COS 226 **Algorithms and Data Structures** **Fall 2024**

Midterm

This exam has 9 questions worth a total of 55 points. You have 80 minutes.

Instructions. This exam is preprocessed by computer. Write neatly, legibly, and darkly. Put all answers (and nothing else) inside the designated spaces. *Fill in* bubbles and checkboxes completely: ● and ■. To change an answer, erase it completely and redo.

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

Honor Code. This exam is governed by Princeton's Honor Code. Discussing the contents of this exam before the solutions are posted is a violation of the Honor Code.

Please complete the following information now.

Name: ☐

NetID: ☐

Exam room: ☐

Precept:

P01	P02	P03	P05	P06	P07	P08	P09	P10
○	○	○	○	○	○	○	○	○

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

☐

Signature

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

   In the spaces provided on the front of the exam, write your name, NetID, and exam room; fill in the bubble of the precept in which you are officially registered; write and sign the Honor Code pledge.

2. **Asymptotics. (6 points)**

   (a) How many times will the following code snippet print '`hello`'? Assume that n is divisible by 100.

```
for (int i = 1; i <= n; i++)
    for (int j = n; j >= i; j--)
        for (int k = 1; k <= n; k = k + n/100)
            System.out.println("hello");
```

   ○ $\sim 200n$ ○ $\sim 50n^2$ ○ $\sim 100n^2$ ○ $\sim \frac{1}{2}n^2 \log_{100} n$ ○ $\sim n^3/100$ ○ $100n^3$

   (b) Which of the following expressions describe the growth of the following function?

$$n \log^2 n + 3n\sqrt{n} - 5n$$

   Fill in all checkboxes that apply.

   ☐ $\sim n\sqrt{n}$ ☐ $\Theta(n \log^2 n)$ ☐ $O(n^3)$ ☐ $O(n\sqrt{n})$ ☐ $O(n)$ ☐ $\Omega(\log n)$

3. **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 letter in the box under the corresponding column. Use each letter exactly once.

49	14	14	48	14	57	14
45	22	22	45	15	56	15
14	45	24	14	22	49	22
97	49	37	15	24	45	24
22	57	40	22	37	40	37
57	97	44	44	40	37	40
40	24	45	40	44	48	44
24	40	48	24	45	14	45
65	63	49	37	48	44	48
93	65	57	49	49	24	49
63	68	63	63	56	15	56
68	93	65	68	57	22	57
75	75	68	75	75	58	58
48	48	75	93	65	60	60
37	37	93	65	93	63	63
44	44	97	57	68	64	64
73	73	73	73	73	65	65
15	15	15	97	97	68	68
58	58	58	58	58	73	73
64	64	64	64	64	75	75
88	88	88	88	88	78	78
60	60	60	60	60	88	88
56	56	56	56	63	93	93
78	78	78	78	78	97	97
A						G

A. Original array

B. Selection sort

C. Insertion sort

D. Mergesort

E. Quicksort (*standard, no shuffle*)

F. Heapsort

G. Sorted array

4. **Data structure invariants. (6 points)**

For each data structure and state below, determine whether it is possible for the state to arise with a sequence of operations on the associated data type.

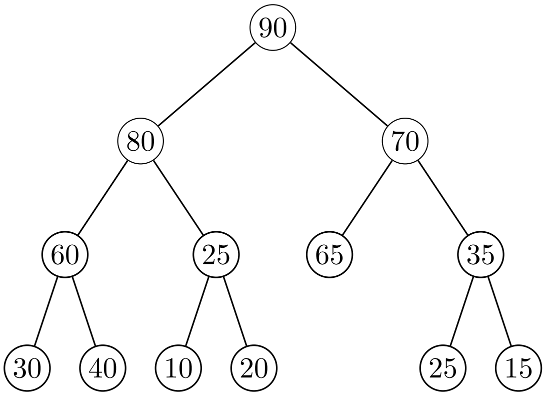
(a) A *weighted quick-union* data structure with the following `parent[]` array:

0	1	2	3	4	5	6	7	8	9
5	6	0	4	0	5	0	5	6	7

○ Possible.

○ Impossible.

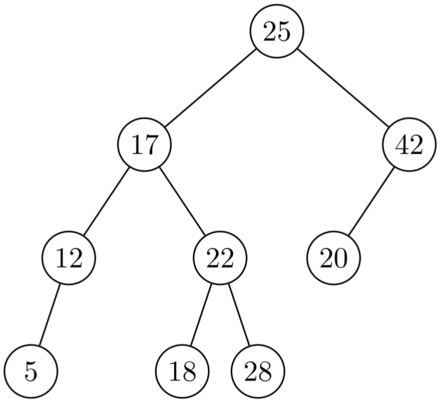
(b) A *maximum-oriented binary heap* corresponding to the following binary tree:



○ Possible.

