COS 226 **Algorithms and Data Structures** **Spring 2025**

Midterm

This exam has 9 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 answer 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: ◯ Friend 101 ◯ Maeder 002 ◯ Andlinger 017 ◯ Other

Precept:

P01	P01A	P01B	P02	P02A	P02B	P03	P03A	P04
◯	◯	◯	◯	◯	◯	◯	◯	◯

"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 and NetID; fill in the bubbles for your exam room and the precept in which you are officially registered; write and sign the Honor Code pledge.

2. **Memory. (5 points)**

   Consider a *binary search tree* that is defined by the following Java implementation:

```
public class BST<Key extends Comparable<Key>, Value> {
    private Node root;   // the root of the red-black BST
    private int n;       // number of key-value pairs

    ...

    private class Node {
        private Key key;         // the key
        private Value value;     // the associated value
        private Node parent;     // link to parent
        private Node left;       // link to left subtree
        private Node right;      // link to right subtree
        private int count;       // number of nodes in subtree
    }
}
```

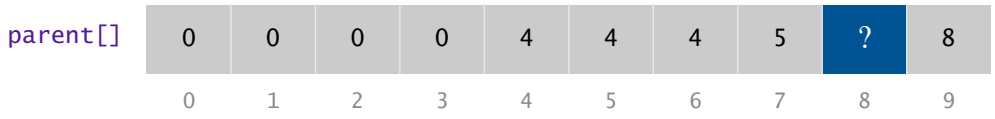
   How much memory (in bytes) does a `BST` with n key–value pairs use as a function of n? Count all memory (including object references) allocated by the `BST`, but do not count the memory for the keys and the values (which the client allocates). Use our 64-bit memory cost model.

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

   ~ bytes

3. **Data structures. (10 points)**

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

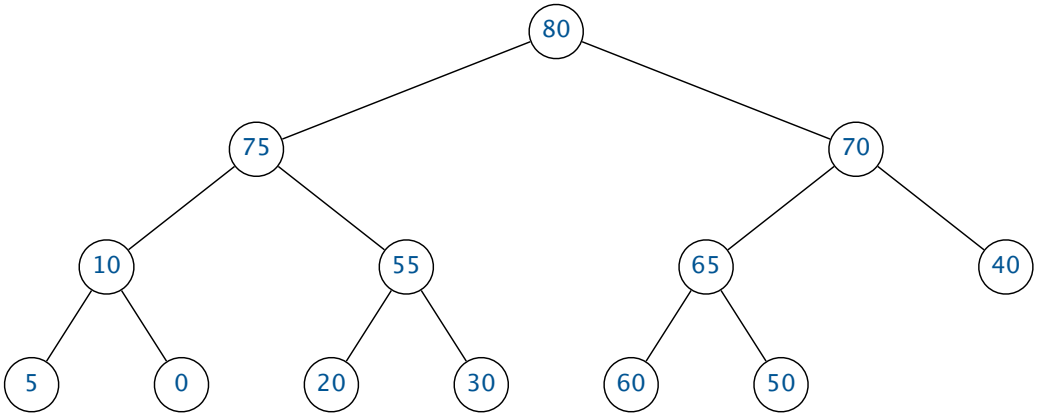


Which of the following values could be `parent[8]`?

Fill in all checkboxes that apply.

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

(b) Consider the following binary tree representation of a *binary heap*.



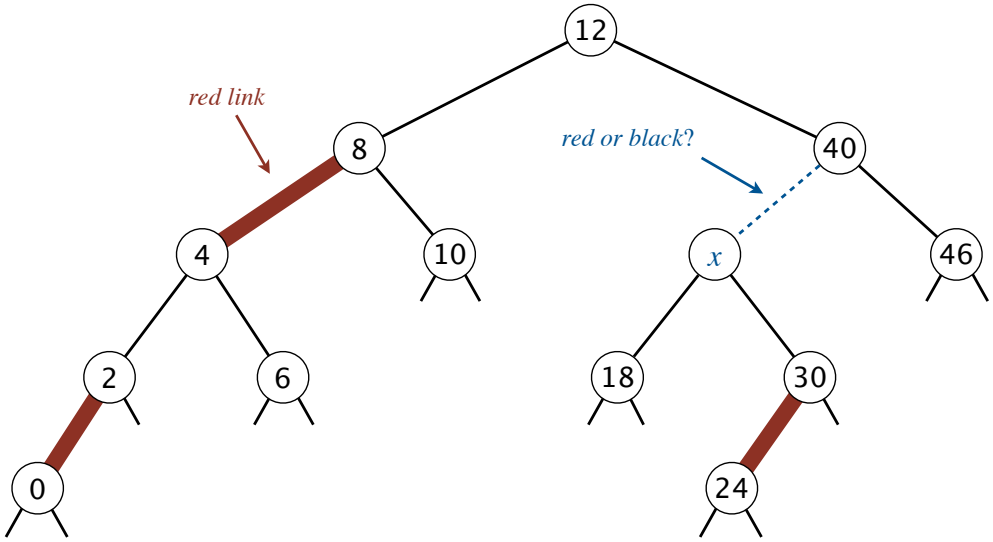
Suppose that the next operation is a DELETE-MAX. Which pairs of keys will be *compared* during the deletion?

