COS 226	Algorithms and Data Structures	Fall 2009

Midterm

This test has 8 questions worth a total of 60 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
0	
1	
2	
3	
Sub 1	

Problem	Score
4	
5	
6	
7	
Sub 2	

Total	

Name:

Login ID:

Precept:

P01	12:30	Anuradha
P02	3:30	Berk
P03	2:30	Corey

0. **Miscellaneous. (1 point)**

   Write your name and Princeton NetID in the space provided on the front of the exam, and circle your precept number.

1. **Analysis of algorithms. (8 points)**

   (a) Tilde notation is *more precise* than Big-Oh notation at describing the growth of a function because:

   I. Tilde notation includes the coefficient of the highest order term.

   II. Tilde notation provides both an upper bound and a lower bound on the growth of a function.

   III. Big-Oh notation suppresses lower order terms, so it does not necessarily accurately describe the behavior of a function for small values of N.

   Circle the best answer.

   (a) I only.
   (b) I and II only.
   (c) I and III only.
   (d) I, II and III.
   (e) None.

   (b) Consider the following code fragment.

```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      for (int k = j+1; k < N; k++)
         if (a[i] + a[j] >= a[k]) count++;
```

   Suppose that it takes 1 second to execute this code fragment when $N = 1000$. Using tilde notation, formulate a hypothesis for the running time (in seconds) of the code fragment as a function of N.

2. **8 sorting algorithms. (8 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.

COS	ARC	ARC	ARC	COS	ART	CHM	CHE	REL	ARC
PHY	CHE	CHE	ART	COS	CEE	ART	COS	PHY	ART
ELE	COS	COS	CEE	ELE	CHM	ARC	CHM	PHY	CEE
COS	COS	COS	CHE	PHY	ARC	CEE	COS	ELE	CHE
MAT	ECO	ECO	CHM	ARC	COS	CHE	COS	PHI	CHM
MOL	ELE	EEB	COS	LIN	CHE	COS	ART	ORF	COS
LIN	GEO	ELE	COS	MAT	EEB	COS	CEE	ORF	COS
ARC	LIN	ELE	COS	MOL	COS	COS	ARC	COS	COS
ECO	MAE	ENG	COS	CHE	COS	COS	COS	ELE	COS
CHE	MAT	GEO	COS	ECO	COS	COS	COS	EEB	COS
MAE	MOL	LIN	COS	GEO	EEB	COS	MAE	MUS	COS
GEO	PHY	MAE	ECO	MAE	COS	ORF	GEO	GEO	ECO
ORF	ORF	MAT	ORF	EEB	COS	EEB	ORF	ORF	EEB
EEB	EEB	MOL	EEB	ELE	ELE	ENG	EEB	MAT	EEB
ENG	ENG	ORF	ENG	ENG	MAE	ELE	ENG	LIN	ELE
ELE	ELE	PHY	ELE	ORF	ELE	GEO	ELE	COS	ELE
COS	COS	ART	MOL	CEE	ECO	ELE	ECO	COS	ELE
ELE	ELE	CEE	ELE	COS	ENG	MAE	ELE	ECO	ENG
CEE	CEE	COS	ELE	EEB	MAT	EEB	LIN	CEE	GEO
EEB	EEB	EEB	EEB	ELE	LIN	ECO	EEB	CHE	LIN
ART	ART	ELE	PHY	ART	ELE	MUS	MOL	ART	MAE
MUS	MUS	MUS	MUS	MUS	MUS	PHI	MUS	MAT	MAT
PHI	PHI	ORF	PHI	ORF	MAT	ORF	PHI	MAE	MAT
ORF	ORF	PHI	ORF	PHI	ORF	LIN	ORF	ELE	MOL
COS	COS	COS	GEO	COS	GEO	PHY	MAT	COS	MUS
PHY	PHY	PHY	PHY	COS	ORF	MOL	PHY	MOL	ORF
COS	COS	COS	LIN	MAT	MOL	MAT	COS	COS	ORF
MAT	MAT	MAT	MAT	PHY	PHY	MAT	MAT	EEB	ORF
CHM	CHM	CHM	MAT	CHM	ORF	ORF	ELE	CHM	PHI
ORF	ORF	ORF	ORF	COS	PHY	ELE	ORF	ENG	PHY
COS	COS	COS	MAE	ORF	PHI	REL	PHY	COS	PHY
REL	REL	REL	REL	REL	REL	PHY	REL	ARC	REL
---	---	---	---	---	---	---	---	---	---
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*)

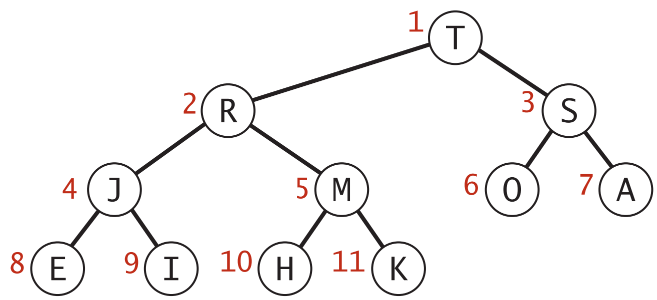
(7) Quicksort (*standard, no shuffle*)

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

(9) Heapsort

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

Consider the following max-heap.



