COS 226 **Algorithms and Data Structures** **Fall 2026**

Midterm

This exam has 9 questions worth a total of 60 points. You have 80 minutes to complete it.

Instructions. This exam is preprocessed by computer. Write neatly and legibly, using a dark pencil or pen. Write only your answers in the designated spaces; anything outside these spaces will not be graded. *Fill in* bubbles and checkboxes completely: ● and ■. To change an answer, erase it completely and rewrite it.

Resources. The exam is closed book, except that you may use a one-page reference sheet (8.5-by-11 paper, one side, handwritten by you). No electronic devices are permitted.

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

Please complete the following information now.

Name:

NetID:

Exam room: ◯ Friend 101 ◯ McCosh 60 ◯ Other

Precept:

P01	P01A	P02	P02A	P03	P05
◯	◯	◯	◯	◯	◯

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

Signature

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

   In the designated spaces on the front page of the exam:

   - Write your name and NetID.
   - Fill in the bubbles for your exam room and precept.
   - Write out and sign the Honor Code pledge.

2. **Memory. (6 points)**

   Consider this Java implementation of a *2–3 tree* with `String` keys and `double` values:

```
public class TwoThreeTree {
   private Node root;

   private static class Node {
      private String key1, key2;         // the one or two keys
      private double val1, val2;         // the one or two values
      private Node left, middle, right;  // the two or three subtrees
   }
   ...
}
```

   (a) Using our 64-bit memory cost model, how much memory does each `Node` object use? Include all memory for the `Node` object itself, including references, but exclude the memory for referenced objects.

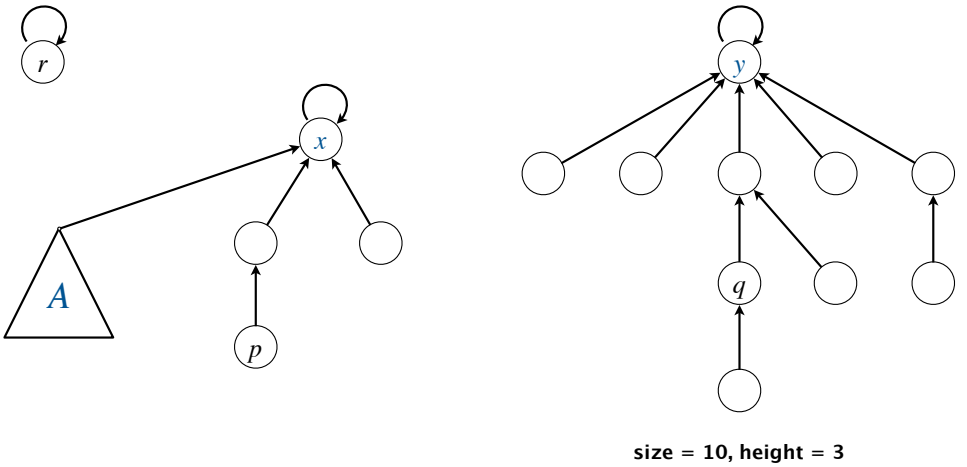
   ☐ *bytes*

   (b) What are the *minimum* and *maximum* possible numbers of `Node` *objects* in a `TwoThreeTree` with n key–value pairs? Express both answers using *tilde notation*.

   ☐ *min number of nodes* ☐ *max number of nodes*

3. **Data structures. (12 points)**

(a) The diagram below shows the *parent-link* representation of a *weighted quick union* (*link-by-size*) data structure (without path compression) immediately before a call to *union*. The triangle A represents a nonempty subtree.



Suppose this call to *union* would make x point to y.
For each part, fill in all checkboxes that apply.

i. Which of the following ordered pairs could have been the arguments to *union*?