Fill in all checkboxes that apply.

☐ 50–30 ☐ 50–55 ☐ 50–60 ☐ 50–65 ☐ 50–70 ☐ 50–75 ☐ 50–80

☐ 0–5 ☐ 20–30 ☐ 10–55 ☐ 70–75 ☐ 40–65

(c) Consider the following *left-leaning red-black BST*, with the key x in one node missing and the color y of one link missing:



Which of the following values could be the value of x? *Fill in all checkboxes that apply.*

☐ 20 ☐ 22 ☐ 24 ☐ 26 ☐ 28 ☐ 30 ☐ 32 ☐ 34 ☐ 36 ☐ 38

Which of the following values could be the color y? *Fill in all checkboxes that apply.*

☐ *red* ☐ *black* ☐ *orange*

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.

66	45	11	11	37	11	11
65	37	14	13	65	13	13
22	41	19	14	22	14	14
19	35	22	16	19	16	16
46	30	30	19	46	19	19
30	22	35	22	30	22	22
11	26	46	26	11	26	26
35	13	65	30	35	30	30
79	11	66	35	41	35	35
99	14	68	37	61	46	37
14	16	79	41	14	50	41
68	19	99	45	45	65	45
50	46	13	50	50	66	46
26	50	16	66	26	68	50
13	61	26	65	13	79	61
16	65	37	46	16	99	65
78	66	41	78	66	78	66
37	68	45	99	78	37	68
93	72	50	93	93	93	72
45	78	61	68	68	45	78
85	79	72	85	85	85	79
61	85	78	61	99	61	85
72	93	85	72	72	72	93
41	99	93	79	79	41	99
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 a *riffle shuffle* array of $2n$ elements of the form $1, n+1, 2, n+2, 3, n+3, \ldots, n, 2n$. For example, here is the array when $n = 8$:

```
1  9  2  10  3  11  4  12  5  13  6  14  7  15  8  16
```

How many *compares* does each sorting algorithm (standard algorithm, from the textbook) make as a function for n? Note that the length of the array is $2n$, not n.

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

(a) *Selection sort*

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

(b) *Insertion sort*

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

(c) *Mergesort*

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

6. **Algorithms. (10 points)**

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

true	*false*	
◯	◯	Given two *singly linked lists*, each containing n elements in *sorted order*, it is possible to create a *singly linked list* on the $2n$ elements in sorted order in $\Theta(n)$ time, using $\Theta(1)$ extra space.
◯	◯	Consider a *resizable array* of n `double` elements whose length *quadruples* ($4\times$) when the array is 100% full and whose lengths *halves* when the array is $\frac{1}{3}$ full. Then, the worst-case memory usage is $\sim 24n$ bytes.
◯	◯	It is possible to shuffle an array in $\Theta(n)$ time in the worst case by *enqueuing* the n items into a `RandomizedQueue` (from Assignment 2), and then *dequeuing* the n items.
◯	◯	Given a *red–black BST* on n distinct keys, it is possible to create a *binary heap* on the same n keys using $O(n)$ compares.
◯	◯	When inserting a key into a *left-leaning red–black BST*, every *right rotation* is followed immediately by a *color flip*.

