NAME: **login ID:**
precept:

COS 226 Midterm Exam, Spring 2009

This test is 10 questions, weighted as indicated. The exam is closed book, except that you are allowed to use a one page cheatsheet. No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided. *Put your name, login ID, and precept number on this page* (*now*), and write out and sign the Honor Code pledge before turning in the test. You have 80 minutes to complete the test.

"I pledge my honor that I have not violated the Honor Code during this examination."

1	/5
2	/5
3	/5
4	/10
5	/10
6	/10
7	/10
8	/10
9	/20
10	/15
TOTAL	/100

March 9, 2009

1. **Partitioning** (5 points). Give the result of partitioning the array with standard Quicksort partitioning (taking the `N` at the left as the partitioning element).

   ```
   N E W P A R T I T I O N Q U E S T I O N
   ```

2. **Estimating running time** (5 points). Suppose that you run the program below (brute-force solution to the 4-sum problem) for `N` = 1000 and observe that it takes 1000 seconds. Predict its running time (in seconds) for `N` = 10000 and give a formula that estimates the running time as a function of `N`.

   ```
   int brute(int a[], int N)
   {
     int i, j, k, m;
     for (i = 0; i < N; i++)
       for (j = i+1; j < N; j++)
         for (k = j+1; k < N; k++)
           for (m = k+1; m < N; m++)
             if (a[i] + a[j] + a[k] + a[m] == 0) return 1;
      return 0;
   }
   ```

   Predicted running time (in seconds) for `N` = 10000: __________________

   Estimated running time (in seconds) as a function of `N`: __________________

3. **Union-find trees** (5 points). Circle the letters corresponding to arrays that could not possibly occur during the execution of weighted quick union with path compression:

		0	1	2	3	4	5	6	7	8	9
	i :	0	1	2	3	4	5	6	7	8	9
A.	a[i]:	0	1	2	3	4	5	6	7	8	9
B.	a[i]:	7	3	8	3	4	5	6	8	8	1
C.	a[i]:	6	3	8	0	4	5	6	9	8	1
D.	a[i]:	0	0	0	0	0	0	0	0	0	0
E.	a[i]:	9	6	2	6	1	4	5	8	8	9
F.	a[i]:	9	8	7	6	5	4	3	2	1	0

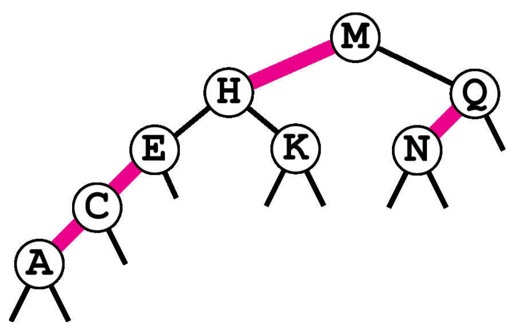
4. **Sorting algorithms** (10 points). Match each of the sorting algorithms below with its *primary distinguishing characteristic* (as presented in lecture and in the book) by writing the letter corresponding to each algorithm in the blank to the left of the corresponding characteristic. You should use each letter once and only once.

A. Mergesort	_____ Adapts well to duplicates
B. Quicksort	_____ Optimal time and space
C. Shellsort	_____ Adapts well to order
D. Insertion sort	_____ Not analyzed
E. Selection sort	_____ Stable and fast
F. 3-way quicksort	_____ Optimal data movement
G. Heapsort	_____ Fastest general-purpose sort

5. **Dynamic arrays** (10 points). The following list gives possible choices for using a dynamic array in a pushdown stack implementation. Write *linear* or *quadratic* in the blank following each choice to best describe the total time required in the worst case for a sequence of `push()` and `pop()` operations.

`A. push():`	always grow array by 1	
`pop():`	always shrink array by 1	________________
`B. push():`	double array if it is full	
`pop():`	never shrink array	________________
`C. push():`	double array if it is full	
`pop():`	halve array if it is half full	________________
`D. push():`	double array if it is full	
`pop():`	halve array if it is 1/3 full	________________
`E. push():`	double array if it is full	
`pop():`	shrink array by 99 if it has 100 empty slots	________________

6. **LLRB insertion** (10 points). The following diagram shows a left-leaning red-black tree just after the node containing A is attached at the bottom. Thick lines are red links. Show the tree that results when this insertion is completed.



7. **ST implementations** (10 points). The following is a list of possible reasons for choosing one of the Java ST implementations given in lecture over another. In the blanks provided, first list the ones that might reasonably justify using red-black trees rather than hash tables, then list the ones that might reasonably justify using hash tables over red-black trees. You need not use all the choices (do not list a choice if there are reasonable arguments on both sides).

