COS 226 Algorithms and Data Structures Fall 2018

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 when grading, 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). Electronic devices are prohibited.

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. In the space provided, write your name and NetID. Also, mark your exam room and precept number. Finally, write and sign the Honor Code pledge. You may fill in this information now.

Name:

NetID:

Course: [✓ COS 226]

Exam room: LTL 003 ○ JAD A08 ○ JAD A09 ○ LEW 122 ○ LEW 134 ○ Other ○

Precept: P01 ○ P01A ○ P02 ○ P02A ○ P03 ○ P03A ○ P04 ○ P05 ○

"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 and NetID; mark your exam room and precept number; write and sign the Honor Code pledge.

1. **Memory. (4 points)**

   Suppose that you implement a symbol table (with `int` keys and `double` values) using a binary search tree with the following data type:

```
public class BinarySearchTree {
    private Node root;            // root of BST
    private int n;                // number of nodes in BST

    private static class Node {
        private Node left;        // left subtree
        private Node right;       // right subtree
        private int key;          // symbol table key
        private double value;     // symbol table value
    }
    ...
}
```

   Using the 64-bit memory cost model from lecture and the textbook, how much memory does a `BinarySearchTree` object use as a function of the number of key–value pairs n? Count all referenced memory, including the keys and values. Use tilde notation to simplify your answer.

   ~ $\boxed{}$ bytes

   Recall: the extra 8-byte overhead for inner classes is needed only for objects from non-static nested classes.

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

   The leftmost column contains an array of 24 integers to be sorted; the rightmost column contains the integers in sorted order; the other columns are the contents of the array 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 once.

63	44	11	19	11	81	11
21	21	19	21	19	79	19
19	19	21	32	21	63	21
32	32	25	45	29	60	25
45	45	29	60	31	71	29
60	60	31	63	32	29	31
31	31	32	11	44	48	32
79	25	44	31	45	45	44
48	48	45	48	48	52	45
11	11	48	71	60	50	48
71	50	50	79	63	67	50
88	52	52	88	71	21	52
29	29	63	29	79	25	60
99	63	99	99	88	31	63
89	89	89	89	89	19	67
44	99	79	44	99	44	71
86	86	86	86	86	11	79
52	88	88	52	52	32	81
92	92	92	92	92	86	86
50	71	71	50	50	88	88
25	79	60	25	25	89	89
67	67	67	67	67	92	92
93	93	93	93	93	93	93
81	81	81	81	81	99	99
0						6

(0) Original array

(1) Selection sort

(2) Insertion sort

(3) Mergesort (*top-down*)

(4) Heapsort

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

(6) Sorted array

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

Consider an *organ-pipe array* that contains two copies of the integers 1 through n, first in ascending order, then in descending order. For example, here is the array when $n = 8$:

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

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.

~ $\boxed{}$ 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.

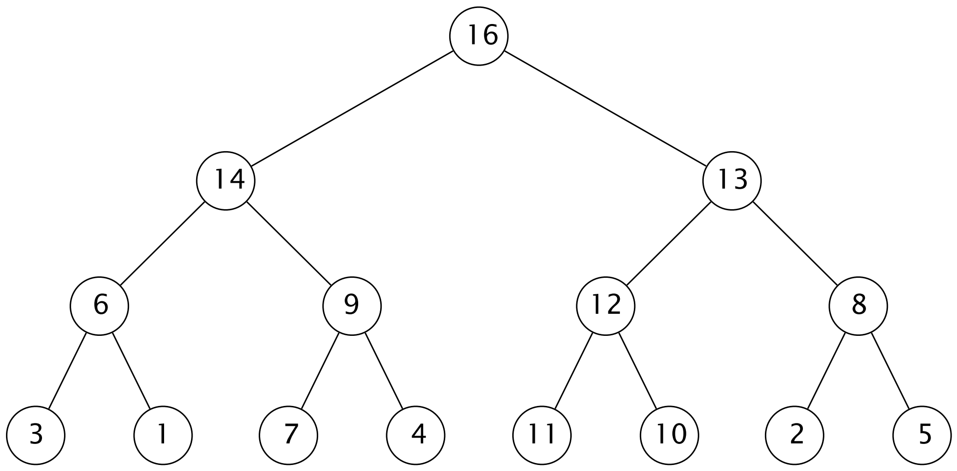
~ $\boxed{}$ compares

(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.

