COS 226 Algorithms and Data Structures Spring 2024

Midterm

This exam has 8 questions worth a total of 55 points. (Question 9 is extra credit, you can skip it without losing any 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
○	○	○	○	○	○	○	○

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

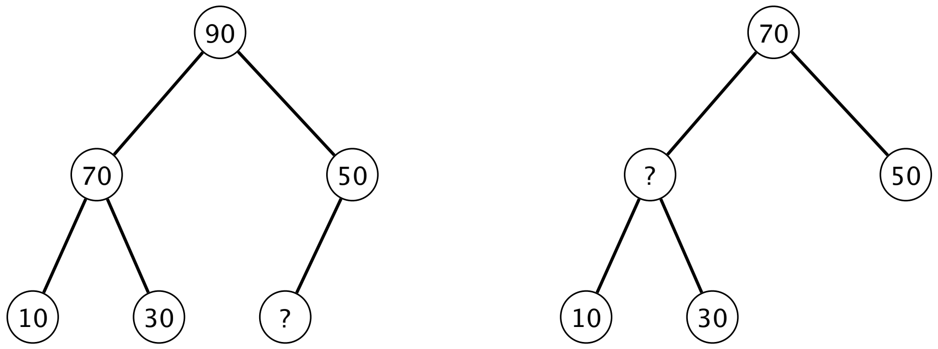
Signature

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. **Heaps and trees. (9 points)**

   (a) A `delMax()` operation on the *maximum-oriented binary heap* on the left produces the binary heap on the right.



Which of the keys below could be the one labeled with a question mark?

Fill in all checkboxes that apply.

☐	☐	☐	☐	☐
25	35	45	55	65

(b) The following is a partial *level-order* traversal of a *binary search tree* (BST):

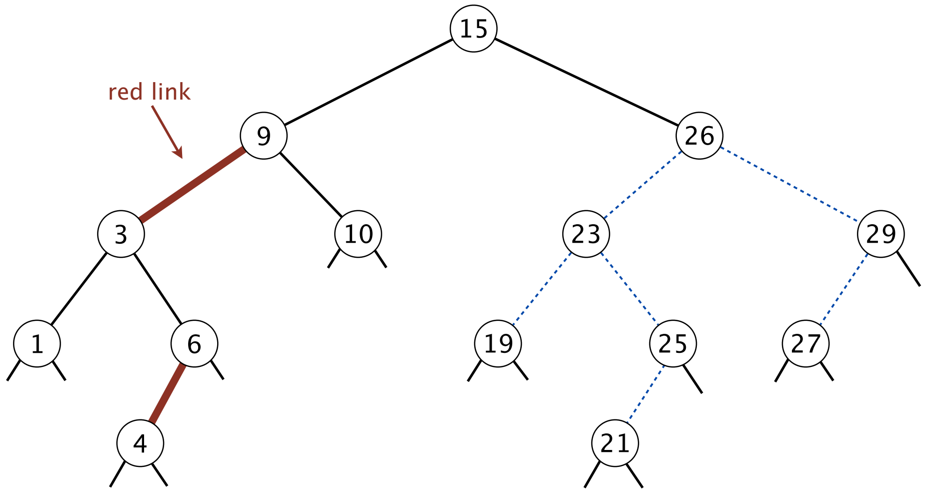
```
80   40   110   ?   100   120   60
```

Which of the keys below could be the one labeled with a question mark?

Fill in all checkboxes that apply.

☐	☐	☐	☐	☐
30	50	70	90	115

(c) Consider the following *left-leaning red-black BST* (some of the edge colors are suppressed):



Which keys below must be red (a key is red if the link between it and its parent is red)?

Fill in all checkboxes that apply.

☐ 19 ☐ 21 ☐ 23 ☐ 25 ☐ 27 ☐ 29

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.

81	12	34	12	60	12	12
12	16	12	16	56	16	16
80	21	80	21	43	21	21
21	33	21	30	47	33	30
75	37	75	33	33	37	33
56	43	56	34	37	41	34
43	56	43	37	41	43	37
65	65	65	41	34	56	41
37	75	37	43	21	65	43
33	80	33	47	16	70	47
16	81	16	56	12	71	56
95	95	69	60	30	75	60
85	30	30	85	65	80	65
71	34	71	71	69	81	69
41	41	41	65	70	85	70
70	47	70	70	71	95	71
47	60	47	75	75	47	75
60	69	60	95	80	60	80
34	70	81	81	81	34	81
30	71	85	80	85	30	85
94	85	94	94	86	94	86
86	86	86	86	93	86	93
93	93	93	93	94	93	94
69	94	95	69	95	69	95
A						G

A. Original array

B. Selection sort

C. Insertion sort

D. Mergesort (*top-down*)

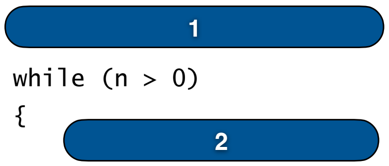
E. Quicksort (*standard, no shuffle*)