○ Impossible.

(c) A *binary search tree* with the following (integer) keys and links:

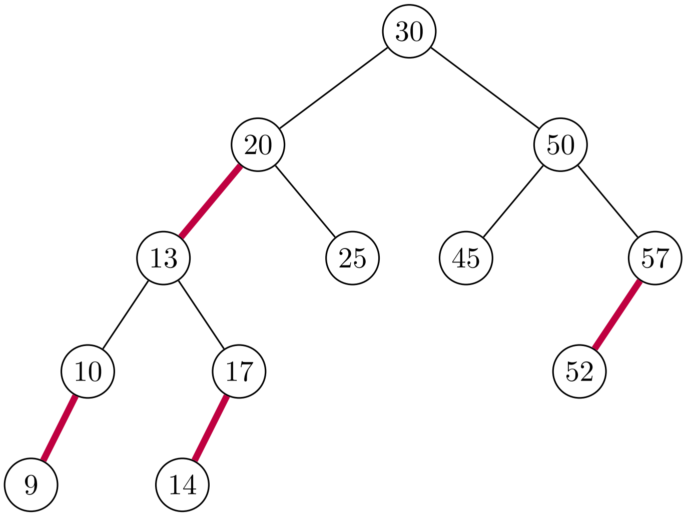


○ Possible.

○ Impossible.

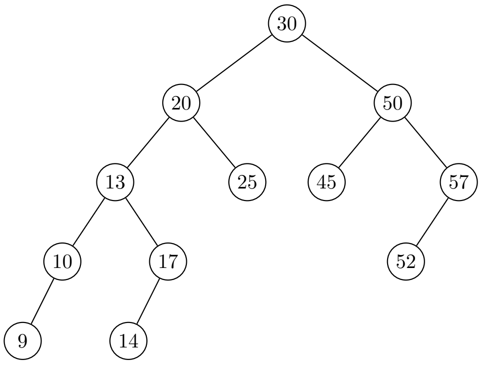
5. **Balanced search trees. (6 points)**

Consider the following *left-leaning red-black BST*:

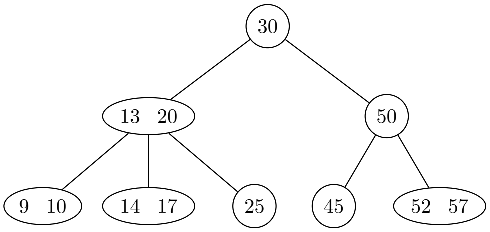


(a) Which of the following *2-3 trees* corresponds to the left-leaning red-black tree above?

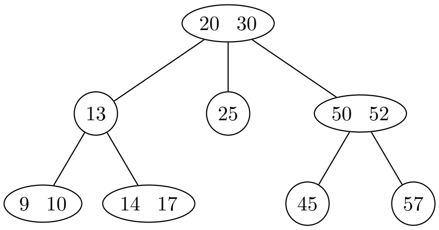
○ ○ ○ ○
A B C D



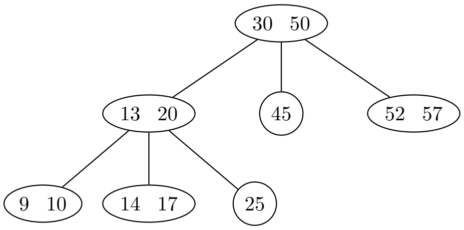
A.



B.



C.



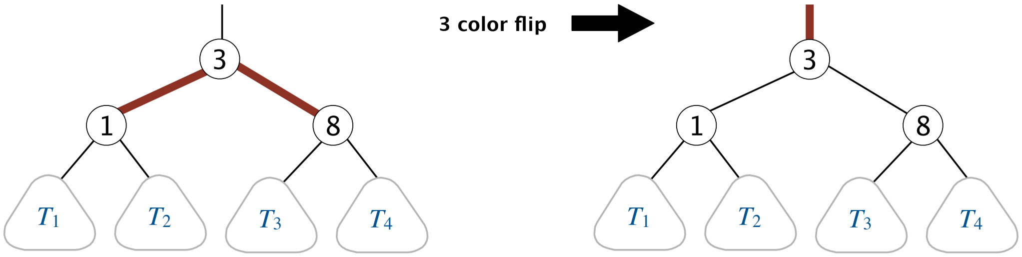
D.

