NAME_____

COMP 2370 Final Exam, Fall 2011

This is an open book, open note exam. If you get stuck, you may purchase hints for points.

In questions 1, 2, and 3, you are asked to select a data structure well-suited to a particular task. Please choose the structure from among the following: array, stack, queue, singly linked list, doubly linked list, doubly linked list with sentinel, direct address table, hash table, binary search tree, red-black tree, and B-tree.

1. (10 points) Suppose you are writing a program to model planetary phenomena. The program needs to look up an experimentally determined reaction rate for the temperature being modeled. Automated testing has provided reaction rates for 1000 temperatures in the range -200° C to 470° C. The tested values are more densely spaced in critical zones, with measurements 0.01° C apart, but are much sparser in other ranges. What data structure would you use so that, on average, your program would have very fast access to the reaction rate for a given temperature? Explain.

   A hash table is a good choice. It has expected $\Theta(1)$ look-up. The model is not a real-time application, so an occasional slow look-up isn't a problem. The application doesn't insert or delete reaction times.

2. (10 points) You are writing a program for a forensic lab that will help classify compounds. One search the program performs is to find the compound in the stored data with the closest melting point to that of the query compound. The key of each data item is its melting point. It also has satellite data regarding the compound. The data set is in a file that fits easily in RAM and is read in when the program is loaded and saved when the program is closed. Items may be added to or deleted from the data set as the program runs. What data structure would you use so that insertion, search for SUCCESSOR and PREDECESSOR, and deletion are all reasonably fast?

   A red-black tree does reasonably fast ($\Theta(\lg n)$) successor, predecessor, insertion, and deletion.

3. Recall the set grouping problem from the programming project : given a set of n numbers and a required spacing, d, group the numbers into subsets so that values within a given subset differ by at least d, using as few subsets as possible.

   Suppose the array A is already sorted. An algorithm for the set grouping problem is described here. Put the first element , A[1], in a new subset, *currentSubset*. Iterate through the array, inserting the element A[j] from the array into the subset *currentSubset* until the distance between A[j] and the most recently inserted element is less than d. Save *currentSubset* into a

data structure *Subsets* by storing a pointer to the subset. Make a new *currentSubset* containing just A[j].

Continue to iterate through the array, inserting the element A[j] from the array into the subset *currentSubset* until the distance between A[j] and the most recently inserted element is less than d. Save *currentSubset* into the data structure *Subsets*. Find the subset in *Subsets* that has gone the longest without an insertion. If A[j] is at least a distance d from the most recently inserted element of that subset make that subset the *currentSubset* . If A[j] is less than d from the most recently inserted element of that subset, make a new *currentSubset* . Insert A[j] into *currentSubset* and continue iterating through A.

```
SUBSET-PARTITION(A)
Make an empty data structure to hold subsets, Subsets.
If A.length>0
    make an empty subset,  currentSubset
    insert A[1] in currentSubset
    for j in 2 to A.length
        if A[j]-(newest element of currentSubset)<d
                insert currentSubset into Subsets
                S=element of Subsets that has gone the longest without an insertion
                if A[j]-newest element of S>=d
                        currentSubset=S
                else
                        make an empty subset, currentSubset
        insert A[j] in currentSubset
return Subsets
```

a) (4 points)What data structure should the algorithm use to store the elements of an individual subset, such as *currentSubset*?
A stack is ideal. We need to be able to insert (PUSH), and read the most recently inserted element (TOP or POP then PUSH it back)

b) (4 points) What data structure should the algorithm use for the collection *Subsets*?
Use a queue. If we execute currentSubset = S, then we dequeue S.

c) (2 points) Give a theta bound on the running time of SUBSET-PARTITION (not including the time to sort A).
$\Theta(n)$: Using stacks for the subsets and a queue for Subsets all the operations in the loop are $\Theta(1)$.

4. (5 points)Suppose you implement a stack with an array, initially of size n=m, as described on page 233 of your text. In addition, you manage overflow by doubling the size of the array and copying the data into the new array. If the size of the stack drops to 1/4 the size of the array, n, you resize the array to max{m, n/2}. Is PUSH worst case $\Theta(n)$? Explain. Yes.
In the worst case, the push causes an overflow, and n+1 items will have to be written into a new array.

5. (5 points) If you implement a stack by insertion and deletion from the head of a singly linked list implemented with pointers, is PUSH worst case Θ(n)? Explain. No.
All PUSH's will be Θ(1)

6. (5 points) Give an order in which keys could have been inserted into the open address hash table below if there was no deletion and the hash function was h(k,i)=(k+i)mod 11.

0	1	2	3	4	5	6	7	8	9	10
44	10	57	13		27					32

probe sequences
44 10 57 13 27 32
0 10 2 2 5 10
0 3
1

, so any order with 13 after 57 and 10 after 32 and 44 will work.
Ex 40, 57, 27, 32, 10, 13

7. In this problem you calculate the expected running time of insertion sort if the array of values to be sorted does not contain any duplicates, and any input order of the elements $z_1<z_2<\ldots z_n$ is equally likely. Let N be the total number of upward shifts done to sort A. Note that the time to sort A, T(n), is in Θ(N+n). Note further that if i<j then positioning z_i will cause z_j to be shifted to a higher index if and only if z_i occurs later than z_j (at a larger index) in A. Define X_{ij} to be that indicator random variable that equals 1 if z_i occurs later than z_j in A, and equals 0 otherwise.