F. Heapsort

G. Sorted array

4. **Stacks and queues. (6 points)**

   The following partial code is intended to print the binary representation of the positive integer n to the standard output. For instance, if $n = 6$ then 110 is printed and if $n = 50$ then 110010 is printed.



```
while (n > 0)
{
    n = n / 2;
}

for (int d : collection)
    StdOut.print(d);
```

A. `Stack<Integer> collection = new Stack<Integer>();`

B. `Queue<Integer> collection = new Queue<Integer>();`

C. `collection.push(n % 10);`

D. `collection.enqueue(n % 10);`

E. `collection.push(n % 2);`

F. `collection.enqueue(n % 2);`

(a) Complete the two missing lines of code in the above partial implementation. For this question, the declaration and implementation of `Stack` and `Queue` are as in the textbook and lectures.

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

(b) How many elements are in `collection` at the end of the execution of the code?

○ $\Theta(1)$ ○ $\Theta(\log\log n)$ ○ $\Theta(\log n)$ ○ $\Theta(n)$ ○ $\Theta(n\log n)$ ○ $\Theta(n^2)$

(c) Suppose that `collection` is implemented as a *resizable array.* How many times would the array expand its size while running the code?

○ $\Theta(1)$ ○ $\Theta(\log\log n)$ ○ $\Theta(\log n)$ ○ $\Theta(n)$ ○ $\Theta(n\log n)$ ○ $\Theta(n^2)$

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

Consider an array of $2n$ elements in the form $1, 2n-1, 2, 2n-2, 3, 2n-3, 4, 2n-4, \ldots, n, n$. For example, here is the array when $n = 8$:

```
1  15  2  14  3  13  4  12  5  11  6  10  7  9  8  8
```

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 $2n$, not n.

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

(a) *Selection sort*

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

(b) *Insertion sort*

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

(c) *Mergesort*

◯	◯	◯	◯	◯
$\sim \frac{1}{2} n \log_2 n$	$\sim \frac{3}{4} n \log_2 n$	$\sim n \log_2 n$	$\sim \frac{3}{2} n \log_2 n$	$\sim 2n \log_2 n$

(d) *3-way Quicksort* (using 3-way partition, *no shuffle*)

◯	◯	◯	◯	◯
$\Theta(\log n)$	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^4)$

6. **Asymptotic. (8 points)**

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

true *false*

○ ○ The loop bellow prints `hello` $\sim 2n^2$ times.

```
for (i = n*n; i > 1; i = i/2)
    for (j = 0; j < i; j++)
        System.out.println("hello");
```

○ ○ The reason we implement binary heaps as arrays instead of using explicit nodes and links is that any implementation using explicit nodes and links will not allow for `insert()` and `delMax()` to both run in $O(\log n)$ time.

○ ○ It is possible to implement a *binary search tree (BST)* data structure where *insert* makes $O(\sqrt{\log n})$ compares in the worst case (*search* and *delete* may make any number of compares).

○ ○ Consider an $n \times n$ two-dimensional array of integers in which the integers in each row are in ascending order and the first integer of each row is greater than the last integer of the previous row. Given an integer x, it is possible to check if x appears in the array by making $O(\log n)$ compares (recall that $\log(n^2) = 2\log(n)$).

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

   Given two integer arrays, `a[]` and `b[]`, the *symmetric difference* between `a[]` and `b[]` is the set of elements that appear in exactly one of the arrays. Design an algorithm that receives two *sorted arrays*, each consisting of *n distinct elements*, and outputs the size of their symmetric difference.

   Full credit: The running time of the algorithm must be $O(n)$ in the worst case and the amount of extra memory must be $O(1)$ (the arrays `a[]` and `b[]` should not be modified).

   Partial credit (at least half credit): The running time of the algorithm must be $O(n \log n)$ in the worst case and the amount of extra memory must be $O(n)$.

   Specify the running time and extra memory usage of your solution.

   Example ($n = 6$). If the array `a[]` is:

   ```
   3   6   10   14   17   18
   ```

   and the array `b[]` is:

   ```
   2   3   7   12   14   15
   ```

   then the algorithm should output 8 because 6, 10, 17 and 18 are only in `a[]` and 2, 7, 12 and 15 are only in `b[]`.

   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.

