COS 226	Algorithms and Data Structures	Spring 2008

Midterm

This test has 8 questions worth a total of 80 points. You have 80 minutes. 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. **Write out and sign the Honor Code pledge before turning in the test.**

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

Problem	Score
1	
2	
3	
4	
Sub 1	

Problem	Score
5	
6	
7	
8	
Sub 2	

Total	

Name:

Login ID:

Precept:

P01	12:30	Moses
P01A	12:30	Szymon
P02	1:30	Szymon
P02A	1:30	Moses
P03	3:30	Nadia

1. **8 sorting algorithms. (16 points)**

   The column on the left is the original input of strings to be sorted; the column on the right are the string in sorted order; the other columns are the contents at some intermediate step during one of the 8 sorting algorithms listed below. Match up each algorithm by writing its number under the corresponding column. Use each number exactly once.

null	hash	fifo	list	type	find	hash	find	exch	exch
type	heap	next	heap	tree	hash	null	hash	fifo	fifo
null	fifo	null	fifo	swim	heap	null	heap	find	find
hash	link	hash	exch	swap	lifo	type	link	hash	hash
null	list	null	next	push	link	heap	list	heap	heap
heap	next	heap	leaf	swap	list	link	null	leaf	leaf
sort	find	exch	find	swap	null	null	null	left	left
link	lifo	link	hash	null	null	sort	null	less	less
list	exch	list	node	list	null	find	push	lifo	lifo
push	leaf	less	left	path	null	list	sort	link	link
find	less	find	less	less	null	push	swap	list	list
swap	left	left	lifo	null	push	swap	type	next	next
null	node	node	null	sink	root	lifo	null	null	node
null	null	leaf	null	null	sort	null	null	null	null
root	null	lifo	null	sort	swap	null	root	root	null
lifo	null	null	link	null	type	root	lifo	null	null
swap	null	swap	null	link	exch	leaf	swap	swap	null
leaf	null	null	push	leaf	fifo	path	leaf	null	null
tree	null	tree	root	hash	leaf	swap	tree	tree	null
path	null	path	null	null	left	tree	path	path	null
node	null	null	null	node	less	exch	node	node	null
left	path	swap	sink	left	next	left	left	sort	path
less	tree	push	sort	find	node	less	less	push	push
exch	swap	sort	null	exch	null	node	exch	null	root
null	sink	null	null	null	null	null	null	null	sink
sink	swim	sink	swap	heap	null	null	sink	sink	sort
swim	root	swim	swim	null	path	sink	swim	swim	swap
null	swap	null	path	null	sink	swim	null	null	swap
next	swap	type	swap	next	swap	fifo	next	swap	swap
swap	push	swap	type	root	swap	next	swap	swap	swim
fifo	sort	null	tree	fifo	swim	null	fifo	type	tree
null	type	root	swap	lifo	tree	swap	null	null	type
----	----	----	----	----	----	----	----	----	----
0									1

(0) Original input

(1) Sorted

(2) Selection sort

(3) Insertion sort

(4) Shellsort (*13-4-1 increments*)

(5) Mergesort (*top-down*)

(6) Mergesort (*bottom-up*)

(7) Quicksort (*standard, no shuffle*)

(8) Quicksort (*3-way, no shuffle*)

(9) Heapsort

2. **More sorting. (8 points)**

   (a) Modern computers have memory caches, which speed up reads and writes if they are to locations near recently-accessed memory. This makes sequential access to memory faster, in general, than random access. Circle the sorting algorithm below that you would expect to *benefit least* from caching?

   insertion sort   mergesort   quicksort   heapsort

   (b) You are managing the accounts for BigIBankCo, and have an array of customers together with their balances. You would like to rearrange the array such that the richest customers (those with balances greater than $1 million) are grouped at the beginning, with everyone else at the end.

   Describe an algorithm for performing this task in linear time, and using only constant extra memory. Adhering to the spirit of code reuse, adapt an algorithm from class and describe only the changes you would make.

3. **Hard problem identification. (10 points)**

   You are applying for a job at a new software technology company. Your interviewer asks you to identify the following tasks as *easy (E)* or *impossible (I)*.

___ Build a balanced BST containing N keys using $\sim 8N$ compares (where the array of keys are given to you in ascending order).

___ Build a balanced BST containing N keys using $\sim 8N$ compares (where the array of keys are given to you in arbitrary order).

___ Build a binary heap containing N keys using $\sim 2N$ compares (where the array of keys are given to you in arbitrary order).

___ Build a BST containing N keys that has height at most $\frac{1}{2} \lg N$.

___ Design a priority queue that does *insert* and *delete-max* in $\sim \lg \lg N$ compares per operation, where N is the number of items in the data structure.

