COS 226	Algorithms and Data Structures	Spring 2014

Midterm

This test has 9 questions worth a total of 55 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	
4	
Sub 1	

Problem	Score
5	
6	
7	
8	
Sub 2	

Total	

Name:

netID:

Room:

Precept:

P01	Th 11	Andy Guna
P02	Th 12:30	Andy Guna
P03	Th 1:30	Chris Eubank
P04	F 10	Jenny Guo
P05	F 11	Madhu Jayakumar
P05A	F 11	Nevin Li
P06	F 2:30	Josh Hug
P06A	F 2:30	Chris Eubank
P06B	F 2:30	Ruth Dannenfelser
P07	F 3:30	Josh Hug

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

   In the space provided on the front of the exam, write your name and Princeton netID; circle your precept number; write the name of the room in which you are taking the exam; and write and sign the honor code.

1. **Memory and data structures. (4 points)**

   Suppose that you implement a binary heap using an explicit binary tree data structure, where each node has three pointers (to its left child, its right child, and its parent).

```
public class BinaryHeapPQ<Key extends Comparable<Key>> {
   private Node root;           // root of tree
   private int N;               // number of keys in the data structure

   private class Node {
      private Key key;          // the key
      private Node left;        // left child
      private Node right;       // right child
      private Node parent;      // parent
   }
   ...
}
```

   Using the 64-bit memory cost model from lecture and the textbook, how much memory (in bytes) does a `BinaryHeapPQ` object use to store N keys (in N nodes)? Use tilde notation to simplify your answer.

   Do not include the memory for the keys themselves but do include all other memory.

   $\sim$ $\boxed{}$ bytes

2. **Eight sorting algorithms and a shuffling algorithm. (9 points)**

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

0	deer	sole	bass	bear	bass	bear	clam	bear	bull	tuna	bass
1	clam	slug	bull	clam	bear	bull	bear	bull	clam	swan	bear
2	bear	clam	bear	deer	bull	calf	bass	calf	bear	sole	bull
3	myna	myna	crow	dove	calf	clam	crow	clam	bass	myna	calf
4	tuna	calf	deer	moth	clam	deer	crab	deer	crow	lion	clam
5	slug	moth	clam	myna	crab	dove	calf	dove	crab	slug	crab
6	dove	bull	calf	slug	crow	gnat	bull	lynx	calf	seal	crow
7	moth	lynx	dove	tuna	deer	lynx	deer	moth	deer	mule	deer
8	lynx	deer	hoki	bull	dove	moth	lynx	myna	lynx	lynx	dove
9	bull	tuna	duck	calf	duck	myna	moth	slug	moth	crow	duck
10	calf	bear	crab	gnat	gnat	pony	sole	sole	dove	clam	gnat
11	sole	dove	mule	lynx	hoki	seal	pony	tuna	sole	puma	hoki
12	pony	pony	moth	pony	pony	slug	seal	gnat	pony	pony	lion
13	seal	seal	lynx	seal	seal	sole	gnat	hoki	seal	dove	lynx
14	gnat	gnat	gnat	sole	myna	swan	swan	mule	gnat	gnat	moth
15	swan	swan	puma	swan	swan	tuna	mule	pony	swan	moth	mule
16	mule	mule	myna	bass	mule	mule	hoki	seal	mule	deer	myna
17	hoki	hoki	seal	crab	sole	hoki	duck	swan	hoki	hoki	pony
18	duck	duck	lion	crow	tuna	duck	dove	bass	duck	duck	puma
19	crab	crab	sole	duck	slug	crab	slug	crab	slug	crab	seal
20	crow	crow	pony	hoki	lynx	crow	tuna	crow	tuna	bull	slug
21	bass	bass	tuna	lion	moth	bass	lion	duck	myna	bass	sole
22	lion	lion	slug	mule	lion	lion	puma	lion	lion	calf	swan
23	puma	puma	swan	puma	puma	puma	myna	puma	puma	bear	tuna
	----	----	----	----	----	----	----	----	----	----	----
	0										1

(0) Original input

(1) Sorted

(2) Selection sort

(3) Insertion sort

(4) Knuth shuffle

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

(6) Mergesort
(*top-down*)

(7) Mergesort
(*bottom-up*)

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

(9) Quicksort
(*Dijkstra 3-way, no shuffle*)

(10) Heapsort

3. **Analysis of algorithms. (6 points)**

Suppose that you have an array of length N consisting of alternating B's and A's, starting with B. For example, below is the array for $N = 16$.

```
B A B A B A B A B A B A B A B A
```

(a) How many compares does *selection sort* make to sort the array as a function of N? Use tilde notation to simplify your answer.

