COS 226	Algorithms and Data Structures	Fall 2019

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 the precept in which you are officially registered. Finally, write and sign the Honor Code pledge. You may fill in this information now.

Name:

NetID:

Course: COS 226 ●

Exam room: Friend 101 ○ Friend 003 ○ McDonnell A02 ○ Other ○

Precept: P01 ○ P02 ○ P04 ○ P05 ○ P07 ○ P08 ○ P09 ○ P10 ○

"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 the precept in which you are officially registered; write and sign the Honor Code pledge.

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

   Suppose that you implement a priority queue (of integer keys) using a 4-way heap (d = 4) with the following data type:

```
public class MultiWayHeap {
    private final int d;            // branching factor
    private Node root;              // root of d-way heap
    private int n;                  // number of keys in d-way heap

    private class Node {
        private final int key;      // key
        private Node parent;        // link to parent
        private Node[] children;    // links to d children

        private Node(int key) {
            this.key = key;
            children = new Node[d];
        }
    }
    ...
}
```

   Use the 64-bit memory cost model from lecture and the textbook to answer the following questions.

   (a) How much memory does each `Node` object in a *4-way* heap use? Count all memory allocated when a `Node` object is constructed.

   bytes

   (b) How much memory does a `MultiWayHeap` object for a *4-way* heap use as a function of the number n of integer keys in the data structure? Count all referenced memory. Use tilde notation to simplify your answer.

   $\sim$ bytes

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.

37	11	18	11	11	60	11
79	18	30	18	18	58	18
11	22	11	28	22	56	22
50	28	22	37	28	57	28
39	30	28	39	30	30	30
66	37	37	50	37	37	37
78	39	78	56	39	39	39
18	50	66	66	50	50	50
80	56	80	78	56	11	56
28	57	39	79	64	22	57
56	58	56	80	66	28	58
98	60	98	98	78	18	60
92	92	92	22	79	61	61
22	66	50	30	80	63	63
30	78	79	57	92	64	64
64	64	64	58	98	66	66
86	86	86	60	86	78	78
57	79	57	61	57	79	79
83	83	83	63	83	80	80
63	63	63	64	63	81	81
60	98	60	81	60	83	83
61	61	61	83	61	86	86
58	80	58	86	58	92	92
81	81	81	92	81	98	98
0						6

(0) Original array

(1) Selection sort

(2) Insertion sort

(3) Mergesort (*top-down*)

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

(5) Heapsort

(6) Sorted array

3. **Quicksort and analysis. (5 points)**

Consider the following drop-in replacement for Hoare's 2-way partitioning algorithm.

```
// rearrange a[lo..hi] so that a[lo..j-1] <= a[j] <= a[j+1..hi]
private int partition(Comparable[] a, int lo, int hi) {
    int j = lo;
    for (int i = lo + 1; i <= hi; i++)
        if (less(a[i], a[lo]))    // strictly less
            exch(a, i, ++j);
    exch(a, lo, j);
    return j;
}
```

(a) What is the *maximum* number of calls to `less()` during one call to `partition()`? Write your answer as a function of the length n of the subarray to be partitioned and use tilde notation to simplify.

~ calls to `less()`

(b) Repeat the previous question, but for the *maximum* number of calls to `exch()`.

~ calls to `exch()`

(c) Which of the following are properties of this `partition()` method? Mark all that apply.

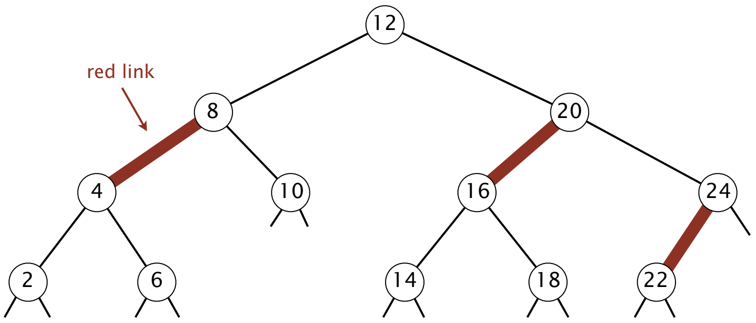
■ *correct* □ *stable* □ *not stable* □ *in-place* □ *not in-place*

(d) Suppose that the `partition()` method defined above is used in a standard recursive version of quicksort. How many total calls to `less()` would this version of *quicksort* make to sort an array of n *equal keys*? Use tilde notation to simplify your answer.

