COS 226 **Algorithms and Data Structures** **Fall 2017**

Midterm

This exam has 10 questions (including question 0) worth a total of 55 points. You have 80 minutes. This exam is preprocessed by a computer, so please **write darkly** and **write your answers inside the designated spaces.**

Policies. The exam is closed book, except that you are allowed to use a one page cheatsheet (8.5-by-11 paper, one side, in your own handwriting). No electronic devices are permitted.

Discussing this exam. Discussing the contents of this exam before solutions have been posted is a violation of the Honor Code.

This exam. Do not remove this exam from this room. Write your name, NetID, and the room in which you are taking the exam in the space below. Mark your precept number. Also, write and sign the Honor Code pledge. You may fill in this information now.

Name:

NetID:

Exam room:

Precept: P01 ◯ P02 ◯ P03 ◯ P03A ◯ P04 ◯ P05 ◯ P06 ◯

"I pledge my honor that I will not violate the Honor Code during this examination."

Signature

0. **Initialization. (1 point)**

   In the space provided on the front of the exam, write your name, NetID, and the room in which you are taking the exam; mark your precept number; and write and sign the Honor Code pledge.

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

   Suppose that you implement a weighted quick-union data type using the following parent-link representation:

```
public class WeightedQuickUnion {
    private final int n;             // number of elements
    private final Node[] nodes;      // array of n nodes

    private static class Node {
        private int size = 1;        // subtree size
        private Node parent = null;  // parent link
    }

    // construct a weighted quick-union data structure with n elements
    public WeightedQuickUnion(int n) {
        this.n = n;
        this.nodes = new Node[n];
    }

    ...
}
```

   Using the 64-bit memory cost model from lecture and the textbook, how much memory does a `WeightedQuickUnion` object use as a function of the number of elements n? Use tilde notation to simplify your answer.

   ~ $\boxed{}$ bytes

2. **Five sorting algorithms. (5 points)**

The column on the left is the original input of strings to be sorted; 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 five sorting algorithms listed below. Match each algorithm by writing its number in the box under the corresponding column. Use each number exactly once.

0	prim	heap	flip	edge	flip	miss	edge
1	left	left	heap	find	left	load	find
2	load	load	java	flip	load	loop	flip
3	push	lazy	left	hash	loop	list	hash
4	sink	node	load	heap	miss	left	heap
5	time	hash	loop	java	null	lazy	java
6	loop	loop	miss	lazy	prim	heap	lazy
7	flip	flip	null	left	push	java	left
8	null	null	prim	list	sink	flip	list
9	miss	miss	push	load	swim	hash	load
10	trie	edge	rank	loop	time	edge	loop
11	swim	find	sink	miss	trie	find	miss
12	java	java	sort	time	find	node	node
13	sort	list	swim	sort	heap	null	null
14	rank	prim	time	rank	java	prim	prim
15	heap	rank	trie	sink	list	push	push
16	list	sort	list	null	rank	rank	rank
17	find	swim	find	swim	sort	sink	sink
18	tree	tree	tree	tree	tree	sort	sort
19	edge	trie	edge	prim	edge	swim	swim
20	hash	time	hash	push	hash	time	time
21	node	sink	node	node	node	tree	tree
22	lazy	push	lazy	trie	lazy	trie	trie
23	type	type	type	type	type	type	type
	0						6

(0) Original input

(1) Selection sort

(2) Insertion sort

(3) Mergesort (*top-down*)

(4) Heapsort

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

(6) Sorted

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

Consider an array that contains two successive copies of the integers 1 through n, in ascending order. For example, here is the array when $n = 8$:

```
1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8
```

Note that the length of the array is $2n$, not n.

(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.

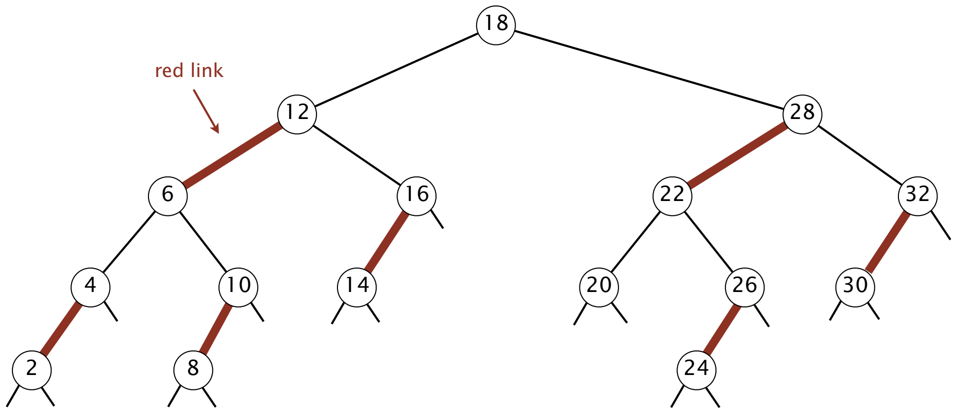
$\sim$ $\boxed{}$ compares

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

$\sim$ $\boxed{}$ compares

4. **Red–black BSTs. (6 points)**

Suppose that you insert the key 31 into the following left-leaning red–black BST:



Which of the following *color flips* and *rotations* result? Mark all that apply.