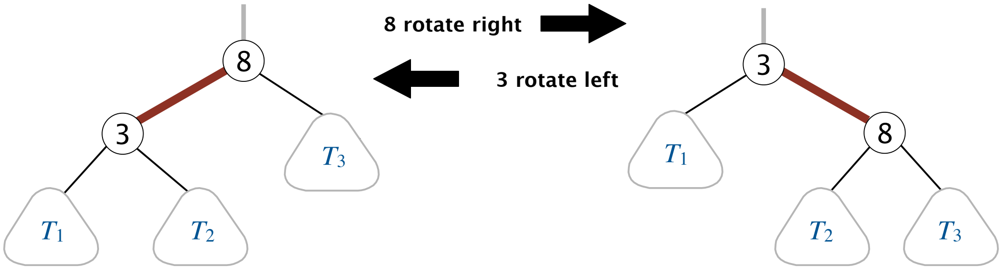
(b) Suppose that you *insert* the key 18 into this red-black BST.
Give the sequence of 4 elementary operations (color flips and rotations) that occur during the insertion.

Fill in all checkboxes that apply.

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

Examples of color flips and rotations (for reference):





6. **Queues. (5 points)**

A *self-printing queue* is a queue of integers, implemented as a singly linked list, which prints the contents of the queue to standard output after every *three* (enqueue or dequeue) operations. For instance, the sequence `enqueue(0), dequeue(), enqueue(0)` prints 0.

(a) What does the following sequence of operations on an initially empty self-printing queue print?

`enqueue(0), enqueue(1), dequeue(), enqueue(2), enqueue(3), dequeue(), enqueue(4), enqueue(5), dequeue(), enqueue(6), enqueue(7), dequeue()`

○ 1 2 3 3 4 5 4 5 6 7.

○ 0 0 2 0 2 4 0 2 4 6.

○ 4 5 6 7.

○ 0 1 2 3.

○ 1 2 2 3 4 5.

○ 0 2 0 2 4 5.

(b) What is the worst-case running time of an `enqueue()` operation on a self-printing queue with n elements?

○	○	○	○	○	○
$\Theta(1)$	$\Theta(\log n)$	$\Theta(\sqrt{n})$	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n^2)$

(c) What is the *amortized, per-operation* running time of n `enqueue()` and `dequeue()` operations on an initially empty self-printing queue? Recall that this quantity is defined as the *worst-case* running time for *any* intermixed sequence of n `enqueue()` and `dequeue()` operations starting from an empty self-printing queue, divided by n.

○	○	○	○	○	○
$\Theta(1)$	$\Theta(\log n)$	$\Theta(\sqrt{n})$	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n^{3/2})$

7. **Analysis of algorithms and sorting. (8 points)**

Let n be a power of 2. Consider an array structured as shown below.

$$2n\ 2n\ \ldots\ 2n\ 0\ 1\ 2\ 2\ 4\ 4\ 4\ 4\ 8\ 8\ 8\ 8\ 8\ 8\ 8\ 8\ \ldots\ n\ n\ \ldots\ n$$

The first half of the array contains $2n$ repetitions of the value $2n$. The second half contains $2n$ numbers: one number 0, one number 1, two repetitions of the number 2, four repetitions of the number 4, eight repetitions of the number 8, sixteen repetitions of the number 16, and so on, until the number n appears n times.

For example, here is the array when $n = 4$:

```
8  8  8  8  8  8  8  8  0  1  2  2  4  4  4  4
```

How many *compares* does each sorting algorithm (standard algorithm, from the textbook) make as a function of n in the worst case? *Note that the length of the array is* $4n$ *and not* n*.*

We write log to denote the base-2 logarithm, and recall that $\log(ab) = \log a + \log b$.

For each sorting algorithm, fill in the best matching bubble.

(a) *Selection sort*

○ $\sim \frac{1}{2}n^2$ ○ $\sim n^2$ ○ $\sim 2n^2$ ○ $\sim 4n^2$ ○ $\sim 8n^2$ ○ $\sim 16n^2$

(b) *Insertion sort*

○ $\sim \frac{1}{2}n^2$ ○ $\sim n^2$ ○ $\sim 2n^2$ ○ $\sim 4n^2$ ○ $\sim 8n^2$ ○ $\sim 16n^2$

(c) *Mergesort*

○ $\sim n \log n$

○ $\sim 2n \log n$

○ $\sim 4n \log n$

○ $\sim \frac{9}{2} n \log n$

○ $\sim 8n \log n$

○ $\sim 16n \log n$

(d) *3-way Quicksort.* Assume that the shuffle places the array in *descending order*:

$$2n\ 2n\ \ldots\ 2n\ n\ n\ \ldots\ n\ \ldots\ 8\ 8\ 8\ 8\ 8\ 8\ 8\ 8\ 4\ 4\ 4\ 4\ 2\ 2\ 1\ 0$$

○ $\Theta(n)$

○ $\Theta(n \log n)$

○ $\Theta(n^2 / \log n)$

○ $\Theta(n^2)$

○ $\Theta(n^2 \log n)$

○ $\Theta(n^4)$

8. **Algorithm design. (10 points)**

A *mountain-like* array is an array of length $2n$ whose the first half (the first n elements) is sorted in ascending order, and the second half (the last n elements) is sorted in descending order.

Here is an example of a mountain-like array of integers with $n = 6$.

```
4  7  8  11  18  20  |  31  7  7  4  2  1
```

(a) Design an algorithm to sort a mountain-like array of integers.

Full credit: The algorithm must run in $O(n)$ time in the worst case.

Partial credit (at least half): You may assume that the input is an *unbalanced* mountain-like array, where the first $2n-100$ integers are in ascending order and the last 100 integers are in descending order. The algorithm must run in $O(n)$ time in the worst case.

Choose one option to attempt:

○ Full-credit solution (mountain-like array).

○ Partial credit solution (*unbalanced* mountain-like array).

In the space provided, give a concise English description of your algorithm for solving the problem. You may use any of the algorithms that we have considered in this course (e.g., lectures, precepts, textbook, assignments) as subroutines. If you modify such an algorithm, be sure to describe the modification. Feel free to use code or pseudocode to improve clarity.

(b) Consider a method with the following signature that takes an array of $2n$ `Comparable` elements as input and rearranges it into a mountain-like array. Note that multiple mountain-like configurations can be formed from the same array, and the method can return any one of these possible arrangements.

```
void makeMountainLike(Comparable[] a)
```

Is it possible to implement this method in $O(n)$ *time?*

○ Yes.

○ No.

If your answer is yes, give a concise English description of your algorithm. If your answer is no, provide a brief explanation of the reason why.

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

Design a data structure named `Tournament` that stores and retrieves player information. The `Tournament` class should support three operations:

- `insert()` adds player details,
- `getScore()` retrieve a player's score, and
- `getLead()` returns the player with the highest score.

The implementation should follow the API provided below.

public class Tournament

`public Tournament()`	*creates an empty collection*
`void insert(String name, int score)`	*adds a player to the collection given their name and score*
`int getScore(String name)`	*returns the player's score given their name*
`String getLead()`	*returns the name of the player with the highest score*

For simplicity, you may assume that there are *no duplicate* names or scores in the collection at any time.

Example. Here is a small example sequence of operations.

```
Tournament t = new Tournament();   //  [ ]
t.insert("Roger", 219);    // [ ("Roger", 219) ]
t.insert("John", 38);      // [ ("Roger", 219), ("John", 38) ]
t.getScore("John");        // returns 38
t.getLead();               // returns "Roger"
t.insert("Freddie", 467); // [ ("Roger", 219), ("John", 38), ("Freddie", 467) ]
t.getLead();               // returns "Freddie"
t.insert("Brian", 189);    // [ ("Roger", 219), ("John", 38),
                           //   ("Freddie", 467), ("Brian", 189) ]
t.getScore("Roger");       // returns 219
t.getLead();               // returns "Freddie"
```

Note: the pairs in square brackets denote players currently in the collection, but the API does not require you to store them in any particular order.

Performance requirements. Denote by n the current number of players in the collection.

Full credit:

- The constructor must take $\Theta(1)$ time.
- The `insert()`, `getScore()`, and `getLeader()` methods must each take $O(\log n)$ time in the worst case.
- The data type must use $O(n)$ extra space.

Partial credit (at least 30%): `getLeader()` takes $O(n)$ time in the worst case. `insert()` and `getScore()` have the same performance requirements as the full credit option.

(a) Specify the instance variables (along with any supporting nested classes) that you would use to implement `Tournament`. You may use code or pseudocode to improve clarity. You may use any of the data types that we have considered in this course (either `algs4.jar` or `java.util` versions). If you make any modifications to these data types, describe them.

(b) Give a concise English description of your algorithm for implementing `insert()`. You may use code or pseudocode to improve clarity.

(c) Give a concise English description of your algorithm for implementing `getScore()`. You may use code or pseudocode to improve clarity.

(d) Give a concise English description of your algorithm for implementing `getLeader()`. You may use code or pseudocode to improve clarity.

The running time of your implementation of `getLeader()` is $\Theta(\qquad\qquad)$

This page is intentionally blank. You may use this page for scratch work.