15-122 : Principles of Imperative Computation

Summer 1 2012

Assignment 5

(Theory Part)

Due: Thur, June 14, 2012 in class

Name: _____

Andrew ID: _____

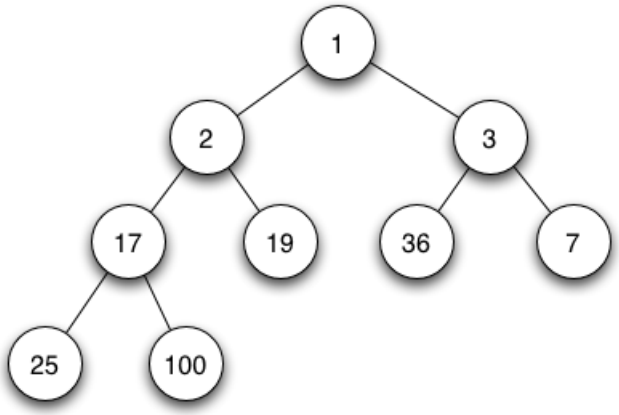
Recitation: _____

The written portion of this weeks homework will give you some practice with formalizing data structure invariants, as well as heaps and priority queues. You can either type up your solutions or write them *neatly* by hand in the spaces provided. You should submit your work in class on the due date just before lecture or recitation begins. Please remember to *staple* your written homework before submission.

Question	Points	Score
1	6	
2	8	
3	6	
Total:	20	

1. **Heaps.**

Consider the min-heap below and answer the following questions



(2) (a) Show how this heap is stored in the array and briefly summarize the method for mapping nodes in the heap to indices in the array.

Solution:

(2) (b) What is the result of inserting an element with value 0 into the heap? Draw your answer as a tree. (You are only required to show the final tree, but may show intermediate steps to ensure maximum partial credit).

Solution:

(2) (c) What is the result of a `heap_delmin` operation on the original tree? Draw your answer as another tree. (You are only required to show the final tree, but may show intermediate steps to ensure maximum partial credit).

Solution:

2. **Runtime Complexity.**

(2) (a) We have n unsorted elements. What is the worst-case runtime complexity of building a heap from these elements by inserting them one at a time to an array? Assume that the array is large enough to hold the final heap. Justify your answer.

Solution:

(3) (b) Consider our implementation of min-heaps from class(e.g., the *only* heap invariant we have is that a parent is always less than its two children and that the binary tree is complete). Given a min-heap of n elements, design and explain an algorithm to find the maximum element (no code is necessary). What is the worst-case runtime complexity of your algorithm? **Hint:** don't think about this strictly in terms of the interface functions for heaps. Just consider the structure of the tree and the heap invariant.

Solution:

(3) (c) If you insert the numbers $1, \ldots, 63$ (inclusively) **in any order** into a min-heap, what is the smallest number that could be a leaf node? Explain your answer.

Solution:

3. **Priority Queues.**

(2) (a) We can implement a sorting algorithm using a heap. Explain, on an algorithmic level, how this might work. Analyze the runtime complexity of this algorithm. **No code is necessary.**

Solution:

(2) (b) How might you use the priority queue to implement the ordinary LIFO stack? Recall that the item with minimum priority is always removed from a priority queue. What is the worst-case runtime complexity of your `push` and `pop` operations? **No code is necessary.**

Solution:

(2) (c) How might you use the priority queue to implement the ordinary FIFO queue? What is the worst-case runtime complexity of your `enqueue` and `dequeue` operations?
No code is necessary.

Solution: