

**Midterm**

This exam has 10 questions worth a total of 60 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	P04	P05	P06	P07	P08	P09
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

*"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. Resizable arrays. (8 points)

Consider the following partial implementation of a stack of integers that uses a resizable array. Assume that the implementation *triples* the length of the array during a `push()` operation if the array is full, *but that it never shrinks the array during a `pop()` operation.*

```
public class StackOfInts {
    private int[] a;    // underlying array
    private int n;     // number of integers in the stack
    ...
}
```

- (a) In the *worst case*, what is the memory of a `StackOfInts` object *after* calling `push()`  $n$  consecutive times on an initially empty stack? Use our 64-bit memory cost model.

*Write your answer in the box below, using tilde notation to simplify your answer.*

~	bytes
---	-------

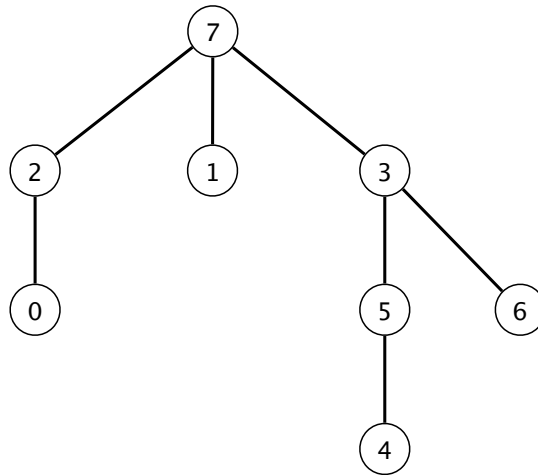
- (b) Identify each statement as *true* or *false* by filling in the appropriate bubble.

*true*      *false*

- |                       |                       |  |
|-----------------------|-----------------------|--|
| <input type="radio"/> | <input type="radio"/> | The <code>push()</code> operation takes $\Theta(1)$ time in the worst case.  |
| <input type="radio"/> | <input type="radio"/> | The <code>pop()</code> operation takes $\Theta(1)$ time in the worst case.   |
| <input type="radio"/> | <input type="radio"/> | Starting from an empty stack, any intermixed sequence of $m$ <code>push()</code> and <code>pop()</code> operations takes $\Theta(m)$ time in the worst case. |

3. Data structures. (6 points)

- (a) Consider the following *parent-link* representation of a *weighted quick union (link-by-size)* data structure:



Suppose that the last operation was a call to `union()` on two elements in different sets. Which pair of elements could it have been?

*Fill in all checkboxes that apply.*

0-1

0-2

0-3

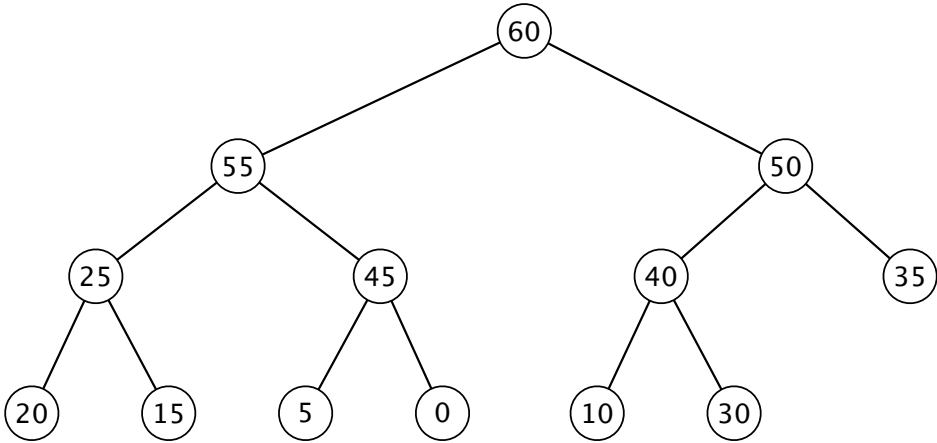
0-4

0-5

0-6

0-7

(b) Consider the following maximum-oriented *binary heap*:



Suppose that the last operation in the binary heap was a call to `insert()`. Which key could have been the last one inserted?

*Fill in all checkboxes that apply.*

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
0	5	10	15	20	25	30
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
35	40	45	50	55	60	

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 exactly once.*

78	23	10	10	69	40	10
69	45	12	12	66	69	12
23	58	13	13	67	23	13
58	67	19	19	58	58	19
45	69	22	20	52	45	20
67	78	23	22	60	67	22
83	10	45	23	54	26	23
49	19	49	26	49	49	26
22	22	54	40	45	22	40
10	49	58	45	10	10	45
80	80	60	49	40	66	49
19	83	67	52	26	19	52
60	60	69	60	19	60	54
12	12	78	69	12	12	58
54	54	80	54	23	54	60
13	13	83	83	13	13	66
20	20	20	78	20	20	67
81	81	81	81	22	52	69
75	75	75	75	75	75	75
40	40	40	67	78	78	78
52	52	52	58	80	81	80
66	66	66	66	81	80	81
89	89	89	89	83	89	83
26	26	26	80	89	83	89

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 that contains  $n$  copies of A, B, C, and D, in that order.  
 For example, here is the array when  $n = 4$ :

A B C D A B C D A B C D A B C D

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

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

(a) Selection sort

- |                       |                       |                       |                       |                       |
|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |
| $\sim \frac{1}{2}n^2$ | $\sim n^2$            | $\sim 2n^2$           | $\sim 4n^2$           | $\sim 8n^2$           |

(b) Mergesort

- |                       |                       |                       |                              |                       |
|-----------------------|-----------------------|-----------------------|------------------------------|-----------------------|
| <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/>        | <input type="radio"/> |
| $\sim n \log_2 n$     | $\sim 2n \log_2 n$    | $\sim 3n \log_2 n$    | $\sim \frac{7}{2}n \log_2 n$ | $\sim 4n \log_2 n$    |

(c) 3-way quicksort

- |                       |                       |                       |                       |                       |
|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |
| $\Theta(n)$           | $\Theta(n \log n)$    | $\Theta(n^2)$         | $\Theta(n^4)$         | $\Theta(2^n)$         |

## 6. Advanced Java. (6 points)

Suppose that you remove the clause `implements Iterable<Item>` from the class declaration of our textbook implementation of `Stack`, but leave everything else exactly the same.

For each client code fragment at left, write the letter of the best matching description at right. Assume all relevant import statements are provided.

- |                          |   |  |
|--------------------------|---|--|
| <input type="checkbox"/> | <pre>Stack&lt;String&gt; stack = new Stack&lt;String&gt;(); stack.push("A"); stack.push(226); StdOut.println(stack.pop());</pre>                          | <p>A. <i>prints A</i></p> <p>B. <i>prints B</i></p> <p>C. <i>prints 226</i></p>        |
| <input type="checkbox"/> | <pre>Stack&lt;String&gt; stack = new Stack&lt;String&gt;(); stack.push("A"); stack.push("B"); Object x = stack.pop(); StdOut.println(x.toString());</pre> | <p>D. <i>prints A B</i></p> <p>E. <i>prints B A</i></p> <p>F. <i>infinite loop</i></p> |
| <input type="checkbox"/> | <pre>Object stack = new Stack&lt;String&gt;(); stack.push("A"); stack.push("B"); StdOut.println(stack.pop());</pre>                                       | <p>G. <i>compile-time error</i></p> <p>H. <i>run-time error</i></p>                    |
| <input type="checkbox"/> | <pre>Stack&lt;String&gt; stack = new Stack&lt;String&gt;(); stack.push("A"); stack.push("B"); for (String s : stack) {     StdOut.println(s); }</pre>     |  |



**7. Properties of BSTs. (6 points)**

Identify each statement as *true* or *false* by filling in the appropriate bubble.

*true*      *false*

- Applying a *left rotation* to a node in a BST yields another BST.
- For any set of  $n$  distinct keys, it is possible to construct a 2–3 tree on those  $n$  keys using a total of at most  $3n$  key compares.
- In a *left-leaning red–black BST* containing  $n$  keys, the maximum number of black links on a path from the root node to a null link is  $\sim \log_2 n$ .
- Suppose that you insert  $n$  distinct keys into an initially empty left-leaning red–black BST. Then, regardless of the order in which you insert the keys, the total number of *color flips* is at most  $n$ .

## 8. Rank in a BST. (6 points)

Complete the following partial implementation of the `rank()` method in a *binary search tree*:

```
// return the number of keys in BST that are strictly less than key
public int rank(Key key) {
    return rank(key, root);
}

// return the number of keys in subtree rooted at x
// that are strictly less than key
public int rank(Key key, Node x) {
    if (x == null) return 0;
    int cmp = key.compareTo(x.key);
    int count = 0;
    if (x.left != null) count = 1 ;
    if (cmp < 0) return 2 + rank(key, 3 );
    if (cmp > 0) return 4 + rank(key, 5 );
    if (cmp == 0) return 6 ;
}
```

- A. 0
- B. 1
- C. count
- D. 1 + count
- E. x.left
- F. x.right
- G. x.key
- H. x.value
- I. x.size
- J. x.left.size
- K. x.right.size

Assume that the keys in the BST are distinct and that each BST node is represented using the following Node class:

```
private class Node {
    private Node left; // left subtree
    private Node right; // right subtree
    private Key key; // key
    private Value value; // value
    private int size; // number of keys in subtree rooted at this node
}
```

For each numbered oval above, write the letter of the corresponding expression on the right in the space provided. You may use each letter once, more than once, or not at all.

1

2

3

4

5

6

## 9. Algorithm design. (8 points)

- (a) Given a *sorted* array  $\mathbf{a}[]$  of  $n$  integers and an integer  $x$ , design an  $O(\log n)$  time algorithm that determines whether  $x$  appears in the array strictly more than  $\frac{n}{4}$  times.