☐ (p, q) ☐ (q, p) ☐ (x, y) ☐ (y, x) ☐ (x, x) ☐ (r, y)

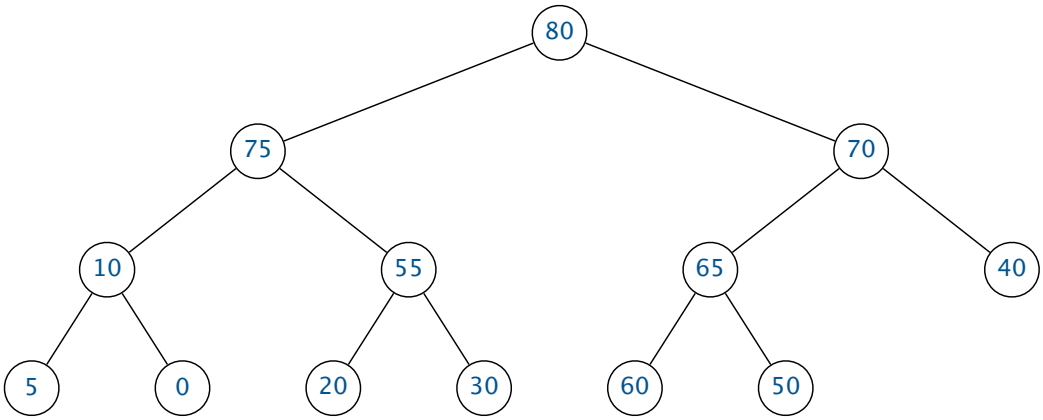
ii. Which of the following values could be the *size* (number of nodes) of subtree A?

☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7

iii. Which of the following values could be the *height* of subtree A?

☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6

(b) Consider the following binary-tree representation of a *max-oriented binary heap*:

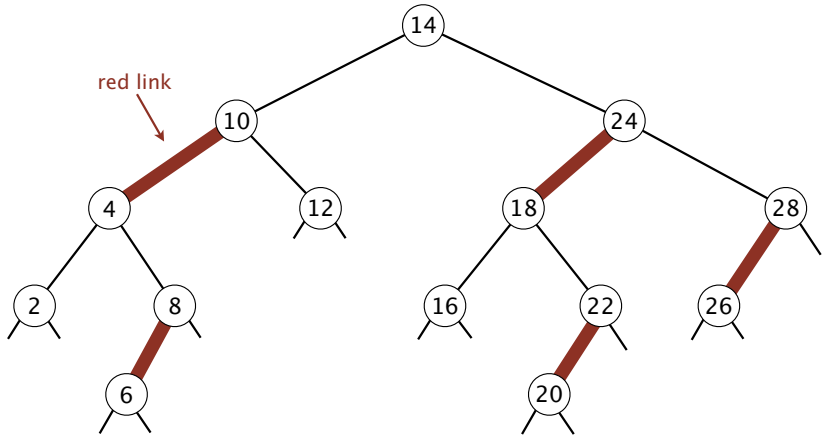


For which values of x does *inserting* x require exactly 3 key compares and exactly 2 exchanges during `swim()`?

Fill in all checkboxes that apply.

☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
31	37	41	51	61	67	71	77	81	87

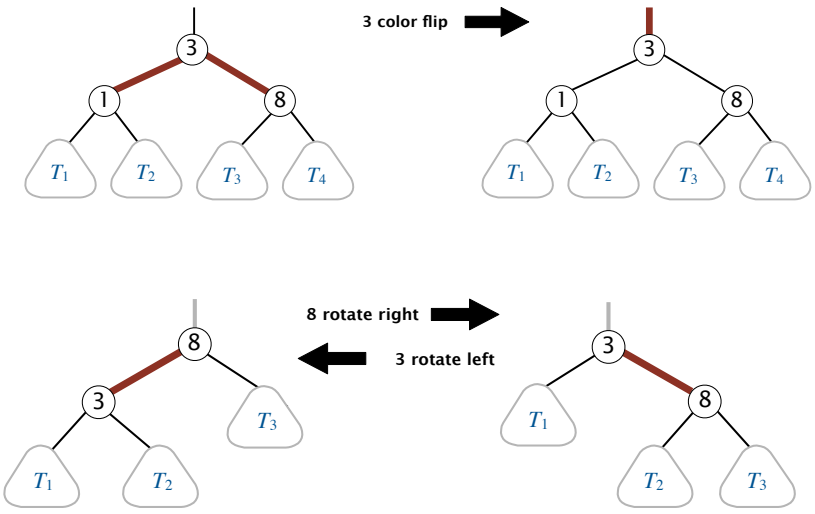
(c) *Insert* the key 9 into the following *left-leaning red–black BST* using the insertion algorithm from lecture.