4. **Priority queues. (10 points)**

Consider the following code fragment.

```
MaxPQ<Integer> pq = new MaxPQ<Integer>();
int N = a.length;
for (int i = 0; i < N; i++) {
   pq.insert(a[i]);
   if (pq.size() > k) pq.delMax();  /* MARK */
}
for (int i = 0; i < k; i++)
   System.out.println(pq.delMax());
```

Assume that `a[]` is an array of integers, `MaxPQ` is implemented using a binary heap, and $N \geq k \geq 1$.

(a) What does it output?

(b) What is the order of growth of its worst-case running time. Circle the best answer.

$k \log k$ $\quad$ $k \log N$ $\quad$ $N \log k$ $\quad$ $N \log N$ $\quad$ N^2

Now suppose the marked line was deleted. Repeat the previous two questions.

(c) What does it output?

(d) What is the order of growth of its worst-case running time. Circle the best answer.

$k \log k$ $\quad$ $k \log N$ $\quad$ $N \log k$ $\quad$ $N \log N$ $\quad$ N^2

5. **Binary heaps. (8 points)**

Consider the following binary heap (i.e., the array-representation of a heap-ordered complete binary tree).

0	1	2	3	4	5	6	7	8	9	10	11	12	13
-	Z	W	Y	T	G	K	V	R	S	F	A	-	-

(a) *Delete* the maximum key. Give the resulting binary heap. *Circle* those values that changed.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
-													

(b) *Insert* the key `X` into the *original* binary heap. Give the resulting binary heap. *Circle* those values that changed.