**Example.** If the array  $\mathbf{a}[]$  consists of the following 16 integers:

1 1 1 2 2 2 2 2 2 4 4 6 6 6 6 6

then the algorithm should output

- *yes* for  $x = 2$  because 2 appears  $6 > \frac{16}{4}$  times
- *yes* for  $x = 6$  because 6 appears  $5 > \frac{16}{4}$  times
- *no* for every other integer  $x$

*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) Given a *sorted* array  $\mathbf{a}[]$  of  $n$  integers, design an  $O(\log n)$  time algorithm that determines whether there exists *any* integer in the array that appears strictly more than  $\frac{n}{4}$  times.

**Example.** In the example array above, the algorithm should output *yes* because there are two integers (2 and 6) that appears more than  $n/4$  times.

*In the space provided, give a concise English description of your algorithm for solving the problem. You may use the algorithm for (a) as a subroutine.*

### 10. Data structure design. (8 points)

Design a collection data type that supports inserting integers and processing *nearest neighbor queries*. Given an integer  $x$ , its *nearest neighbor* is the integer in the collection that is *closest* to  $x$ , where the distance between two integers is the absolute value of their difference. (If there is a tie between two values, return either one.)

```
public class NearestNeighbor


---


    NearestNeighbor()    create an empty collection

    void insert(int x)   add x to the collection

    int nearest(int x)   return an integer in the collection that is closest to x
```

**Example.** Here is a small example sequence of operations.

```
NearestNeighbor nn = new NearestNeighbor(); // [ ]
nn.insert(126); // [ 126 ]
nn.insert(226); // [ 126 226 ]
nn.insert(217); // [ 126 226 217 ]
nn.insert(423); // [ 126 226 217 423 ]
nn.insert(487); // [ 126 226 217 423 487 ]
nn.nearest(324); // returns 423
nn.nearest(101); // returns 126
nn.insert(333); // [ 126 226 217 423 487 333 ]
nn.nearest(324); // returns 333
```

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

**Performance requirements.** For full credit,

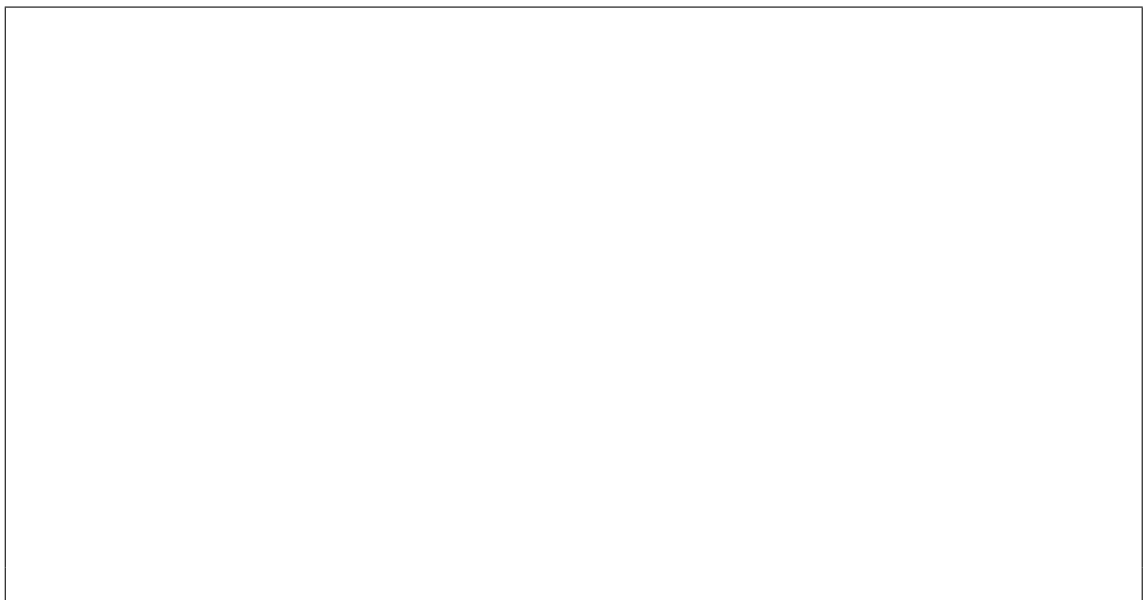
- The constructor must take  $\Theta(1)$  time.
- The `insert()` and `nearest()` methods must each take  $O(\log n)$  time in the worst case.

Here,  $n$  is the number of integers in the collection.

- (a) Using Java code, declare the instance variables (along with any supporting nested classes) that you would use to implement `NearestNeighbor`. 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) *Draw* a diagram of the underlying data structures (such as resizable arrays, linked lists, or binary trees) for a nearest neighbor data type containing the integers 126, 226, 217, 423, 487, and 333. For linked data structures, draw all links.



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

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

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