a) (3 points) What is the expected value of X_{ij}, $E[X_{ij}]$? $\frac{1}{2}$
The number of permutations with z_i after z_j is $\frac{n!}{2} = \binom{n}{2}(n-2)!$, so the expected value of the indicator X_{ij} for this event is $\frac{n!}{2}\cdot\frac{1}{n!} = \frac{1}{2}$

b) (3 points) What is $E\left[\sum_{i\neq j} X_{ij}\right]$?

$$= \sum_{i\neq j} \frac{1}{2} = \frac{n(n-1)}{2}$$

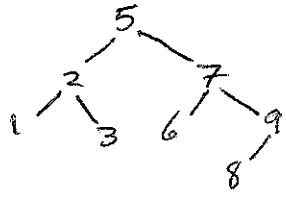
c) (2 points) What is the expected value of N? You may give your answer in terms of your answer to part b, say B(n).

$$= E\left[\sum_{i<j} X_{ij}\right] = \sum_{i<j} E[X_{ij}] = \frac{n(n-1)}{2}\cdot\frac{1}{2} = \frac{n(n-1)}{4}$$

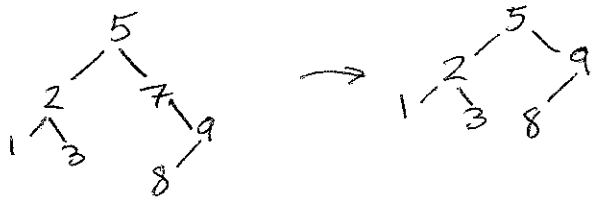
d) (2 points) Give a Θ-bound on the expected value of T(n). You may give your answer in terms of your answer to part c, calling that C(n).

$$\Theta\left(\frac{n(n-1)}{4} + n\right) = \Theta(n^2)$$

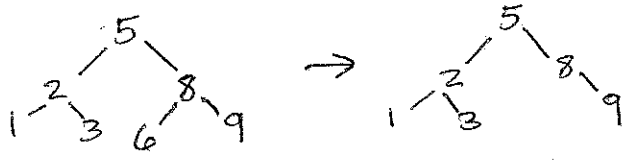
8. The next two parts refer to the binary search tree below:



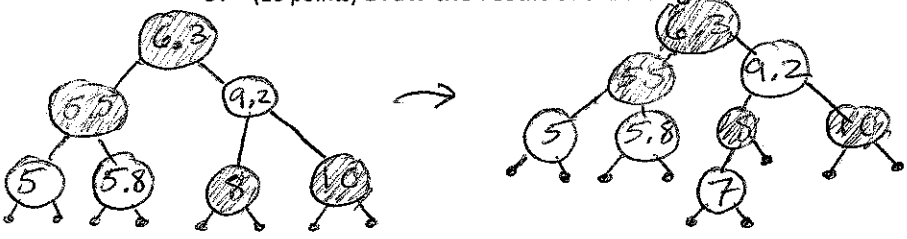
a) (5points) Draw the result of deleting 6 then 7.



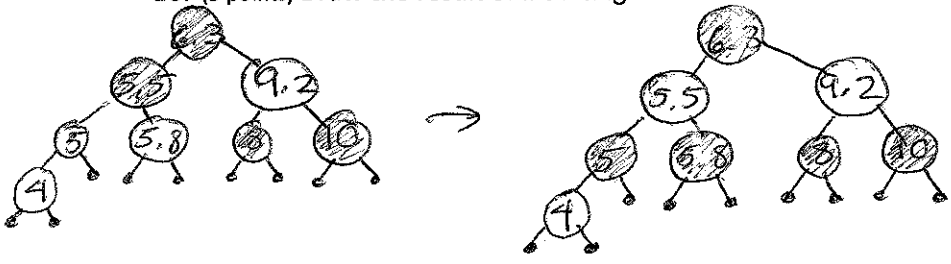
b) (5points) Draw the result of deleting 7 then 6.



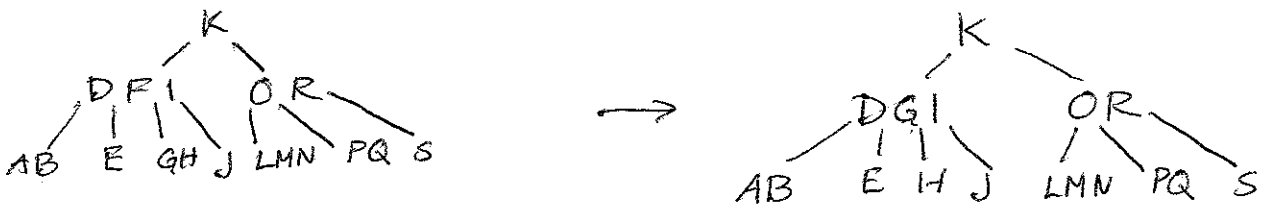
9. (10 points) Draw the result of inserting 7 in the red-black tree below.



10. (5 points) Draw the result of inserting 4 in the red-black tree below.



11. (10 points) Delete F from the B-tree below. The B-tree has minimum degree t=2.



12. (10 points) Trace the result of applying PARTITION to the array below. Give the configuration of the array each time j is set, indicating the positions of I and j.

5	15	16	4	3	10

(p=1, r=6)
x=10

i 5 j 15 16 4 3 10

5 i 15 j 16 4 3 10

5 i 15 16 j 4 3 10

5 i 15 16 4 j 3 10

5 4 i 16 15 3 j 10

5 4 3 i 15 16 10 j

(The function finishes with 5 4 3 i 10 16 15 j)

returning 4, the index of the pivot x after the swap.)