Give the sequence of four elementary operations triggered by this insertion.

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

Examples of elementary operations (for reference):



4. **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 B–F exactly once.

63	31	31	12	39	68	12
59	59	39	26	59	65	26
71	63	56	31	12	56	31
65	65	59	33	33	45	33
87	70	63	39	45	63	39
91	71	65	45	26	47	45
70	87	68	47	47	33	47
31	91	70	56	31	39	56
81	81	71	59	56	26	59
39	39	81	63	63	59	63
68	68	87	65	68	31	65
56	56	91	68	81	12	68
47	47	12	70	70	70	70
72	72	26	72	72	71	71
75	75	33	75	75	72	72
74	74	45	74	74	74	74
86	86	47	86	86	75	75
26	26	72	81	91	81	81
45	45	74	91	87	83	83
33	33	75	71	65	86	86
88	88	83	88	88	87	87
12	12	86	87	71	88	88
98	98	88	98	98	91	91
83	83	98	83	83	98	98
A						G

A. Original array

B. Selection sort

C. Insertion sort

D. Mergesort (*top-down*)

E. Quicksort (*standard, no shuffle*)

F. Heapsort

G. Sorted array

5. **Analysis of algorithms and sorting. (6 points)**

Consider an array of length $4n$ that consists of n copies of the block `2 2 2 6`.
For example, here is the array when $n = 4$:

```
2  2  2  6  2  2  2  6  2  2  2  6  2  2  2  6
```

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

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

(a) Selection sort.

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

(b) Insertion sort.

◯ $\sim \frac{1}{2}n^2$ ◯ $\sim \frac{3}{2}n^2$ ◯ $\sim 3n^2$ ◯ $\sim 4n^2$ ◯ $\sim 8n^2$

(c) 3-way quicksort (no shuffle).

◯ $\sim 4n$ ◯ $\sim 5n$ ◯ $\sim 4n \log_2 n$ ◯ $\sim 8n \ln n$ ◯ $\sim 8n^2$

6. **Percolation. (8 points)**

Generalize `Percolation.java` to handle an n-by-n-by-n cubic grid, where $n \geq 3$. Sites have coordinates (i, j, k), with $0 \leq i, j, k < n$. Each interior site is adjacent to six neighboring sites. The system percolates when there is a path of open sites between the top and bottom faces.

Model the clusters of open sites using one *union–find data structure* with n^3 elements for the grid sites and two additional elements for the virtual top and bottom faces.

An experiment starts with all sites blocked and repeatedly samples a site uniformly at random, opening it if blocked, until percolation. For running-time analysis, assume that approximately $0.312n^3$ sites are opened, corresponding to the 3D percolation threshold.

(a) Complete the Java instance method below to map each site (i, j, k) to a unique index between 0 and $n^3 - 1$. Assume `n` is an instance variable storing the grid dimension.

```
private int siteToIndex(int i, int j, int k) {

  return                                        ;

}
```

(b) What is the maximum number of `union()` calls needed to open a site?

◯ 1 ◯ 2 ◯ 4 ◯ 6 ◯ n

(c) After opening a site, how many `find()` calls are needed to check for percolation?

◯ 2 ◯ 3 ◯ $2n$ ◯ $2n^2$ ◯ n^4

(d) Using `QuickFindUF`, what is the expected running time of one experiment?

◯ $\Theta(n^3)$ ◯ $\Theta(n^4)$ ◯ $\Theta(n^5)$ ◯ $\Theta(n^6)$ ◯ $\Theta(n^8)$

7. **Linked lists. (5 points)**

Consider the following linked-list representations for implementing selected operations from the `Deque` API:

A. Singly linked list with 1 pointer.

B. Singly linked list with 2 pointers.

C. Doubly linked list with 1 pointer.

D. Doubly linked list with 2 pointers.

Each representation uses a null-terminated linked list, plus the specified number of node pointers maintained by the collection data type (e.g., to the first or last node of the list). You may choose how the items are arranged in the list. No other data structures or instance variables are permitted.