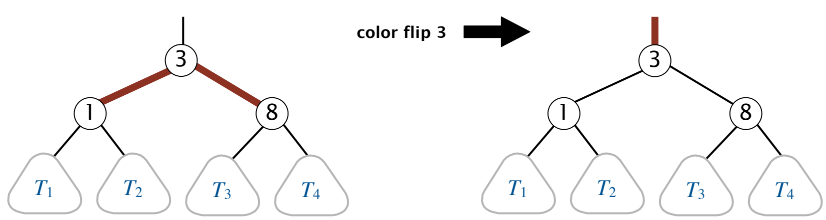
Color flips:

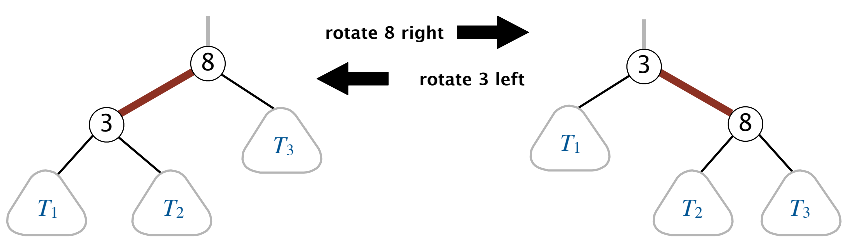
☐ 18 ☐ 22 ☐ 26 ☐ 28 ☐ 30 ☐ 31 ☐ 32

Rotations:

☐ 18 left ☐ 18 right ☐ 28 left ☐ 28 right ☐ 30 left ☐ 30 right ☐ 32 left ☐ 32 right ☐ 33 left ☐ 33 right

Examples of color flips and rotations (for reference):





5. **Hash tables. (5 points)**

Suppose that the following keys are inserted into an initially empty linear-probing hash table, but not necessarily in the order given,

key	*hash*
B	3
G	5
I	3
N	5
O	1
P	5
R	4

and it results in the following hash table:

0	1	2	3	4	5	6
P	**R**	**O**	**B**	**I**	**N**	**G**

For each description at left, mark all keys that apply. Assume that the initial size of the hash table is 7 and that it neither grows nor shrinks.

	B	**G**	**N**	**O**	**R**
could have been first key inserted	☐	☐	☐	☐	☐
could have been last key inserted	☐	☐	☐	☐	☐
must have been inserted before I	☐	☐	☐	☐	☐
must have been inserted after I	☐	☐	☐	☐	☐

6. **Programming assignments. (6 points)**

Answer the following questions about the COS 226 programming assignments.

(a) Consider implementing `Percolation` using a union–find implementation that supports *union* and *find* in $\sqrt{m}$ time per operation on a set of m elements. What is the order of growth of the running time to perform one percolation experiment (open random sites until the system percolates) on an n-by-n system? Mark the best answer.

$n^{1/2}$	n	$n^{3/2}$	n^2	$n^{5/2}$	n^3	$n^{7/2}$	n^4
○	○	○	○	○	○	○	○

(b) Consider implementing a `Deque` using a singly linked list, storing the first item in the deque in the first node in the linked list (and the last item in the deque in the last node). Which of the following operations could be implemented to run in constant time in the worst case? Mark all that apply.

`addFirst()`	`addLast()`	`removeFirst()`	`removeLast()`	`isEmpty()`
☐	☐	☐	☐	☐

(c) Consider nearest-neighbor search in a 2d tree with $n \geq 1$ points. Which of the following are features of the algorithm and pruning rule specified in the assignment? Mark all that apply.

☐ Guarantees to return a nearest neighbor.

☐ Guarantees $\log n$ time per operation in the worst case.

☐ Guarantees $\log n$ time per operation in the worst case if the 2d tree is balanced.

☐ Achieves $\log n$ time per operation on inputs likely to arise in practice.

7. **Data structure and algorithm properties. (6 points)**

Match each quantity on the left by writing the letter of the best matching term at right. You may use each letter more than once or not at all. Assume each algorithm is the standard version, presented in the textbook.

☐ *Maximum* number of key compares to binary search for a key in a sorted array.

☐ *Maximum* number of key compares to perform a DELETE-MAX operation in a binary heap containing n keys.

☐ *Minimum* height of a weighted quick-union tree with n elements.

☐ *Minimum* height of a binary search tree with n keys.

☐ *Minimum* number of black links on path from the root to a null link in a left-leaning red–black BST containing n keys.

