Exam 2

Due Mar 18, 2021 at 10:45am **Points** 100 **Questions** 25
Available until Mar 18, 2021 at 10:45am **Time Limit** 75 Minutes

Instructions

This exam consists entirely of multiple-choice questions and has a 75-minute time limit. For each question, choose the **best** response of those given.

(https://ht

This quiz is no longer available as the course has been concluded.

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	34 minutes	96 out of 100

Score for this quiz: **96** out of 100
Submitted Mar 18, 2021 at 10:04am
This attempt took 34 minutes.

Question 1 — 4 / 4 pts

Consider the three methods below on a **Set** collection; that is, a collection in which no duplicates are allowed and the values are not required to be kept in any particular order.

```
public void add(Comparable value)
public void remove(Comparable value)
public void contains(Comparable value)
```

Suppose we implement this collection using a **singly-linked list** for the physical storage of the elements, and suppose that we maintain the nodes in the linked list in **ascending order** relative to the values they contain. Based on the ideas and techniques we have discussed to this point in class, which of the time complexity profiles below characterize the most efficient implementation strategy that you could guarantee?

A. add $O(N)$, remove $O(N)$, contains $O(N)$

B. add $O(N)$, remove $O(N)$, contains $O(\log N)$

C. add $O(\log N)$, remove $O(\log N)$, contains $O(\log N)$

D. add $O(1)$, remove $O(1)$, contains $O(N)$

Correct!

- (●) A
- ( ) B
- ( ) C
- ( ) D

Question 2

4 / 4 pts

Consider the three methods below on a **Set** collection; that is, a collection in which no duplicates are allowed and the values are not required to be kept in any particular order.

```
public void add(Comparable value)
public void remove(Comparable value)
public void contains(Comparable value)
```

Suppose we implement this collection using a **singly-linked list** for the physical storage of the elements, and suppose that we maintain the nodes in the linked list in **no particular order**. Based on the ideas and techniques we have discussed to this point in class, which of the time complexity profiles below characterize the most efficient implementation strategy that you could guarantee?

A. add $O(N)$, remove $O(N)$, contains $O(N)$

B. add $O(N)$, remove $O(N)$, contains $O(\log N)$

C. add $O(\log N)$, remove $O(\log N)$, contains $O(\log N)$

D. add $O(1)$, remove $O(1)$, contains $O(N)$

Correct!

- (●) A
- ( ) B
- ( ) C
- ( ) D

Question 3

4 / 4 pts

Consider the three methods below on a **Set** collection; that is, a collection in which no duplicates are allowed and the values are not required to be kept in any particular order.

```
public void add(Comparable value)
public void remove(Comparable value)
public void contains(Comparable value)
```

Suppose we implement this collection using an **array** for the physical storage of the elements, and suppose that we maintain the array values in **ascending order**. Based on the ideas and techniques we have discussed to this point in class, which of the time complexity profiles below characterize the most efficient implementation strategy that you could guarantee? Assume amortization is included as appropriate to account for array resizing in those marked $O(1)$.

A. add $O(\log N)$, remove $O(\log N)$, contains $O(\log N)$

B. add $O(1)$, remove $O(1)$, contains $O(N)$

C. add $O(N)$, remove $O(N)$, contains $O(N)$

D. add $O(N)$, remove $O(N)$, contains $O(\log N)$

- ○ A
- ○ B
- ○ C
- ◉ D

Correct!

Question 4

4 / 4 pts

Consider the three methods below on a **Set** collection; that is, a collection in which no duplicates are allowed and the values are not required to be kept in any particular order.

```
public void add(Comparable value)
public void remove(Comparable value)
public void contains(Comparable value)
```

Suppose we implement this collection using an **array** for the physical storage of the elements, and suppose that we maintain the array values in **no specific order**. Based on the ideas and techniques we have discussed to this point in class, which of the time complexity profiles below characterize the most efficient implementation strategy that you could guarantee? Assume amortization is included as appropriate to account for array resizing in those marked $O(1)$.

A. add $O(1)$, remove $O(1)$, contains $O(N)$

B. add $O(\log N)$, remove $O(\log N)$, contains $O(\log N)$

C. add $O(N)$, remove $O(N)$, contains $O(\log N)$

D. add $O(N)$, remove $O(N)$, contains $O(N)$

- ○ A
- ○ B
- ○ C
- ◉ D

Correct!

