Final

Instructions: This is a closed book, closed notes exam. Do all **8** problems (13 pages). 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:

SIGNATURE:

On problem	you got	out of
1		20
2		20
3		20
4		20
5		20
6		20
7		20
8		20
Σ		160

1 Definitions [20 points]

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

A. Information Hiding

B. Universal hashing

C. Amortized running time

D. Heap Property

2 Sorting [20 points]

Give the best estimate you can to answer each of the following questions. All running times refer to worst-case running times. Assume the machine in question has ample memory so that you can access an entry in a 10 million element array just as efficiently as you can access an entry in a 1 million element array.

A. Suppose you have written a program P_A which sorts **1 million** 64-bit numbers in **1 second** (or less) using **insertion sort**. Estimate the amount of time P_A will need to sort **10 million** 64-bit numbers.

B. Suppose you have written a program P_B which sorts **1 million** 64-bit numbers in **1 second** (or less) using **heapsort**. Estimate the amount of time P_B will need to sort **10 million** 64-bit numbers.

C. Suppose you have written a program P_C which sorts **1 million** 64-bit numbers in **1 second** (or less) using **radix sort**. Estimate the amount of time P_C will need to sort **10 million** 64-bit numbers.

3 Design [20 points]

Suppose you are given a list of unsorted 32-bit integers $x_1, \ldots, x_n$. You know nothing about the distribution $\mathcal{D}$ that this list of numbers is taken from. Think of n being large — several million. Your goal is to design an algorithm with fast **worst-case** running time which finds a number which appears **most often** in the list. Discuss how you would solve this problem.

4 Splay trees [20 points]

Suppose you start with the indicated splay tree and then do an **insert** of a node with key **20**. Find the resulting splay tree (the one that results after you do an insert of the node and then splay it all the way up until the root). Show the intermediate steps.

5 Some basic algorithms [20 points]

Describe (in clear English or clear pseudo-code — no need to write correct C++):

A. an efficient algorithm to find the **in-order successor**, s, of node q in a binary tree T. (Note: your algorithm should be more efficient than doing an in-order traversal starting from the root, and checking to see what is the successor of q.)

B. an efficient (linear-time) algorithm to randomly shuffle the array elements $a[0], \ldots, a[n-1]$.

C. an efficient algorithm to increment an n-bit binary number M, this number stored as $b = b[n-1] \cdots b[0]$, each $b[i]$ a number, 0 or 1, with $b[n-1]$ the most significant bit and $b[0]$ the least significant bit.

6 Dijkstra's algorithm

A. Use Dijkstra's algorithm to find the shortest path from node s to each other node. Indicate what you are doing by neatly crossing out the upper-bounds on distances ($d[v]$ values in the notation of class) which the algorithm finds as it runs. I have started you off by giving s a value $d[s] = 0$ and all the other nodes v a $d[v]$-value of ∞.

B. What data structure did we use to obtain the fastest running time for Dijkstra's algorithm?

C. Using this data structure, what is the time to find the shortest path from s to each other node v in an n-node, m-edge connected graph?

7 Heaps and amortized analysis

A. Recall the potential function we used for analyzing binomial heaps: Φ =number of trees. Fill in the following table. Explain any notation you introduce.

	actual cost	$\Delta\Phi$	amortized cost
`insert`			
`delete_min`			
`merge`			

B. Describe (in English) how the `delete_min` operation is performed above, and why you filled in the second row of the table they way you did.

C. Which (if any) of the above operations work differently in a Fibonacci heap (F-heap). Explain the difference in a sentence or two.

8 Binary Tries

Recall a data structure we talked about on the last day of class: a **binary trie**. This is a binary tree with two types of nodes: *branch nodes* and *element nodes*. The leaves of the tree are element nodes, while the internal nodes are branch nodes. Only the element nodes contain keys. All keys have the same number of bits. Here is an example:

A. Devise a simple algorithm to **insert** a node in a binary trie. Do not describe your algorithm; instead, draw the binary trie which results if you insert a node with key **1111** into the binary trie pictured above.

B. How long does it take to decide if a given key is **in** a binary trie? What is the $\Theta(\cdot)$ time complexity of this operation (in the worst case)? Let n denote the number of data nodes in the binary trie; let m denote the number of branch nodes; and let k denote the number of bits in a key.

C. *Starting from the initially depicted binary trie,* draw the *Patricia* that represents the same elements. (Recall that a Patricia is obtained by steps which include: "compressing" the binary trie (this includes adding bit numbers to branch nodes— use the book's convention of numbering bits left-to-right by 1-to-4); eliminating the element nodes; adding a new node; and adding various pointers. This was the last data structure we talked about, on the last day of class.)