COS 226 Written Exam 1 Fall 2016

There are ten questions on this exam, weighted as indicated at the bottom of this page. There is one question per lecture, numbered corresponding to the lectures, *not in order of difficulty*. If a question seems difficult to you, skip it and come back to it.

Policies. The exam is closed book, though that you are allowed to use a single-page one-sided hand-written cheatsheet. No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided. You will have 80 minutes to complete the test. **This exam is preprocessed by computer.** If you use pencil (and eraser), write darkly. Fill in circles *completely* when asked. Write all answers inside the designated rectangles. Do not write on the corner marks.

This page. *Print your name, login ID, and precept number on this page* (now), and write out and sign the Honor Code pledge.

Discussing this exam. As you know, discussing the contents of this exam before solutions have been posted is a serious violation of the Honor Code.

Name

Login

Precept

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

_____

_____

[*copy the pledge here*]

_____

[*signature*]

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	TOTAL

/8	/8	/7	/7	/7	/9	/7	/6	/8	/8	/75

Q1. Union-Find (8 points). Given 1 billion (*N*) elements, suppose that a union-find client performs *N* `union()` operations, then *N* `find()` operations (one for each element). Now consider the *average* cost of these *N* `find()` operations (the total number of array accesses, divided by *N*). In each row, fill in all circles that correspond to a true statement about this average cost for the given algorithm. This question is worth 1 point for each correct answer, but 1 point will be subtracted for each incorrect answer.

	Quick-find	**Quick-union**	**Weighted quick-union**
Cannot be larger than 100.	○	○	○
Could be less than 40.	○	○	○
Could be larger than 1 million.	○	○	○
Could be larger than 100 million.	○	○	○

Q2. Analysis of Algorithms (8 points). Consider the following tables, which give experimental running times in seconds for four programs **A**, **B**, **C**, and **D** for various values of the input size *N*.

A

N	*running time*
1,000	21
2,000	80
4,000	325
8,000	1,275

B

N	*running time*
100	2
1,000	25
10,000	260
100,000	2,600

C

N	*running time*
100	3
1,000	36
10,000	400
100,000	4,500

D

N	*running time*
1,000	2
3,000	17
9,000	150
27,000	1,360

To the right of each option, mark the one-word hypothesis on the order of growth of the running time that best explains the given experimental evidence.

	linear	*linearithmic*	*quadratic*	*cubic*
A	○	○	○	○
B	○	○	○	○
C	○	○	○	○
D	○	○	○	○

Q3. Stacks and queues (7 points). In each square at right, write the letter corresponding to the best match among the terms at left. In a correct answer, three letters will be unused; the other seven appear once each.

		averaging cost over multiple operations	☐
A	**iterator**	*a language mechanism that enables use of the same code for multiple types of data*	☐
B	**underflow**		
C	**amortization**	*an unordered collection*	☐
D	**loitering**		
E	**box**	*when an unused memory reference cannot be reclaimed by the system*	☐
F	**allocation**		
G	**resizing**	*object version of a primitive type*	☐
H	**bag**		
I	**wrapper**	*code that keeps track of the process of returning each item in a collection to a client*	☐
J	**generics**		
		a way to handle overflow in stacks and queues	☐

Q4. Elementary sorts (7 points). For each permutation below, fill in the circle corresponding to a true statement. This question is worth 1 point for each correct answer, but 1 point will be subtracted for each incorrect answer.

	more than 15 inversions	*3-sorted*	*could occur exactly halfway through selection sort*
1 2 3 4 5 6 7 8 9	○	○	○
9 8 7 6 5 4 3 2 1	○	○	○
1 2 4 3 5 6 7 8 9	○	○	○
1 5 6 7 8 2 3 4 9	○	○	○
7 4 1 8 5 2 9 6 3	○	○	○

Q5. Mergesort (7 points).