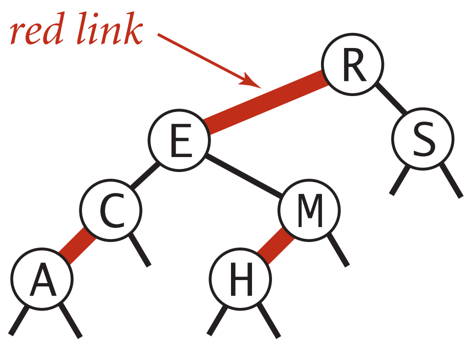
(a) The max-heap above resulted after a sequence of *insert* and *remove-the-maximum* operations. Assume that the last operation was an *insert*. Which key(s) could have been the one inserted last? Circle all possible keys.

A E H I J K M O R S T

(b) Draw the heap that results after deleting the maximum key from the heap above.

4. **Red-black trees. (8 points)**

   Consider the following left-leaning red-black tree. Add the key `Z`, then add the key `P`.



   (a) Draw the resulting left-leaning red-black tree.

   (b) How many left rotations, right rotations, and color flips are performed in total to insert the two keys?

   ___ left rotation(s) ___ right rotation(s) ___ color flip(s)

5. **Hashing. (6 points)**

Suppose that the following keys are inserted in some order into an initially empty linear-probing hash table of size 7, using the following table of hash values:

key	hash
A	3
B	1
C	4
D	1
E	5
F	2
G	5

Which of the following could be the contents of the linear-probing array?

I.

0	1	2	3	4	5	6
G	B	D	F	A	C	E

II.

0	1	2	3	4	5	6
B	G	D	F	A	C	E

III.

0	1	2	3	4	5	6
E	G	F	A	B	C	D

Circle the best answer.

(a) I only.

(b) I and II only.

(c) I and III only.

(d) I, II and III.

(e) None.

6. **Data structures. (9 points)**

Given an N-by-N grid of sites, you wish to repeated select a site (i, j) at random among all sites not yet chosen. Consider the following code fragment for accomplishing this task.

```
ArrayList<Integer> sites = new ArrayList<Integer>();
for (int id = 0; id < N*N; id++) {   // for each site,
    sites.add(id);                   // add to end of list
}
while (!sites.isEmpty()) {
    int n = sites.size();            // number of elements left in list
    int r = StdRandom.uniform(n);    // between 0 and n-1
    int id = sites.remove(r);        // remove and return item at index r
    int i = id / N, j = id % N;      // site (i, j)
    ...
}
```

(a) The `java.util.ArrayList` data type is implemented using an array (with doubling and halving). What is the order-of-growth of the *worst-case* running time of the code fragment as a function of N? Circle the best answer.

N $\quad$ $N \log N$ $\quad$ N^2 $\quad$ $N^2 \log N$ $\quad$ N^3 $\quad$ N^4 $\quad$ 2^N

(b) Which data structure that we've encountered in this course should you use instead of `java.util.ArrayList`? Circle the best answer.

i. union-find

ii. stack / queue

iii. deque

iv. randomized queue

v. binary heap

vi. red-black tree

vii. hash table

(c) For the improved version in (b), what is the order-of-growth of the *worst-case* running time as a function of N? Circle the best answer.

N $\quad$ $N \log N$ $\quad$ N^2 $\quad$ $N^2 \log N$ $\quad$ N^3 $\quad$ N^4 $\quad$ 2^N

7. **Generalized queue. (12 points)**

Design a data structure that supports the following API for a *generalized queue*.

	public class GQ<Item item>	
	GQ()	*create an empty generalized queue*
Item	**get(int i)**	*return the i^{th} item from queue*
void	**addFirst(Item item)**	*insert item at the front of the queue*
void	**addLast(Item item)**	*append item to the end of the queue*
Item	**remove(int i)**	*remove the i^{th} item from the queue*

Here is a sample client, showing the contents of the queue after each insertion / deletion.

```
GQ<String> gq = new GQ<String>();
gq.addFirst("A");          //  A
gq.addFirst("B");          //  B A
gq.addLast("C");           //  B A C
gq.addLast("D");           //  B A C D
gq.addFirst("E");          //  E B A C D
gq.addFirst("F");          //  F E B A C D
gq.addLast("G");           //  F E B A C D G

String s1 = gq.get(2);     //  s1 = "B"
gq.remove(2);              //  F E A C D G
String s2 = gq.get(2);     //  s2 = "A"
```

Your data structure should implement all operations in logarithmic time (or better) as a function of the size of the queue. You may use any of the data structures we have discussed in this course in your solution. Your solution will be graded for correctness, efficiency, clarity, and conciseness.

For half-credit, describe a data structure that implements all operations except *remove* in constant amortized time.

(a) Describe the underlying data structure. Draw it after the first 7 insertions for the example above.

(b) Describe how to implement `get()`.

(c) Describe how to implement `addFirst()` and `addLast()`.

(d) Describe how to implement `remove()`.