For each set of operations below, choose the *first* representation among A–D that supports every operation in *constant worst-case time*; choose E if none does.

Write one of A–E in each box. You may use each letter any number of times.

☐ `addFirst()`, `removeFirst()`

☐ `addFirst()`, `addLast()`, `removeFirst()`

☐ `addFirst()`, `addLast()`, `removeLast()`

☐ `addFirst()`, `removeFirst()`, `removeLast()`

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

8. **Binary search. (6 points)**

Given two integer arrays of equal length, with `a[]` strictly increasing and `b[]` strictly decreasing, design an algorithm to find an index i such that `a[i] == b[i]`, or return `-1` if none exists. For the arrays below, the method should return index `4`.

a[]	1	2	3	5	8	13	21
	0	1	2	3	4	5	6
b[]	128	64	32	16	8	4	2

To do so, complete the following partial Java implementation:

```
// precondition: a[] sorted, b[] reverse sorted, same length
public static int findCrossingIndex(int[] a, int[] b) {
    int lo = 0;
    int hi = a.length - 1;
    while ( 1 ) {
        int mid = lo + (hi - lo) / 2;
        if      ( 2 ) lo = 3 ;
        else if ( 4 ) hi = 5 ;
        else return 6 ;
    }
    return -1 ;   // no such index
}
```

A. `hi <= lo`
B. `hi >= lo`
C. `hi > lo`
D. `hi > lo + 1`
E. `mid - 1`
F. `mid`
G. `mid + 1`
H. `a[mid] < b[mid]`
I. `a[mid] > b[mid]`
J. `a[mid] == b[mid]`
K. `a[mid] >= b[mid]`
L. `a[mid] <= b[mid]`

For each numbered oval above, write the uppercase letter of the corresponding expression on the right in the space provided. You may use each letter any number of times.

1	2	3	4	5	6

9. **Sum of the k smallest. (11 points)**

Design a data type that maintains a set of distinct integers and returns the sum of the k smallest integers, where $k > 0$. Let n be the number of integers in the set.

operation	description	worst-case running time
`KSmallestSum(int k)`	*create an empty set*	$\Theta(1)$
`void add(int x)`	*add integer x (do nothing if already present)*	$O(\log n)$
`void remove(int x)`	*remove integer x (do nothing if absent)*	$O(\log n)$
`int sum()`	*sum of k smallest integers in set (or all integers if n < k)*	$\Theta(1)$

Partial credit. For *half credit*, your implementation may instead satisfy either of the following worst-case running-time requirements:

- `add()` and `remove()` take $O(n)$ time, and `sum()` takes $\Theta(1)$ time.
- `add()` and `remove()` take $O(\log n)$ time, and `sum()` takes $O(k \log n)$ time.

Example.

```
KSmallestSum kss = new KSmallestSum(3);  // { }
kss.add(2);                              // { 2 }
kss.add(5);                              // { 2, 5 }
kss.sum();                               // returns 7
kss.add(7);                              // { 2, 5, 7 }
kss.add(9);                              // { 2, 5, 7, 9 }
kss.sum();                               // returns 14
kss.remove(2);                           // { 5, 7, 9 }
kss.sum();                               // returns 21
```

Note: the comments at right list the integers in ascending order; the API does not require any particular internal order.

Your answer will be graded for correctness, efficiency, and clarity (but not Java syntax). If your solution relies upon an algorithm or data structure from the course, do not reinvent it; simply describe how you are applying it.

(a) *Using Java code*, declare the instance variables and any supporting nested classes for `KSmallestSum`. You may use the full APIs of any data types covered in this course.

```
public class KSmallestSum {









}
```

(b) Concisely describe your algorithm for `sum()`, using English, pseudocode, or Java code. Ignore integer overflow.

```
public int sum() {









}
```

(c) Concisely describe your algorithm for `add()`, using English, pseudocode, or Java code.

```
public void add(int x) {









}
```

(d) Concisely describe your algorithm for `remove()`, using English, pseudocode, or Java code.

```
public void remove(int x) {









}
```

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