$\sim$ $\boxed{}$ compares

(b) How many compares does *insertion sort* make to sort the array as a function of N? Use tilde notation to simplify your answer.

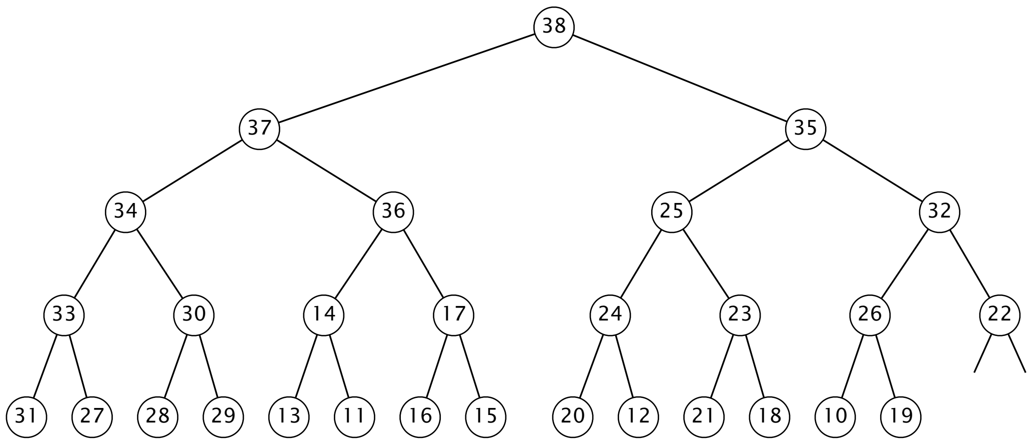
$\sim$ $\boxed{}$ compares

(c) How many compares does *mergesort* make to sort the array as a function of N? You may assume N is a power of 2. Use tilde notation to simplify your answer.

$\sim$ $\boxed{}$ compares

4. **Binary heaps. (6 points)**

Consider the following binary heap.



(a) Suppose that the last operation performed in the binary heap above was inserting the key x. Circle all possible value of x.

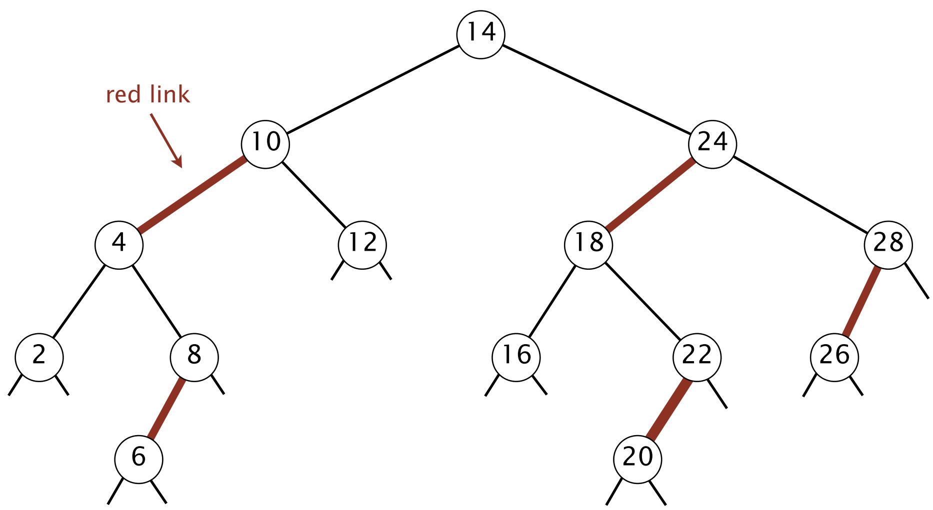
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39

(b) Suppose that you delete the maximum key from the binary heap above.
Circle all keys that are involved in one (or more) compares.

