temp, dest, source)
```

6 (a) First, what *should* happen when the procedure above is called as shown below:

`moveTower(2, A, B, C)`

So the goal is to move a stack of 2 disks from A (source) to B (destination), using C as a place for temporary storage. Draw out each step of this procedure as the software developer intended the code to work.



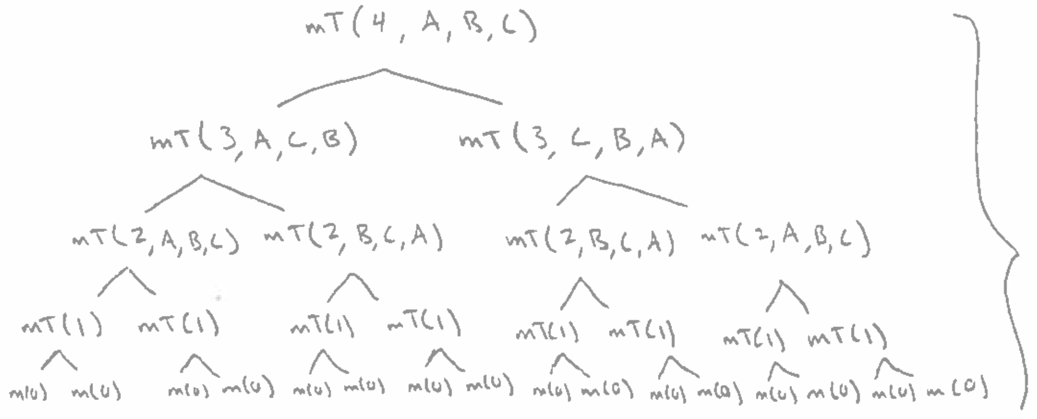
4 (b) What has the software developer forgotten to do in this code? Write some sort pseudocode to correct their algorithm, and explain where it should go.

- forgotten the base case.
- if the height is 0, the program is done + doesn't need to do anything
- we can simply add this code

8 (c) Now we call the algorithm on

`moveTower(4, A, B, C)`

How many calls to `moveTower` are made, including this first call? Justify your answer.



$\boxed{\text{31 calls to moveTower}}$

<u>Note</u>: depending on your base case, the correct result might be $\boxed{\text{15 calls}}$

7 (d) What is the runtime of `moveTower` in terms of the initial number of disks n? Count the runtime as the number of times a disk is moved. Justify your answer.

exact runtime: $\boxed{2^{n} - 1}$ $\Rightarrow$ $\boxed{O(2^n)}$ or $\boxed{\text{exponential}}$

$\Rightarrow$ In terms of the number of disk movements, the runtime for the example above is $\boxed{15}$, since for height 0 nothing is moved. So the runtime is $2^4 - 1 = 16 - 1 = 15$. It is always the number of nodes in the recursion tree that are not height 0.

$\Rightarrow$ If you gave the runtime as the number of calls to moveTower, that is also fine.

Part 5: Graphs and Shortest Path

The graph below shows 5 cities: A, B, C, D, and E, with roads between some pairs of cities. You are working for a map company, but on your first day you accidentally delete all the distance information on the edges. However, you still have the results of Dijkstra's shortest path algorithm. Your job is to reconstruct the edge information.

6

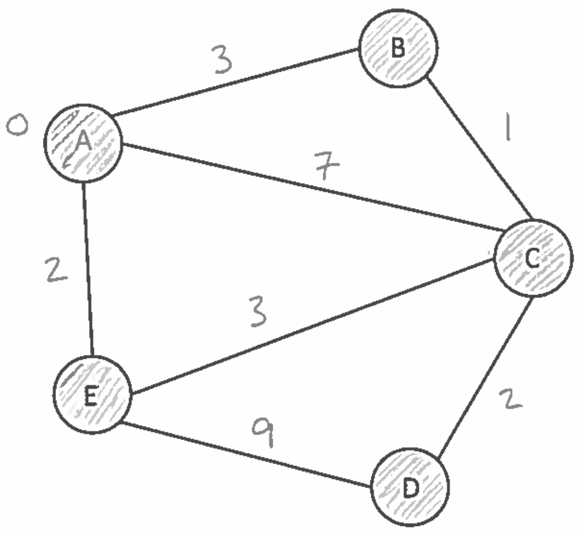
(a) The table on the below shows the distances from node A to each of the other nodes after each step of Dijkstra's shortest path algorithm. So the last row is the shortest distance to each node from A. Fill in the "visit node" column showing the order in which nodes are visited.

visit node	A	B	C	D	E
-	$\boxed{0}$	∞	∞	∞	∞
A	0	3	7	∞	$\boxed{2}$
E	0	$\boxed{3}$	5	11	2
B	0	3	$\boxed{4}$	11	2
C	0	3	4	$\boxed{6}$	2
D	0	3	4	6	2

} minimum of unvisited nodes circled each time, it becomes the new current node.

14

(b) Using the data above, write the edge weight (distance) on each edge below (7 edges total). Show your work.



(x, y) will denote the edge weight from x to y, not the shortest path between x + y.

visit A: ⇒ (A, B) = 3, (A, C) = 7, + (A, E) = 2

visit E: ⇒ (E, C) = 5 - 2 = 3, (E, D) = 11 - 2 = 9

visit B: ⇒ (B, C) = 4 - 3 = 1

visit C: ⇒ (C, D) = 6 - 4 = 2

visit D: ⇒ no unvisited neighbours, no updates.

Page for extra work (tell me which problem this is for).