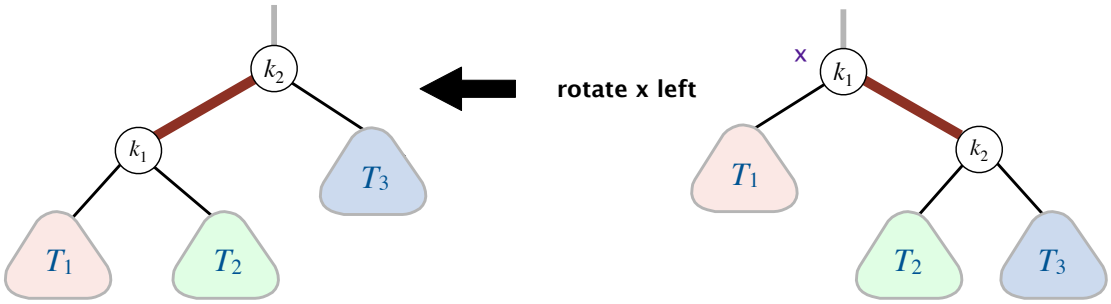
7. **Linked structures. (5 points)**

Complete the following partial implementation of the `rotateLeft()` method that *rotates a node left* in a red–black BST:

```
// rotate x left and return the root of the resulting subtree
// precondition: x.left is BLACK; x.right is RED
public Node rotateLeft(Node x) {
    Node y   = 1 ;
    x.right = 2 ;
    y.left   = 3 ;
    y.color = 4 ;
    x.color = 5 ;
    return     y         ;
}
```

A.	`BLACK`
B.	`RED`
C.	`x`
D.	`x.left`
E.	`x.right`
F.	`x.left.left`
G.	`x.left.right`
H.	`x.right.left`
I.	`x.right.right`
J.	`x.color`
K.	`null`

Recall that *left rotation* in a BST modifies the links in the tree as follows:



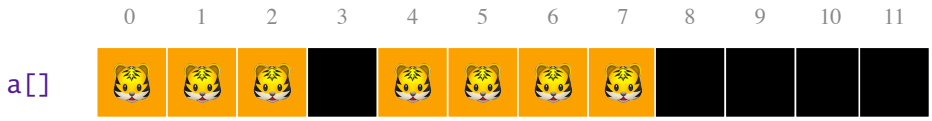
For each numbered oval above, write the letter of the corresponding expression on the right in the space provided. You may use each letter once, more than once, or not at all.

1	2	3	4	5

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

Consider an array `a[]` of length n in which each element is colored either orange or black, with `a[0]` orange and `a[n-1]` black. Design an algorithm to find two *adjacent* indices i and $i+1$ such that `a[i]` is orange and `a[i+1]` is black. Your algorithm must take $O(\log n)$ time in the worst case.

Example. If the array `a[]` consists of the following $n = 12$ elements, then the algorithm must output either $i = 2$ or $i = 7$.



Partial credit (for 50% credit). You may make the simplifying assumption that all of the orange elements appear before all of the black elements.

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, describe the modification. You may use code or pseudocode to improve clarity.

Your solution will be graded for correctness, efficiency, and clarity.

Are you attempting the partial or full credit solution? You may attempt only one.

◯ *partial credit* ◯ *full credit*

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

Design a data type for *union–find* that supports the *union* and *find* operations on a set of n elements. In particular, it should implement the following API:

`public class UF`	**description**
`UF(int n)`	*initialize with n singleton sets (0 to n – 1)*
`void union(int p, int q)`	*merge sets containing elements p and q*
`int find(int p)`	*return the leader of set containing element p*

Performance requirements (for 100% credit). The constructor must take $\Theta(1)$ time in the worst case. *So, for example, it cannot allocate an array of length n.* The `union()` and `find()` methods must each take $O(\log n)$ time in the worst case.

Performance requirements (for 50% credit). Same performance requirements as above but `union()` can take $O(n)$ time in the worst case.

Example. Here is a sample sequence of operations:

```
UF uf = new UF(1000000);       //  { 0 }, { 1 }, { 2 }, ..., { 999999 }
uf.union(0, 1);                //  { 0, 1 }, { 2 }, { 3 }, { 4 }, ...
uf.union(2, 3);                //  { 0, 1 }, { 2, 3 }, { 4 } , { 5 }, ...
uf.union(4, 6);                //  { 0, 1 }, { 2, 3 }, { 4, 6 }, { 5 } , ...
uf.find(0) == uf.find(6);      //  false
uf.union(1, 6);                //  { 0, 1, 4, 6 }, { 2, 3 }, { 5 }, ...
uf.find(0) == uf.find(6);      //  true
uf.find(226) == uf.find(2);    //  false
```

Note: the comments provide the disjoint sets in the union–find data type, but the API does not require you to maintain them in any particular order.

Are you attempting the partial or full credit solution? You may attempt only one.

◯ *partial credit* ◯ *full credit*

(a) *Using Java code*, declare the instance variables (along with any supporting nested classes) that you would use to implement `UF`. 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 the modifications.

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

(c) Give a concise English description of your algorithm for implementing the method `union(int p, int q)`. You may use code or pseudocode to improve clarity.

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