~ calls to `less()`

4. **Red–black BSTs. (4 points)**

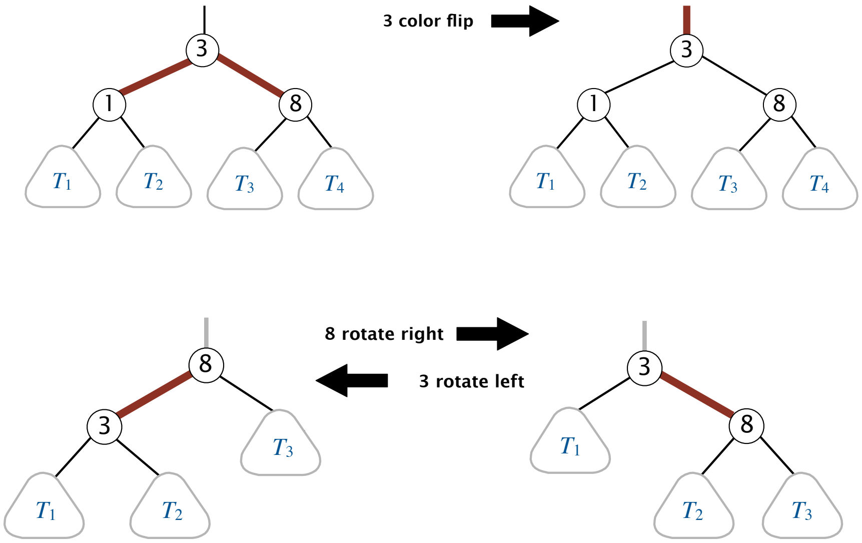
Suppose that you *insert* the key 21 into the following left-leaning red–black BST:



Give the sequence of 4 elementary operations (*color flips* and *rotations*) that result.

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

Examples of color flips and rotations (for reference):



5. **Collections. (5 points)**

To perform each task at left, write the letter of the *best-matching* collection at right. Use each letter *exactly once.*

□ Implement the *A* search algorithm.*

□ Remove duplicates from a mailing list.

□ Implement function calls in the Java Virtual Machine.

□ Estimate the percolation threshold in an n-by-n grid.

□ Order students randomly for draw times in a housing lottery.

□ Find all intersections among a set of n vertical and horizontal line segments using the sweep-line algorithm.

□ Create a buffer of samples to send to a sound card in a streaming audio application. The data must leave the buffer in the same order that it entered.

□ Implement the *A-Steal job scheduling algorithm*, in which Each processor maintains a list of threads to be executed. To execute the next thread, the processor gets the thread at the front of its list. If the list is empty, it *steals* a thread from the back of another processor.

A. Stack

B. Queue

C. Randomized queue

D. Deque

E. Priority queue
(binary heap)

F. Symbol table
(hash table)

G. Ordered symbol table
(red–black BST)

H. Union–find

6. **Why did we do that? (8 points)**

For each pair of algorithms or data structures, identify a critical reason why we prefer the first to the second. Write the letter of the best-matching answer. You may use each letter once, more than once, or not at all.

□ Implement a *stack* with a *singly linked list* instead of a *resizing array.*

□ Implement a *binary heap* with a *resizing array* instead of a *binary tree* (with explicit parent and children pointers).

□ In *weighted quick union*, merge the *smaller tree into the larger tree* instead of the *larger tree into the smaller tree.*

□ Use *quicksort* instead of *mergesort* to sort an array of primitive types.

□ Use *mergesort* instead of *heapsort* to sort an array of objects.

□ During a *delete-the-maximum* operation in a *binary heap*, exchange a key with the *larger* of its two children instead of the *smaller* of its two children.

□ *Rehash all of the keys into new chains* in a *separate-chaining hash table* when resizing the underlying array instead of keeping the old chains.

□ When performing a *2d range search* in a *2d-tree*, explore the left subtree before the right subtree.

A. Guarantees correctness

B. Improves order of growth of worst-case running time

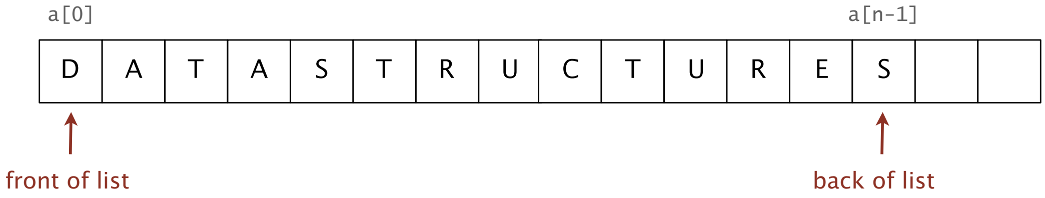
C. Uses less memory

D. Stability

E. Arbitrary decision

7. **Data structures. (5 points)**

Consider Java's `java.util.ArrayList` data type. It stores a list (sequence) of n elements in a *resizing array* (double when full, halve when one-quarter full), with the first element in the list (front) always at array index 0 and the last element in the list (back) at array index $n-1$.