~ $\boxed{}$ compares

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

Consider the following maximum-oriented binary heap.



(a) Suppose that you *insert* the key `15` into the binary heap.
Mark all keys that will be involved in a *compare*.

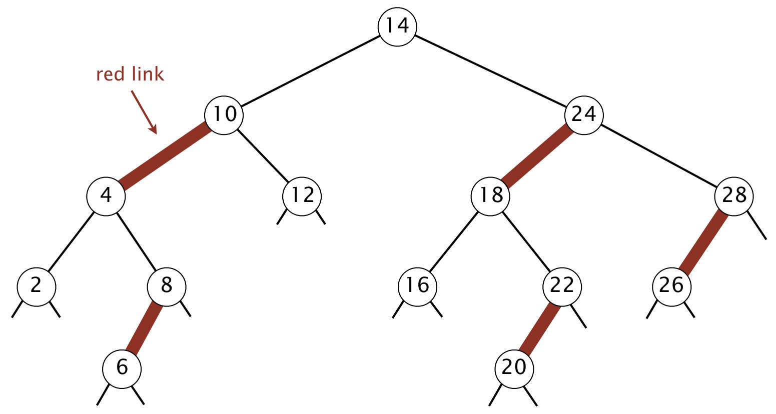
☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	■	☐
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

(b) Suppose that you *delete-the-maximum* key from the original binary heap.
Mark all keys that will be involved in a *compare*.

☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐		☐
1	2	3	4	5	6	7	8	9	10	11	12	13	14		16

5. **Red–black BSTs. (6 points)**

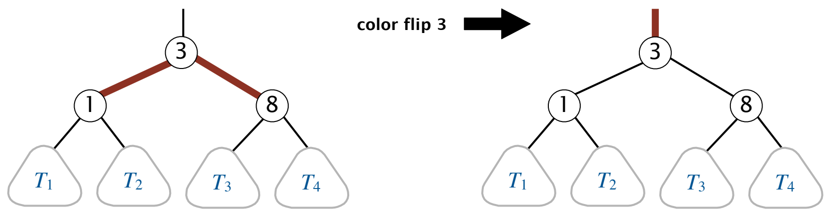
Suppose that you insert the key 23 into the following left-leaning red–black BST:

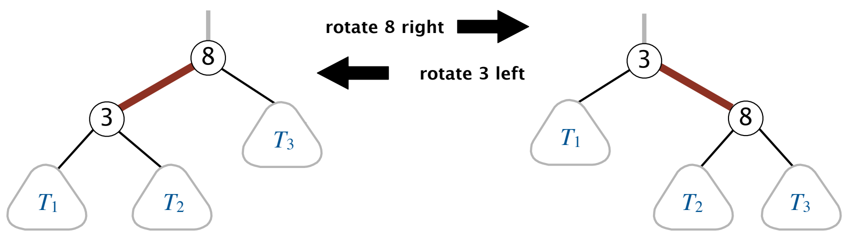


Give the sequence of 5 elementary operations (color flips and rotations) that result.

	operation 1	*operation* 2	*operation* 3	*operation* 4	*operation* 5
key	**22**				
color flip	●	○	○	○	○
rotate left	○	○	○	○	○
rotate right	○	○	○	○	○

Examples of color flips and rotations (for reference):





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

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

☐	*Expected* number of array accesses to perform one computational experiment in an n-by-n percolation system (i.e., repeatedly open random sites under the system percolates) using *quick find.*	A. *constant*
		B. $\log n$
☐	*Maximum* number of array accesses to perform an intermixed sequence of n *enqueue*, *dequeue*, and *sample* operations in an initially empty *randomized queue* that is implemented efficiently with a resizing array.	C. n
		D. $n \log n$
☐	*Maximum* number of compares to *count* the number of matching terms in any *autocomplete query.* Assume the terms are already sorted by query key.	E. n^2
		F. $n^2 \log n$
☐	*Maximum* number of boards objects created to solve an n-by-n slider puzzle using the *A* algorithm* with the Manhattan priority function.	G. n^3
		H. n^4
☐	*Maximum* number of compares to perform an intermixed sequence of n *insert* and *delete-the-max* operations in an initially empty *binary heap.*	I. *exponential*
☐	*Minimum* number of x- and y-coordinate compares to perform n consecutive *insert* operations in an initially empty *2d-tree.* Assume the n points to be inserted are all distinct.	
☐	*Maximum* number of array accesses to perform an intermixed sequence of n *insert* and *search* operations in an initially empty *linear-probing hash table* of capacity $m = 2n$. Do not make any assumptions about the hash function.	

