Name:

Instructions

1. Write your name at the top of the *first* page and your initials at the bottom of *every* page.
2. Do *not* staple the exam.
3. Return the exam with *all* the pages, arranged in *ascending* order.
4. This is a closed-book exam.
5. No electronic devices are permitted.
6. You may use the blank spaces for any scratch work.
7. Discussing the exam before the solutions have been posted is a violation of the Honor Code.
8. There are 9 problems on this exam and you have 75 minutes to answer them.
9. Problems 1 – 7 involve 19 multiple-choice questions, each worth 4 points. Each question must have *exactly one* response clearly marked in the circle provided — or else your answer will be marked incorrect.
10. Problems 8 (worth 16 points) and 9 (worth 8 points), must be answered clearly in the boxed space provided for those problems.

Problem 1. Insert the following keys in that order into a maximum-oriented heap-ordered binary tree:

```
B  Z  Q  K  V  F  S  N  I
```

a. What is the key with index 1?

- ○ B
- ○ I
- ● Z
- ○ N
- ○ F

b. What is the key with index 6?

- ○ Q
- ● F
- ○ S
- ○ K
- ○ N

c. If we perform a `delMax()` operation on the tree, what is the key that will replace the current maximum before it is sunk down?

- ○ F
- ○ B
- ● I
- ○ Q
- ○ K

Problem 2. Consider inserting the following key-value pairs in that order into a symbol table `st`.

key:	R	Q	J	G	L	R	M	I	Q	H	R	V
value:	1	2	3	4	5	6	7	8	9	10	11	12

a. What is the value returned by `st.size()`?

- ○ 12
- ○ 11
- ○ 8
- ● 9
- ○ 10

b. What is the value returned by `st.get("R")`?

- ○ 6
- ● 11
- ○ 3
- ○ 18
- ○ 1

Problem 3. Consider inserting the following keys (assume values to be non `null` and arbitrary) into a binary search tree (BST) symbol table `st`, an object of type `BST`.

G T J Q H Z K A O C M B

a. What is the height of the BST (assume root to be at height 0)?

- (●) 6
- ( ) 5
- ( ) 7
- ( ) 4
- ( ) 8

b. What is the value returned by `st.rank("M")`?

- ( ) 5
- ( ) 8
- (●) 7
- ( ) 6
- ( ) 4

c. What is the order in which the keys are visited if we traverse the BST in pre-order?

- ( ) A B C G H J K M O Q T Z
- ( ) G A C B T J H Q O K Z M
- ( ) B C A H M O K Q J Z T G
- ( ) G A C B T J H M K Z O Q
- (●) G A C B T J H Q K O M Z

d. What is the order in which the keys are visited if we traverse the BST in in-order?

- (●) A B C G H J K M O Q T Z
- ( ) A B C G H J K M Z Q O T
- ( ) A B C G H J K Q Z T O M
- ( ) B C A H M O K Q J Z T G
- ( ) G A C B T J H Q K O M Z

e. What is the order in which the keys are visited if we traverse the BST in post-order?

- ( ) B C A H M O K Z Q J G T
- ( ) B C A H M O K J G Q Z T
- ( ) A B C G H J K M O Q T Z
- (●) B C A H M O K Q J Z T G
- ( ) G A C B T J H Q K O M Z

Problem 4. Consider inserting the following keys into an initially empty 2-3 search tree.

B Q P F N W G J L H U X

a. What is the height of the tree that results (assume root to be at height zero)?

- ○ 3
- ○ 5
- ● 2
- ○ 4
- ○ 1

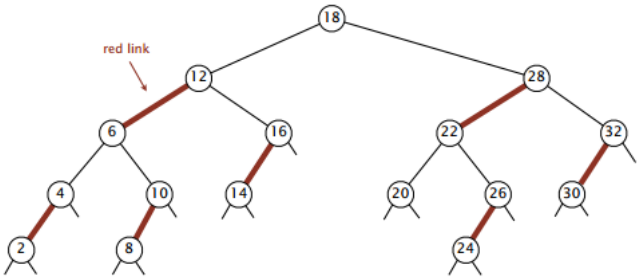
b. How many 2-nodes does the tree contain?

- ○ 6
- ○ 5
- ○ 3
- ○ 7
- ● 4

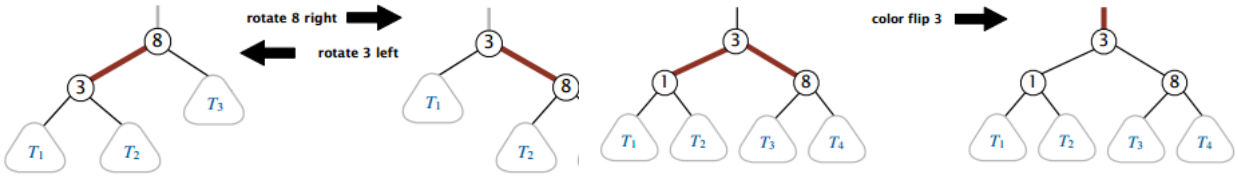
c. How many 3-nodes does the tree contain?

- ● 4
- ○ 6
- ○ 5
- ○ 3
- ○ 7

Problem 5. Suppose you insert the key 9 into the following left-leaning red-black BST:



Allowed operations (rotations and color flip):



a. What is the *third* operation that results?

- ○ `Rotate 8 left`
- ○ `Rotate 12 right`
- ○ `Rotate 10 right`
- ○ `Rotate 6 left`
- ● `Color flip 9`

b. What is the *fifth* operation that results?