10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39

5. **Red-black BSTs. (6 points)**

Consider the following left-leaning red-black BST.



Suppose that you insert the given key into the left-leaning red-black BST above. Which key is in the root node immediately after the insertion?

The insertions are not cumulative—you are inserting each key into the red-black BST above.

insert key	*key in root after insertion*
25	
21	
19	
1	14

6. **Problem identification. (7 points)**

You are applying for a job at a new software technology company. Your interviewer asks you to identify the following tasks as either *possible* (with algorithms and data structures learned in this course), *impossible*, or an *open research problem.* You may use each letter once, more than once, or not at all.

I. Impossible

P. Possible

O. Open

_____ Implement a union-find data type so that all operations (except construction) take logarithmic time in the worst case.

_____ Find two strings in Java that are not equal but have the same `hashCode()`

_____ Design a compare-based algorithm to merge two sorted arrays, each of length $N/2$, into a sorted array of length N that makes $\sim \frac{1}{2}N$ compares in the worst case.

_____ Given a binary heap on N distinct keys, build a BST containing those keys using $\sim 17N$ compares in the worst case.

_____ Given a binary search tree (not necessarily balanced) on N distinct keys, build a binary heap containing those N keys using $\sim 17N$ compares in the worst case.

_____ Design a stable compare-based sorting algorithm that sorts any array of N comparable keys using $\sim N \log_2 N$ compares in the worst case.

_____ Given a sorted array of N keys (not necessarily distinct), find the number of keys equal to a given query key using $\sim 2 \log_2 N$ compares in the worst case.

7. **Leaky stack. (8 points)**

A *leaky stack* is a generalization of a stack that supports adding a string; removing the most-recently added string; and deleting a random string, as in the following API:

`public class LeakyStack`

	`LeakyStack()`	*create an empty randomized stack*
`void`	`push(String item)`	*push the string on the randomized stack*
`String`	`pop()`	*remove and return the string most recently added*
`void`	`leak()`	*remove a string from the stack, uniformly at random*

All operations should take time proportional to $\log N$ *in the worst case, where* N *is the number of items in the data structure.*

For example,

```
LeakyStack stack = new LeakyStack();
stack.push("A");       // A              [ add A ]
stack.push("B");       // A B            [ add B ]
stack.push("C");       // A B C          [ add C ]
stack.push("D");       // A B C D        [ add D ]
stack.push("E");       // A B C D E      [ add E ]
stack.pop();           // A B C D        [ remove and return E ]
stack.push("F");       // A B C D F      [ add F ]
stack.leak();          // A B C F        [ choose D at random; delete D ]
stack.leak();          // A C F          [ choose B at random; delete B ]
stack.pop();           // A C            [ remove and return F ]
stack.pop();           // A              [ remove and return C ]
stack.pop();           //                [ remove and return A ]
```

Give a crisp and concise English description of your data structure. Your answer will be graded on correctness, efficiency, and clarity.

(a) Declare the instance variables for your data structure. For example, here are the declaration for a data type that contains a linear-probing hash table (string keys and integer values) along with an integer:

```
private LinearProbingHashST<String, Integer> st;
private int N;
```

(b) Brief describe how to implement each of the operations, using either prose or code.

- `void push(String item)`:

- `String pop()`

- `void leak()`:

8. **Largest common item. (8 points)**

Given an N-by-N matrix of real numbers, find the largest number that appears (at least) once in each row (or report that no such number exists).

9	6	3	8	5
3	5	1	6	8
0	7	5	3	5
3	5	7	8	6
4	3	5	7	9

The running time of your algorithm should be proportional to $N^2 \log N$ in the worst case. You may use extra space proportional to N^2.

(a) Give a crisp and concise English description of your algorithm.
Your answer will be graded on correctness, efficiency, and clarity.

(b) What is the order of growth of the worst-case running time of your algorithm as a function of N? Circle your answer.

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