   The running time of your solution is $\Theta(\qquad\qquad)$

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

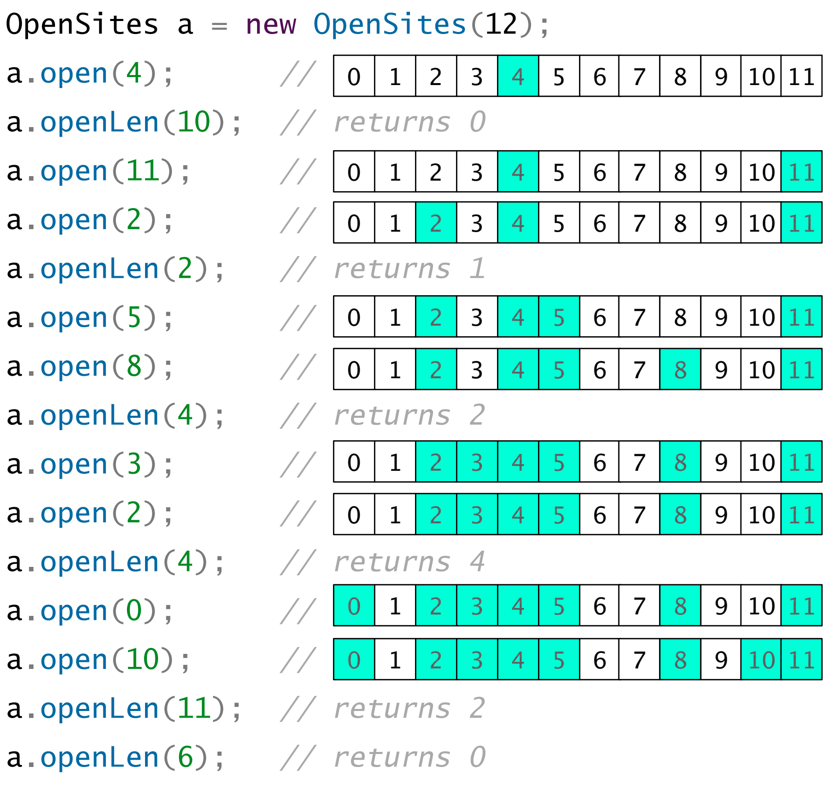
Design a collection data type that represents the state of n sites, each labeled with a number from 0 to $n-1$. When the collection is constructed, all sites are *blocked.* The data structure supports two operations. The first one, `open()`, takes an integer i between 0 and $n-1$ and marks site i as *open* (if site i was open before, its state doesn't change). The second operation, `openLen()`, takes an integer i and returns the length of the maximum consecutive sequence of open sites that includes site i. For example, if site 1 is blocked, sites 2, 3, 4, 5 are open, and site 6 is blocked, then `openLen(3)` returns 4.

`public class OpenSites`

	`OpenSites(int n)`	*creates a collection of n blocked sites*
`void`	`open(int i)`	*marks site i as open*
`int`	`openLen(int i)`	*returns the length of the maximum consecutive sequence of open sites that contains i*

Example. Here is an example sequence of operations.

Sites highlighted in green are open.



```
OpenSites a = new OpenSites(12);
a.open(4);        // 0 1 2 3 4 5 6 7 8 9 10 11
a.openLen(10);    // returns 0
a.open(11);       // 0 1 2 3 4 5 6 7 8 9 10 11
a.open(2);        // 0 1 2 3 4 5 6 7 8 9 10 11
a.openLen(2);     // returns 1
a.open(5);        // 0 1 2 3 4 5 6 7 8 9 10 11
a.open(8);        // 0 1 2 3 4 5 6 7 8 9 10 11
a.openLen(4);     // returns 2
a.open(3);        // 0 1 2 3 4 5 6 7 8 9 10 11
a.open(2);        // 0 1 2 3 4 5 6 7 8 9 10 11
a.openLen(4);     // returns 4
a.open(0);        // 0 1 2 3 4 5 6 7 8 9 10 11
a.open(10);       // 0 1 2 3 4 5 6 7 8 9 10 11
a.openLen(11);    // returns 2
a.openLen(6);     // returns 0
```

Performance requirements. Denote by n the number of sites in the collection (specified at construction).

Full credit:

- The constructor must take $O(n)$ time in the worst case.
- The `open()` and `openLen()` 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 half credit): Same performance guarantees as in the full credit option, except that `open()` can take time $O(n)$.

(a) Specify the instance variables (along with any supporting nested classes) that you would use to implement `OpenSites`. 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 the *constructor*. You may use code or pseudocode to improve clarity.

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

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

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

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

9. **EXTRA CREDIT. (4 extra points)**

THIS QUESTION IS EXTRA CREDIT. YOU CAN SKIP IT WITHOUT LOSING ANY POINTS.

Design an `OpenSites` data structure as in the previous question, with the additional functionality of blocking a site. That is, in addition to the `open()` and `openLen()` methods, the collection should also support the method `void block(int i)` that marks site i as blocked (if site i was blocked before, its state doesn't change). In addition to the performance requirements of the previous question, the `block()` method should work in time $O(\log n)$.

A solution to this question implies a solution to the previous question. You have the option to write separate solutions for both questions (this is advisable), or, if you are sure about your answer to this more challenging problem, you can ask us to only look at your answer to this problem (this is discouraged).

Please read:

- ◯ Both my solution to the previous question and my solution to this question.
- ◯ Only my solution to this question for the credit of both questions.

(a) Specify the instance variables (along with any supporting nested classes) that you would use to implement `OpenSites`. 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 the *constructor*. You may use code or pseudocode to improve clarity.

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

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

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

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