   A. Easier to use properly for built-in key types (such as `String` and `Integer`)

   B. Easier to use properly for user-defined key types

   C. Extends to handle useful operations for ordered keys

   D. Uses less space

   E. Better worst-case performance guarantee

   F. Faster for `int` keys

   Reasons to use red-black trees: __________________________

   Reasons to use hash tables: __________________________

8. **Red-black tree invariants** (10 points). The following is a list of various descriptions of possible states of nodes in a red-black 2-3 tree. Circle the ones that cannot be found in a tree built by a sequence of `put()` operations. Recall that the color of a node is the color of the link to its parent, and that the root is always black.

   A. Red node with red parent and two black children.

   B. Black node with two null children.

   C. Red node with two null children.

   D. Black node with a left child described by C. and right child described by B.

   E. Red node with black parent and black children.

   F. Black node with red parent and one red child and one null child.

   G. More red nodes than black nodes.

9. **7 sorting algorithms** (20 points). The leftmost column is the original input of strings to be sorted, and the rightmost column is the sorted result. The other columns are the contents at some intermediate step during one of the 7 sorting algorithms listed below. Match up each algorithm by writing its letter under the corresponding column. Use each letter exactly once.

that	been	also	also	into	been	year	been	also
even	even	down	back	even	even	with	even	back
than	ever	come	been	than	from	will	than	been
been	fell	been	come	been	more	more	that	come
from	from	back	down	from	next	were	from	down
next	loss	even	even	next	over	plea	next	even
show	more	ever	ever	show	plea	well	show	ever
with	next	into	fell	jobs	show	lost	with	fell
more	over	fell	from	more	than	even	more	from
were	plea	from	have	much	that	some	over	have
over	show	jobs	into	over	were	very	plea	into
plea	than	next	jobs	plea	with	next	were	jobs
fell	that	have	with	fell	fell	lead	ever	lead
time	time	lead	time	back	time	time	fell	loss
loss	were	loss	loss	loss	loss	that	loss	lost
ever	with	over	show	ever	ever	jobs	time	more
lost	also	lost	lost	lost	lost	been	also	much
also	come	more	that	also	also	also	down	next
down	down	much	more	down	down	down	lost	over
said	have	show	said	said	said	said	said	plea
some	lost	plea	some	some	some	from	come	said
have	said	that	were	have	have	have	have	show
very	some	said	very	lead	very	over	some	some
come	very	some	than	come	come	come	very	than
into	back	will	over	that	into	into	back	that
lead	into	very	lead	very	lead	fell	into	time
back	jobs	time	next	time	back	back	lead	very
year	lead	than	year	year	year	than	year	well
will	much	with	will	will	will	show	jobs	were
well	well	well	well	well	well	loss	much	will
much	will	were	much	were	much	much	well	with
jobs	year	year	plea	with	jobs	ever	will	year
	____	____	____	____	____	____	____	

A. Bottom-up mergesort
B. Shellsort
C. Insertion sort
D. Quicksort (with no random shuffle)
E. Selection sort
F. Top-down mergesort
G. Heapsort

10. **Dynamic median-finding** (15 points). You need to support a client that reads a huge a stream of numbers that are all different and needs to keep track of the *median* element in the entire stream seen so far. For example, if the client gives you the numbers `2 9 7 4 1` and then asks for the median, you must return `4`, and if the client then adds the numbers `6 8 5` and again asks for the median, you must return `5` or `6`. There are three requirements: First, you have only *constant* extra space (beyond what is needed to store the numbers themselves). Second, you must return the median in *constant* time. Third, you must process the Nth element in time proportional to $\log N$.

    Is it possible to discard some portion of the input, such that your algorithm finds the median element accurately even in the future?

    A. Yes
    B. No

    Assume that you have seen N numbers and know that the median is of those numbers is v. Which of the following is true of the median when you process the $(N+1)$st number?

    A. It does not change.
    B. It is the largest of the numbers smaller than v.
    C. It is the smallest of the numbers larger than v.
    D. Either A. or B. or C.
    E. Either B. or C, but not A.
    F. It could be any of the numbers seen so far.

    Which of the following data structures can support inserting numbers in logarithmic time and returning the maximum in constant time, using only a constant amount of extra space (beyond what is needed to store the numbers)?

    A. BST.
    B. Red-black BST.
    C. Binary heap.
    D. Sorted array.
    E. Linked list.

    In one or two sentences, describe how you would solve the problem.