☐ *Maximum* number of times an array can be resized (doubled or halved) during a sequence of n push and pop operations (starting from an empty data structure) in a resizing-array implementation of a stack.

A. *constant*

B. $\sim \frac{1}{2} \log_2 n$

C. $\sim \log_3 n$

D. $\sim \log_e n$

E. $\sim \log_2 n$

F. $\sim 2 \log_e n$

G. $\sim 2 \log_2 n$

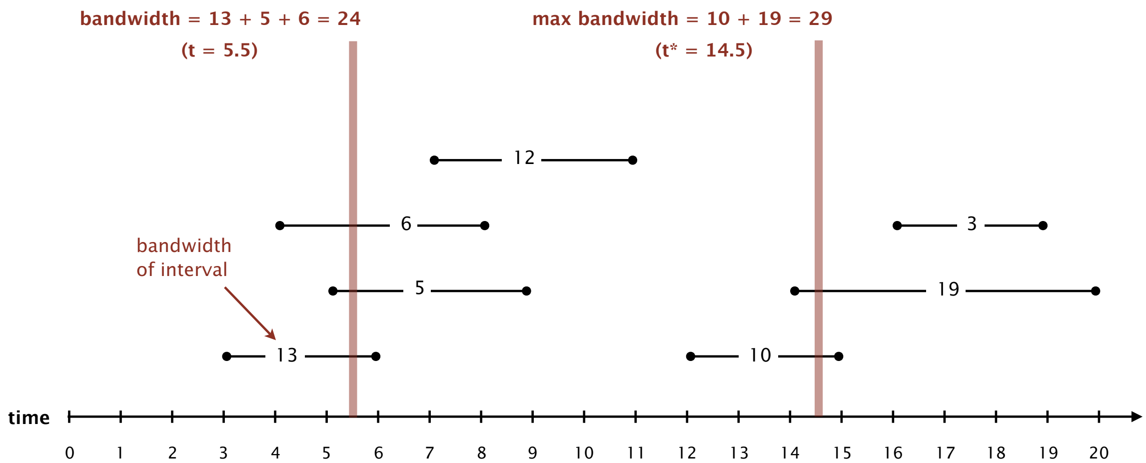
H. $\sim 4.311 \log_e n$

I. *linear*

8. **Largest bandwidth. (8 points)**

   Given n time intervals (l_i, r_i) and associated bandwidths $b_i > 0$, the *bandwidth demand* at time t is the sum of the bandwidths of all intervals that contain t. Design an algorithm to find a time t^* that has the *largest* bandwidth demand.

   In the example below, there are $n = 7$ time intervals. The largest bandwidth demand is 29 and occurs at time $t^* = 14.5$: the bandwidth of the interval $(12, 15)$ is 10 and the bandwidth of the interval $(14, 20)$ is 19.



   Give a crisp and concise English description of your algorithm in the space below.

   Your answer will be graded for correctness, efficiency, and clarity. For full credit, the worst-case running time must be proportional to $n \log n$*. You may assume that all of the interval endpoints are distinct.*

9. **Data structure design. (8 points)**

   Create a data type `UniQueue` that implements a FIFO queue of strings with *no duplicates*, according to the following API:

public class UniQueue		
	`UniQueue()`	*create an empty uni-queue*
`void`	`enqueue(String s)`	*add the string to the uni-queue (if it is not already in the uni-queue)*
`String`	`dequeue()`	*remove and return the string least recently added to the uni-queue*

That is, if a duplicate key is added to the uni-queue, it is ignored. Here is an example:

```
UniQueue queue = new UniQueue();
queue.enqueue("ant");          // [ "ant" ]
queue.enqueue("bear");         // [ "ant", "bear" ]
queue.enqueue("cat");          // [ "ant", "bear", "cat" ]
queue.enqueue("bear");         // [ "ant", "bear", "cat" ]
queue.enqueue("dog");          // [ "ant", "bear", "cat", "dog" ]
queue.dequeue();               // [ "bear", "cat", "dog" ]
queue.dequeue();               // [ "cat", "dog" ]
queue.enqueue("bear");         // [ "cat", "dog", "bear" ]
```

Your answer will be graded for correctness, efficiency, and clarity.

(a) Declare the instance variables for your `UniQueue` data type. You may declare nested classes or use any of the data types that we have considered in this course.

```
public class UniQueue {





}
```

(b) Briefly describe how to implement `enqueue()` and `dequeue()`, using either crisp and concise prose or code.

```
public void enqueue(String s) {



}
```

Assume that the argument to `enqueue()` *is not* `null`.

```
public String dequeue() {



}
```

Do not worry about underflow.

(c) What is the order of growth of each operation as a function of the number of items n in the data structure? Mark whether it is an amortized bound and/or makes the uniform hashing assumption.

operation	*order of growth*	*amortized bound*	*uniform hashing assumption*
`enqueue()`	____________	☐	☐
`dequeue()`	____________	☐	☐