7. **System sort. (5 points)**

The Java 10 system sort uses a combination of *insertion sort*, *Timsort*, and *dual-pivot quicksort*. For each of the following properties, mark each algorithm that possesses that property.

Recall: Timsort is an optimized version of bottom-up mergesort that forms subarrays by identifying natural runs of consecutive ordered elements.

	insertion sort	dual-pivot quicksort	Timsort
Stable.	☐	☐	☐
In-place.	☐	☐	☐
At most $\sim n \log_2 n$ *compares.*	☐	☐	☐
Linear number of compares on arrays with only 3 distinct keys.	☐	☐	☐
Linear number of compares on arrays in ascending order.	☐	☐	☐

8. **Duplicate in two arrays. (8 points)**

   Given two integer arrays `a[]` and `b[]`, find an integer that appears in both arrays (or report that no such integer exists). Let m and n denote the lengths of `a[]` and `b[]`, respectively, and assume that $m \leq n$.

   Here are the performance requirements:

   - *Space:* the amount of extra space (besides `a[]` and `b[]`) must be constant. It is fine to modify `a[]` and `b[]`.
   - *Time:* the order of growth of the running time must be $n \log m$ in the worst case.

   Give a crisp and concise English description of your algorithm in the space below. Your answer will be graded for correctness, efficiency, and clarity. Partial credit for $n \log n$ *time and* $\log n$ *extra space.*

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

Create a Duo data type that supports adding integers to (and deleting integers from) either of two unordered lists (with duplicates allowed) and checking whether any integer appears in both lists. To do so, implement this API:

public class Duo		
	`Duo()`	*create an empty duo data type*
`void`	`addToList1(int x)`	*add the integer to the first list*
`void`	`addToList2(int x)`	*add the integer to the second list*
`void`	`deleteFromList1(int x)`	*delete the integer from the first list*
`void`	`deleteFromList2(int x)`	*delete the integer from the second list*
`boolean`	`hasDuplicate()`	*does any integer appear in both lists?*

Here is an example:

```
Duo duo = new Duo();        //  [ ]              [  ]
duo.addToList1(23);         //  [ 23 ]           [  ]
duo.addToList1(45);         //  [ 23, 45 ]       [  ]
duo.addToList2(56);         //  [ 23, 45 ]       [ 56 ]
duo.addToList2(56);         //  [ 23, 45 ]       [ 56, 56 ]
duo.hasDuplicate();         //  false
duo.addToList2(23);         //  [ 23, 45 ]       [ 56, 56, 23 ]
duo.hasDuplicate();         //  true
duo.deleteFromList2(56);    //  [ 23, 45 ]       [ 56, 23 ]
duo.deleteFromList1(23);    //  [ 45 ]           [ 56, 23 ]
duo.hasDuplicate();         //  false
```

Your answer will be graded for correctness, efficiency, and clarity (but not precise Java syntax). For full credit, each operation must take constant time (subject to standard technical assumptions that we have seen in this course). For most of the credit, implement all operations except deletion.

(a) Declare the Java instance variables for your `Duo` data type using Java code. You may use any of the data types that we have considered in this course (either `algs4.jar` or `java.util` versions).

```
public class Duo {

}
```

(b) Implement `addToList1()`. We'll assume the code for `addToList2()` is symmetric.

```
// add x to the first list
public void addToList1(int x) {

}
```

(c) Implement `hasDuplicate()`.

```
// does any integer appear in both lists?
public boolean hasDuplicate() {











}
```

(d) Implement `deleteFromList1()`. You may assume that the integer `x` appears at least once in the specified list; if `x` appears more than once, delete only one copy. We'll assume the code for `deleteFromList2()` is symmetric.

```
// delete x from the first list
public void deleteFromList1(int x) {











}
```

This page is intentionally blank. You may use this page for scratch work but do not remove it from the exam.

This page is intentionally blank. You may use this page for scratch work but do not remove it from the exam.