A. (3 points) Consider the following implementation of Mergesort, adapted from the text:

```
public class Merge
{
   private static Comparable[] aux;

   private static void merge(Comparable[] a, int lo, int mid, int hi)
   { /* Merge implementation */ }

   private static void sort(Comparable[] a, Comparable[] aux,
                            int lo, int hi)
   { /* Sort implementation */ }

   public static void sort(Comparable[] a)
   {
      /* MISSING LINE OF CODE */
      sort(a, aux, 0, a.length - 1);
   }
}
```

In the box below, write the *one line of Java code* that is missing from the public `sort()`.

B. Fill in a circle on each line to indicate whether each statement is **True** or **False**.

	True	**False**
Mergesort is stable.	○	○
To sort an array of *N* elements, standard mergesort implementations need extra space proportional to *N*.	○	○
Bottom-up mergesort does *not* require the use of extra space.	○	○
Mergesort can be improved by using a different method for small subarrays.	○	○

Q6. Quicksort (9 points). Answer these questions about fully sorting an array using quicksort with 3-way partitioning.

A. (3 points) Suppose that the input is a randomly-ordered array with $N = 4M$ elements in total, having $2M$ occurrences of each of two different values. Fill in the one circle on each row that best describes the number of compares used in each case.

	~2*M*	~3*M*	~4*M*	~5*M*	~6*M*
Best case	○	○	○	○	○
Average case	○	○	○	○	○
Worst case	○	○	○	○	○

B**. (6 points)** Suppose that the input is a randomly-ordered array with $N = 9M$ elements having $3M$ occurrences of each of *three* different values. Fill in the one circle on each row that best describes the number of compares used in each case.

	~9*M*	~12*M*	~15*M*	~16*M*	~17*M*	~18*M*	~21*M*	~24*M*	~27*M*
Best case	○	○	○	○	○	○	○	○	○
Average case	○	○	○	○	○	○	○	○	○
Worst case	○	○	○	○	○	○	○	○	○

Q7. Priority queues (7 points). In each part of this question, the top row gives the contents of an array of length 10 representing a binary heap (with array entries not in the heap left blank) and the bottom row gives the results of an operation on that heap. Fill in the squares with black outlines. You may fill in all the squares and use the blank space for scratch, but only squares with black outlines will count for your grade.

A.

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
heap-ordered array		W	D	M	C	A	E	F		
result of removing the maximum										

B.

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
heap-ordered array		W	M	D	E	A	C			
result of inserting G										

C.

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
heap-ordered array		C	B	A						
result of inserting D										

D.

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
heap-ordered array		W	M	F	C	A	E	D	B	
result of removing the maximum										

E.

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
heap-ordered array		W	P	F		A		D	B	
result of removing the maximum			C	F			E			

Q8. BSTs (6 points). To the right of each option, fill in the one circle corresponding to the *height* of the BST produced when the given keys are inserted in the given order into an initially empty BST. For reference, the height of a 1-node BST is 1 and the height of a 2-node BST is 2. You may use the rest of this page for scratch space, but only the marked circles will be used to calculate your grade.

	3	4	5	6
C B F A E D	○	○	○	○
A B C D E F	○	○	○	○
F E D C B A	○	○	○	○

Q9. Red-black BSTs (8 points). To the right of each option, fill in the circle corresponding the the *height* of the LLRB BST produced when the given keys are inserted in the given order into an initially empty LLRB BST. You may use the rest of this page for scratch space, but only the marked circles will be used to calculate your grade.

	3	4	5	6
F B D A C E	○	○	○	○
C B F A E D	○	○	○	○
A B C D E F	○	○	○	○
F E D C B A	○	○	○	○

Q10. Hashing (8 points). Suppose that the keys A through H have the following hash values:

key	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**
hash value	5	6	0	8	5	6	7	8	9

Now suppose that the following table results from inserting these keys into an initially empty table using hashing with linear probing.

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
C	**H**	**I**	**D**		**E**	**A**	**F**	**G**	**B**

Fill in all circles in each row corresponding to keys that fits the description:

	A	**B**	**C**	**D**
***could* have been *first* key inserted**	○	○	○	○
***must* have been *last* key inserted**	○	○	○	○
***must* have been inserted *after* H**	○	○	○	○
***must* have been inserted *before* H**	○	○	○	○