Question 5

4 / 4 pts

Recall our implementation of the `ArrayBag` class that used a *dynamic resizing* strategy for the underlying array. Specifically, the `add` method checked to see if the array was full and used the `resize` method to double the capacity of the `elements` array, as shown below.

```
public boolean add(T element) {
   if (isFull()) {
      resize(size * 2);
   }
   elements[size] = element;
   size++;
   return true;
}
```

The `resize` method is $O(N)$, but we said that the overall `add` method was $O(1)$ by using what analysis technique?

A. best case analysis

B. worst case analysis

C. amortized analysis

D. doubling analysis

- ○ A
- ○ B
- ◉ C — Correct!
- ○ D

Question 6

4 / 4 pts

Consider the object `b` below, an instance of the `ArrayBag` class discussed in lecture and illustrated in lab. Recall that the `ArrayBag` uses an array as the physical storage structure and uses the *dynamic resizing* strategy exactly as we discussed in class. Assuming the array begins with capacity 1, what will the **capacity** (i.e., length) of the array be after the following sequence of statements are executed?

```
ArrayBag<String> b = new ArrayBag<>();

b.add("A");
b.add("B");
b.add("C");
b.add("D");
b.add("E");

b.remove("E");
b.remove("D");
b.remove("C");
b.remove("B");
```

A. 4

B. 8

C. 16

D. 32

Correct!

- (selected) A
- B
- C
- D

Question 7

4 / 4 pts

Consider the three methods below on a **Bag** collection; that is, a collection in which duplicates are allowed and the values are not required to be kept in any particular order.

```
public void add(Comparable value)
public void remove(Comparable value)
public void contains(Comparable value)
```

Suppose we observe the following performance profile for a particular implementation of this collection.

- `add`: $O(1)$
- `remove`: $O(N)$
- `contains`: $O(N)$

Which data structure option below is being used in this implementation, assuming that these worst-case time complexities are the best that could be achieved for the given data structure?

A. an array with values maintained in ascending natural order

B. an array with values maintained in no particular order

C. a singly-linked list of nodes with values maintained in ascending natural order

D. a doubly-linked list of nodes with values maintained in ascending natural order

- ○ A
- Correct! ◉ B
- ○ C
- ○ D

Question 8

4 / 4 pts

What *doubly-linked* list of nodes is accessible from `n` after the following statements have executed?

```
Node n = new Node(1);
n.prev = new Node(2);
n.next = new Node(3);
n.prev.next = n;
n.next.prev = n;
n = n.prev;
Node m = n.next;
Node p = new Node(4);
p.prev = m;
p.next = m.next;
m.next = p;
p.next.prev = p;
m = null;
p = null;
```

A. [3]

B. [4]⇆[3]

C. [1]⇆[2]⇆[3]⇆[4]

D. [2]⇆[1]⇆[4]⇆[3]

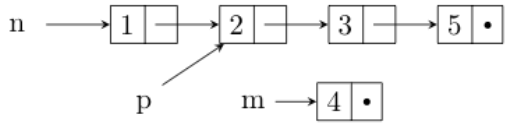
- ○ A
- ○ B
- ○ C
- ◉ D

Correct!

Question 9

4 / 4 pts

Suppose `n`, `m`, and `p` are references to `Node` objects as shown in the image below. Which set of statements would modify the pointer chain so that the nodes accessible from `n` are in the order 1, 2, 5, 4? (That is, the pointer chain referenced from `n` becomes `[1]→[2]→[5]→[4]`.)



A.
```
p.next = p.next.next;
p.next.next = m;
```

B.
```
m.next = p.next.next;
p.next = m;
```

C.
```
p.next = m;
m.next = p.next.next;
```

D.
```
m.next = p.next;
p.next = m;
m.next.next = p.next.next.next;
```

Correct!

- (●) A
- ( ) B
- ( ) C
- ( ) D

Question 10

4 / 4 pts

How many *singly-linked* `Node` objects could the Java Virtual Machine mark as "garbage" after all the following statements have executed?

```
Node n = null;
n = new Node("T");
n.next = new Node("I", new Node("G"));
Node m = n.next.next;
m.next = new Node("E", new Node("R"));
n = null;
```

A. 2

B. 3

C. 4

D. 5

Correct!

- (•) A
- ( ) B
- ( ) C
- ( ) D

Question 11

4 / 4 pts

What *singly-linked* list of nodes is accessible from `n` after the following statements have executed?

```
Node n = new Node("a");
n = new Node("b", n);
n.next = new Node("c", n.next);
n = new Node("d", n.next);
n.next.next = new Node("e", new Node("f"));
```

A. [a]→[b]→[c]→[d]→[e]→[f]

B. [f]→[e]→[d]→[c]→[b]→[a]

C. [b]→[c]→[d]→[e]→[f]

D. [d]→[c]→[e]→[f]

A

B

C

Correct!

