Midterm #2

Instructions: This is a closed book, closed notes exam. There are **6** problems. Do your best to communicate your ideas *clearly* and *succinctly*. Draw a picture if it is useful to explain something. Show all work on these pages.

Good luck!

NAME:

On problem	you got	out of
1		15
2		10
3		15
4		15
5		15
6		25
Σ		100

1 Tree traversals [15 points]

Consider the following binary tree, T:

A. Pre-order traversal.

List the order in which nodes are visited in a **pre-order** traversal of T.

B. In-order traversal.

List the order in which nodes are visited in an **in-order** traversal of T.

2 Minimum Spanning Tree [10 points]

Think back to Kruskal's algorithm — the one which finds a minimum spanning tree by growing a forest. Suppose you run Kruskal's algorithm on the weighted tree T pictured below. **Darken** the edges which will appear in the resulting minimum spanning tree.

3 Union / Find [15 points]

A. Suppose you use the oriented forest described in class (and in your text) to implement the `union/find/makeset` operations for manipulating disjoint sets. Suppose you use **union by size** and **collapsing find**. Suppose you currently have a set representation which looks like this:

Draw the oriented forest which you'll have after performing the operation `Union(3,2)`.

B. Starting the forest which **I gave you** in part (A), draw the oriented forest which will result after you do a `Find(15)`.

4 Hashing [15 points]

In a sentence (or two) describe each of the following concepts.

A. Dynamic Rehashing

B. The load factor, α (of an m-slot hash table currently containing n items).

---

5 Sorting [15 points]

The *median* of n numbers $a_0 \leq a_1 \leq \cdots \leq a_{n-1}$ is element $a_{\lfloor n/2 \rfloor}$ — that is, the "middle" element if there are an odd number of elements, and the left one of the two middle elements if there are an even number of elements. (Thus 3 is the median of $2, 3,\ 3,\ 9, 10$; and 3 is the median of $2, 3,\ 3, 9,\ 10, 80$.)

The **Median Quicksort** algorithm is just like ordinary Quicksort — only that we run a **linear-time** (comparison-based) algorithm which (somehow) manages to find the median element of the numbers we pass to it. Median quicksort then pivots on this median element (rather than the first element or a random element, as we talked about in class).[1]

Write a **recurrence relation** which describes the **worst case** running time, $T(n)$, for the time it takes **Median Quicksort** to sort n numbers.

[1] In other words, we find the median and call it the *pivot*; in linear time, we partition the elements of the array into (i) those less than or equal to the pivot (but not the pivot itself), (ii) the pivot itself, (iii) and those greater than or equal to the pivot (but not the pivot itself); and then, finally, we recurse on lists (i) and (iii).

6 Heaps **[25 points]**

A. Modify the following binary heap to show the one which results after you do a `delete_max()` on it.

B. Modify the following binary heap to show the one which results after you do an `insert(50)` into it.

C. Give the "standard" array representation (the one used in class and in your text) for the heap **I gave you** in part **A**.

1	2	3	4	5	6	7	8	9

D. In a binary tree T represented as an array $T[1], \ldots, T[n]$, as above,

the **parent** of node $T[i]$ is stored at $\boxed{\qquad\qquad}$;

the **left child** of the node $T[i]$ is stored at $\boxed{\qquad\qquad}$; and

the **right child** of the node $T[i]$ is stored at $\boxed{\qquad\qquad}$.

E. Now consider generalizing a **binary** ("2-ary") heap in the natural way, to make a k**-ary** heap. For example, the following would be a **3-ary** heap:

Generalizing Parts **C** and **D**, the above tree would be stored in a 9-element array $T[1], \ldots, T[9]$. In general: for a k-ary binary tree,

the **parent** of node $T[i]$ is stored at $\boxed{\qquad\qquad}$;

the **first (leftmost) child** of the node $T[i]$ is stored at $\boxed{\qquad\qquad}$; and

the **second child** of the node $T[i]$ is stored at $\boxed{\qquad\qquad}$;

. . .; and

the **last (rightmost) child** of the node $T[i]$ is stored at $\boxed{\qquad\qquad}$.

E. Figure out the most efficient `delete_max` for a k-ary heap which you can find. In the worst case, how many comparisons —as a function of n and k— does it take to `delete_max` from an n-node, k-ary binary tree?

F. Figure out the most efficient `insert` for a k-ary heap which you can find. In the worst case, how many comparisons —as a function of n and k— does it take to `insert` a new node into an n-node, k-ary binary tree?