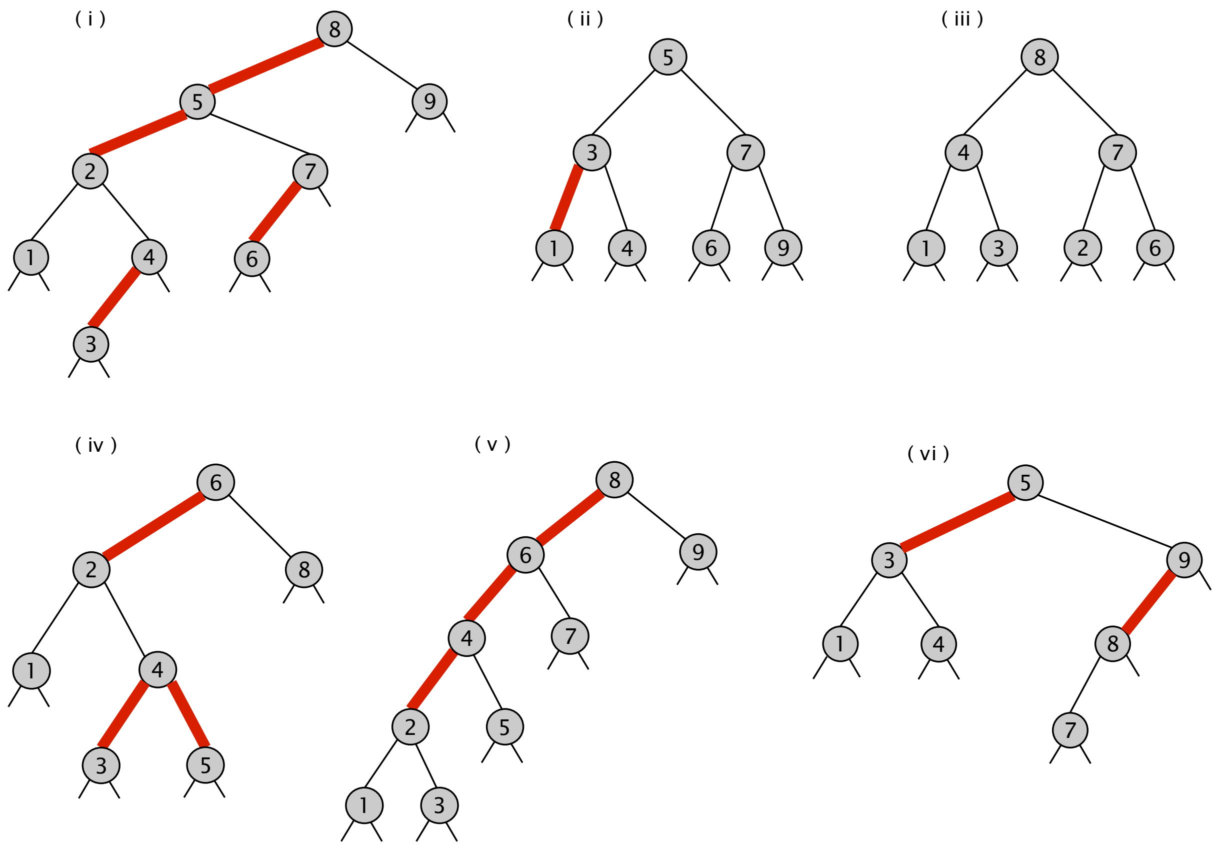
0	1	2	3	4	5	6	7	8	9	10	11	12	13
-													

6. **Left-leaning red-black trees. (10 points)**

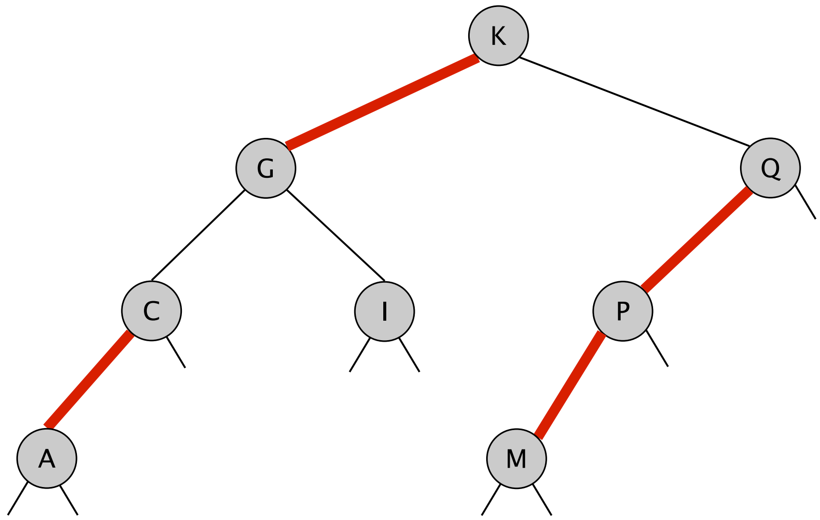
   (a) Identify which of the figures below represent legal *left-leaning red-black trees*?
   (As usual red links are drawn with thick lines.)

   legal: _________________

   illegal: _________________



(b) Consider the following left-leaning red-black tree.



Add the key H, then add the key T. Draw the resulting left-leaning red-black tree.

7. **Binary search trees. (8 points)**

Consider the following binary search tree method.

```
public Key mystery(Key key) {
   Node best = mystery(root, key, null);
   if (best == null) return null;
   return best.key;
}

private Node mystery(Node x, Key key, Node best) {
   if (x == null) return best;
   int cmp = key.compareTo(x.key);
   if      (cmp < 0) return mystery(x.left, key, x);
   else if (cmp > 0) return mystery(x.right, key, best);
   else              return x;
}
```

(a) What does `mystery(key)` return? Circle the best answer.

A. *Predecessor*: the largest key in the symbol table $<$ the search key?

B. *Floor*: the smallest key in the symbol table $\leq$ the search key?

C. *Ceiling*: the smallest key in the symbol table $\geq$ the search key?

D. *Successor*: the smallest key in the symbol table $>$ the search key?

E. *Get*: the key in the symbol table equal to the search key if it's there; null otherwise.

F. *Bad code*: Null pointer exception or infinite loop on some inputs.

(b) What is the worst-case number of compares for `mystery()`? Assume that the BST is balanced. Circle the best answer.

1 $\quad$ $\log N$ $\quad$ N $\quad$ N^2 $\quad$ 2^N

8. **Randomized queue. (10 points)**

For Assignment 2, you implemented a randomized queue that supported *enqueue* and *dequeue (delete and return random)* in *amortized* constant time (using space proportional to the number of items on the queue).

Use a balanced binary search tree to implement the two operations in *logarithmic time* per operation in the *worst-case* (using space proportional to the number of items on the queue).

Hint: simulate a dynamically resizable array using `st.get(i)`, `st.put(i, item)`, and `st.delete(i)`. Use `StdRandom.uniform(N)` to generate a random integer between 0 and $N - 1$.

```
public class RandomizedQueue<Item> {
   private RedBlackBST<Integer, Item> st = new RedBlackBST<Integer, Item>();

   // add the item to the queue
   public void enqueue(Item item) {
      int N = st.size();

      //  YOUR CODE HERE









   }


   // delete and return a random item from the queue
   public Item dequeue() {
      int N = st.size();
      if (N == 0) throw new RuntimeException("Randomized queue underflow");

      //  YOUR CODE HERE









   }
}
```