D

Question 12

4 / 4 pts

Consider the three methods below on a **Bag** collection; that is, a collection in which duplicates are allowed and the values are not required to be kept in any particular order.

```
public void add(Comparable value)
public void remove(Comparable value)
public void contains(Comparable value)
```

Suppose we observe the following performance profile for a particular implementation of this collection.

- `add`: $O(N)$
- `remove`: $O(N)$
- `contains`: $O(N)$

Which data structure option below is being used in this implementation, assuming that these worst-case time complexities are the best that could be achieved for the given data structure?

A. an array with values maintained in ascending natural order

B. an array with values maintained in no particular order

C. a linked list of nodes with values maintained in ascending natural order

D. a red-black tree

A

B

Correct!

C

D

Question 13

4 / 4 pts

Consider the three methods below on a **Bag** collection; that is, a collection in which duplicates are allowed and the values are not required to be kept in any particular order.

```
public void add(Comparable value)
public void remove(Comparable value)
public void contains(Comparable value)
```

Suppose we implement this collection using a **singly-linked list** for the physical storage of the elements, and suppose that we maintain the nodes in the linked list in **ascending order** relative to the values they contain. Based on the ideas and techniques we have discussed to this point in class, which of the time complexity profiles below characterize the most efficient implementation strategy that you could guarantee?

A. add $O(1)$, remove $O(N)$, contains $O(N)$

B. add $O(\log N)$, remove $O(\log N)$, contains $O(\log N)$

C. add $O(N)$, remove $O(N)$, contains $O(\log N)$

D. add $O(N)$, remove $O(N)$, contains $O(N)$

A

B

C

Correct!

D

Question 14

0 / 4 pts

Consider the `RandomizedList` interface shown below.

```
public interface RandomizedList extends List {

    /**
     * Adds the specified element to this list.
     */
    void add(T element);

    /**
     * Selects and removes an element selected
     * uniformly at random from the elements
     * currently in the list.
     */
    T remove();

    /**
     * Selects but does not remove an element
     * selected uniformly at random from the
     * elements currently in the list.
     */
    T sample();

}
```

Suppose that the following performance guarantees are made for a particular implementing class for this interface.

- `add`: $O(1)$
- `remove`: $O(1)$
- `sample`: $O(1)$

Which data structure is being used in this implementing class?

A. an array with values maintained in no particular order

B. a linked list of nodes with values maintained in no particular order

C. an array with values maintained in ascending natural order

D. a linked list of nodes with values maintained in ascending natural order

orrect Answer

- ○ A
- ○ B
- ○ C

ou Answered

- ◉ D

Question 15

4 / 4 pts

Adding a new element to an ordered collection (like an indexed list) can be thought of as a two-part process: (1) Find the right spot, then (2) physically add the element. If an array is used as the underlying data structure for the collection, the right spot is identified by a legal index value in the array. Based on the concepts and techniques talked about to this point in the course, what is the worst-case time complexity of physically adding the element to the array, assuming the index of the right spot has already been found?

A. $O(N^2)$

B. $O(N)$

C. $O(\log N)$

D. $O(1)$

- ○ A
- Correct! ◉ B
- ○ C
- ○ D

Question 16

4 / 4 pts

Adding a new element to an ordered collection (like an indexed list) can be thought of as a two-part process: (1) Find the right spot, then (2) physically add the element. If a chain of linked nodes is used as the underlying data structure for the collection, the right spot is identified by a reference to a node in the chain. Based on the concepts and techniques talked about to this point in the course, what is the worst-case time complexity of physically adding the element to the chain of nodes, assuming the reference to the right spot has already been found?

A. $O(1)$

B. $O(\log N)$

C. $O(N)$

D. $O(N^2)$

Correct!

- (●) A
- ( ) B
- ( ) C
- ( ) D

Question 17

4 / 4 pts

Consider the three methods below on a **SortedList** collection; that is, a collection in which elements are stored according to the defined *natural order* of the underlying data type.

```
public void add(Comparable element)
public Comparable remove(Comparable element)
public boolean contains(Comparable element)
```

Suppose we implemented this collection using an **array** for the physical storage of the elements. Based on the ideas and techniques we have discussed to this point in class, which of the worst-case time complexity profiles below characterize the most efficient implementation strategy that you could guarantee? Assume amortization is included as appropriate to account for array resizing in those marked $O(1)$.