For each part below, assume that there are n elements currently in the data structure. Based on the given internal representation, for each expression at left, identify the best-matching order-of-growth term at right. You may use each letter once, more than once, or not at all.

□ *Return* the element at index $n/2$ in the list.

□ Maximum *amount of memory* currently in use to represent the data structure.

□ *Worst-case* running time to perform n consecutive *add-to-back* operations. Each *add-to-back* operation adds an element to the back of the list.

□ *Worst-case* running time to perform n consecutive *remove-from-front* operations. Each *remove-from-front* operation removes the element at the front of the list.

□ Maximum number of times the array is resized when performing n consecutive *remove-from-back* operations.

A. *constant*

B. $\log n$

C. n

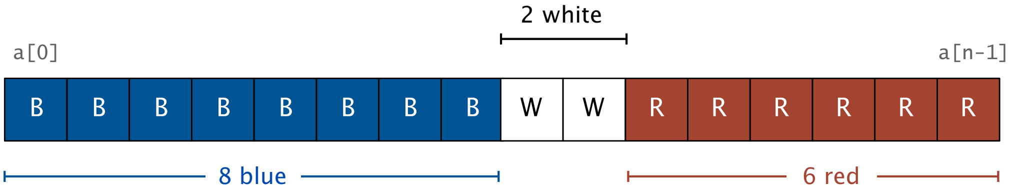
D. $n \log n$

E. n^2

F. n^3

8. **Red, white, and blue pebbles. (8 points)**

Suppose that you have an array `a[]` of length n containing n pebbles, each of which is colored *red*, *white*, or *blue*. Moreover, assume that there is at least one pebble of each color and that all pebbles of a given color are *contiguous* (but not necessarily in the order red, white, blue). The only operation you may perform on a pebble is to check the color of a pebble (e.g., to check whether two pebbles are the same color). Design an efficient algorithm to determine the number of pebbles of each color.



Give a crisp and concise English description of your algorithm in the space below. Your answer will be graded for correctness, efficiency, and clarity. For full credit, the order of growth of the running time must be $\log n$ *in the worst case.*

If your solution relies upon an algorithm or data structure from the course, do not reinvent it; simply describe how you are applying it.

9. **Data-type design. (10 points)**

Design a data type to implement a *double-ended priority queue*. The data type must support inserting a key, deleting a smallest key, and deleting a largest key. (If there are ties for the smallest or largest key, you may choose among them arbitrarily.)

To do so, create a `MinMaxPQ` data type that implements the following API:

`public class MinMaxPQ<Key extends Comparable<Key>>`

	`MinMaxPQ()`	*create an empty priority queue*
`void`	`insert(Key x)`	*add x to the priority queue*
`Key`	`min()`	*return a smallest key*
`Key`	`max()`	*return a largest key*
`Key`	`delMin()`	*return and remove a smallest key*
`Key`	`delMax()`	*return and remove a largest key*

Here are the performance requirements:

- The `insert()`, `delMin()`, and `delMax()` must take time proportional to $\log n$ (or better) in the worst case, where n is the number of keys in the priority queue. Significant partial credit for $\log n$ amortized.
- The `min()` and `max()` methods must take constant time in the worst case. Significant partial credit for $\log n$.

Here is an example:

```
MinMaxPQ<Integer> pq = new MinMaxPQ<Integer>();  //  [ ]
pq.insert(30);                                   //  [ 30 ]
pq.insert(40);                                   //  [ 30 40 ]
pq.max();                                        //  [ 30 40 ]       => 40
pq.delMin();                                     //  [ 40 ]          => 30
pq.insert(20);                                   //  [ 20 40 ]
pq.insert(10);                                   //  [ 10 20 40 ]
pq.delMax();                                     //  [ 10 20 ]       => 40
pq.insert(20);                                   //  [ 10 20 20 ]
pq.min();                                        //  [ 10 20 20 ]    => 10
pq.delMax();                                     //  [ 10 20 ]       => 20
pq.max();                                        //  [ 10 20 ]       => 20
```

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 (along with any supporting nested classes) that you would use to implement `MinMaxPQ`. You may use any of the data types that we have considered in this course (either `algs4.jar` or `java.util` versions). You may also make modifications to these data types; if you do so, describe the modifications.

```
public class MinMaxPQ<Key extends Comparable<Key>> {
```

(b) *Draw* the data structure(s) for a double-ended priority queue containing the following seven keys: 10, 20, 20, 20, 30, 40, 50. (Do not worry about the order in which the keys were inserted.) For linked data structures, draw all links.

(c) Give a concise English description of your algorithm for implementing `min()` and `max()`. If symmetric, describe only `min()`.

(d) Give a concise English description of your algorithm for implementing `insert(x)`.

(e) Give a concise English description of your algorithm for implementing `delMin()` and `delMax()`. If symmetric, describe only `delMin()`.

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.