- ● `Rotate 12 right`
- ○ `Rotate 6 left`
- ○ `Rotate 10 right`
- ○ `Color flip 9`
- ○ `Rotate 8 left`

Problem 6. Consider inserting the following keys (assume values to be non `null` and arbitrary) into an initially empty hash table of $M = 5$ lists, using separate chaining. Use the hash function $h(k) = k \bmod M$ to transform the kth letter of the alphabet into a table index, where $1 \leq k \leq 26$.

`J D W E V U L P F K X Y`

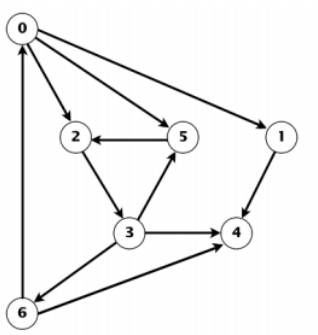
a. What is the length of the longest chain?

- ○ `1`
- ● `4`
- ○ `3`
- ○ `5`
- ○ `2`

b. Which of the following keys is in the longest chain?

- ○ `O`
- ○ `V`
- ○ `J`
- ○ `W`
- ● `U`

Problem 7. Consider the digraph shown below. Assume that, in the internal representation, all vertices appear in ascending order in each adjacency list.



a. Do a breadth-first search with 0 as the source vertex, and list the order in which vertices are processed by the algorithm?

- ○ 0 1 4 5 2 3 6
- ● 0 1 2 5 4 3 6
- ○ 0 1 3 2 4 5 6
- ○ 0 1 2 5 3 4 6
- ○ 0 1 2 3 6 4 5

b. Do a depth-first search with 0 as the source vertex, and list all vertices in pre-order.

- ○ 0 1 5 6 4 2 3
- ○ 0 1 2 3 5 6 4
- ○ 0 1 4 2 3 6 5
- ○ 0 1 3 5 6 4 2
- ● 0 1 4 2 3 5 6

Problem 8. (16 points) Design an efficient data structure called `ThreadedSet` to store a *threaded set of strings*, which maintains a set of strings (no duplicates) and the order in which the strings were inserted, according to the following API:

	Constructor/method	Description
a. (6 points)	`ThreadedSet()`	create an empty threaded set
b. (3 points)	`void add(String s)`	add the string s to the set (if it is not already in the set)
c. (3 points)	`boolean contains(String s)`	is the string s in the set?
d. (3 points)	`String previousKey(String s)`	the string added to the set immediately before s (`null` if s is the first string added; throw `java.util.NoSuchElementException` if s is not in the set)

Here is an example:

```
ThreadedSet set = new ThreadedSet();
set.add("aardvark");          // { "aardvark" }
set.add("bear");              // { "aardvark", "bear" }
set.add("cat");               // { "aardvark", "bear", "cat" }
set.add("bear");              // { "aardvark", "bear", "cat" }
                              // (adding a duplicate key has no effect)
set.contains("bear");         // true
set.contains("tiger");        // false
set.previousKey("cat");       // "bear"
set.previousKey("bear");      // "aardvark"
set.previousKey("aardvark"); // null
```

Your answer will be graded on correctness, efficiency, and clarity. You may use data types that we have considered in this course.

```
import edu.princeton.cs.algs4.*;
import java.util.NoSuchElementException;

public class ThreadedSet {
    private final static String NULL = "__NULL__";
    private String prev;
    private SeparateChainingHashST<String, String> st;

    public ThreadedSet() {
        prev = NULL;
        st = new SeparateChainingHashST<String, String>();
    }

    public void add(String s) {
        if (!contains(s)) {
            st.put(s, prev);
            prev = s;
        }
    }

    public boolean contains(String s) {
        return st.contains(s);
    }

    public String previousKey(String s) {
        if (!contains(s)) {
```

```
            throw new NoSuchElementException();
        }
        String key = st.get(s);
        return key.equals(NULL) ? null : key;
    }

    public static void main(String[] s) {
        ThreadedSet set = new ThreadedSet();
        set.add("aardvark");          // { "aardvark" }
        set.add("bear");              // { "aardvark", "bear" }
        set.add("cat");               // { "aardvark", "bear", "cat" }
        set.add("bear");              // { "aardvark", "bear", "cat" }
                                      // (adding a duplicate key has no effect)
        StdOut.println(set.contains("bear"));         // true
        StdOut.println(set.contains("tiger"));        // false
        StdOut.println(set.previousKey("cat"));       // "bear"
        StdOut.println(set.previousKey("bear"));      // "aardvark"
        StdOut.println(set.previousKey("aardvark")); // null
    }
}
```

e. (1 point) Under reasonable technical assumptions, what is the order of growth of each of the methods as a function of the number of keys N in the data structure? Assume that the length of all strings is bounded by a constant.

	1	$\log N$	$\sqrt{N}$	N	$N \log N$	N^2
`add()`	●	○	○	○	○	○
`contains()`	●	○	○	○	○	○
`previousKey()`	●	○	○	○	○	○

Problem 9. a. (6 points) Given two integer arrays `a[]` and `b[]`, find an integer that appears in both arrays (or report that no such integer exists). Let m and n denote the lengths of `a[]` and `b[]`, respectively, and assume that $m \leq n$.

Insert the integers from `b[]` into a hashtable `st`. Scan through the integers in `a[]` looking up each element `v` from `a[]` within `st`. If the lookup succeeds, write `v` and stop. Otherwise, continue. If none of the lookups succeed, report that `a[]` and `b[]` do not have an integer in common.

b. (2 points) What is the order of growth of the worst case running time of your algorithm?

$n + m$ (linear)