A. add $O(1)$, remove $O(N)$, contains $O(N)$

B. add $O(1)$, remove $O(1)$, contains $O(1)$

C. add $O(N)$, remove $O(N)$, contains $O(\log N)$

D. add $O(N)$, remove $O(N)$, contains $O(N)$

- ○ A
- ○ B
- Correct! ◉ C
- ○ D

Question 18

4 / 4 pts

Consider the three methods below on a **SortedList** collection; that is, a collection in which elements are stored according to the defined *natural order* of the underlying data type.

```
public void add(Comparable element)
public Comparable remove(Comparable element)
public boolean contains(Comparable element)
```

Suppose we implemented this collection using a chain of **linked nodes** for the physical storage of the elements. Based on the ideas and techniques we have discussed to this point in class, which of the worst-case time complexity profiles below characterize the most efficient implementation strategy that you could guarantee?

A. add $O(1)$, remove $O(N)$, contains $O(N)$

B. add $O(N)$, remove $O(N)$, contains $O(N)$

C. add $O(1)$, remove $O(1)$, contains $O(1)$

D. add $O(N)$, remove $O(1)$, contains $O(\log N)$

- ○ A
- Correct! ◉ B
- ○ C
- ○ D

Question 19

4 / 4 pts

Recall the stack-based "shunting yard algorithm" from class for evaluating postfix expressions. If it were applied to the following postfix expression, what would the stack contain **after** the first $\times$ has been processed but **before** the subsequent $\times$ has been read?

$1 \quad 2 \quad 3 \quad + \quad 4 \quad 5 \quad \times \quad \times \quad +$

A. *top* | 101

B. *top* | 20, 5, 1

C. *top* | 5, 4, 5, 1

D. *top* | $\times$, 5, 4, $+$, 3, 2, 1

- ○ A
- Correct! ◉ B
- ○ C
- ○ D

Question 20

4 / 4 pts

What does the stack `s` contain after the following sequence of operations? Note that the top of stack `s` is the left-most element listed.

```
s.push("I");
s.push("believe");
s.push("in");
s.push("Auburn");
s.pop();
s.push("and");
s.pop();
s.pop();
s.push("love");
s.pop();
s.pop();
s.push("it");
```

A. *top* | love, it

B. *top* | it, love

C. *top* | I, it

D. *top* | it, I

- ○ A
- ○ B
- ○ C
- ◉ D

Correct!

Question 21

4 / 4 pts

Using a "circular array" implementation strategy for a queue collection provides what time complexity for the `enqueue` and `dequeue` operations?

A. `enqueue` $O(1)$, `dequeue` $O(N)$

B. `enqueue` $O(N)$, `dequeue` $O(1)$

C. `enqueue` $O(1)$, `dequeue` $O(1)$

D. `enqueue` $O(N)$, `dequeue` $O(N)$

A

B

Correct! C

D

Question 22

4 / 4 pts

Suppose the object `q` below is an instance of a class that uses a "circular array" scheme to implement a queue. Assume that the `front` and `rear` markers begin at index 0. Suppose also that the queue has a fixed capacity of 5. That is, there is no array resizing done.

Which choice below depicts the contents of the internal array after the following sequence of operations? (The array is shown from left to right beginning at index 0. The symbol • is used to denote an empty cell.)

```
q.enqueue("W");
q.enqueue("A");
q.enqueue("R");
q.enqueue("E");
q.dequeue();
q.enqueue("A");
q.enqueue("G");
q.dequeue();
q.dequeue();
q.enqueue("L");
q.dequeue();
q.enqueue("E");
```

A. `[G, L, E, •, A]`

B. `[G, L, E, A, •]`

C. `[A, G, L, E, •]`

D. `[•, A, G, L, E]`

Correct! A

B

C

D

Question 23

4 / 4 pts

What does the queue `q` contain after the following sequence of operations? Note that the front of queue `q` is the left-most element listed and the rear of queue `q` is the right-most element listed.

```
q.enqueue("I");
q.enqueue("believe");
q.enqueue("in");
q.enqueue("Auburn");
q.dequeue();
q.enqueue("and");
q.dequeue();
q.dequeue();
q.enqueue("love");
q.dequeue();
q.dequeue();
q.enqueue("it");
```

A. *front* | I, it | *rear*

B. *front* | it, I | *rear*

C. *front* | love, it | *rear*

D. *front* | it, love | *rear*

A

B

Correct!

C

D

Question 24

4 / 4 pts

The numbers in the two-dimensional array below indicate the order in which each position in the array was examined by a search algorithm. Which search algorithm does this sequence suggest most strongly?

26	27	28	29	30
17	18	19	22	23
20	10	11	12	15
21	13	2	3	4
24	14	5	1	6
25	16	7	8	9

A. depth-first search

B. breadth-first search

- ○ A
- Correct! ◉ B

Question 25

4 / 4 pts

The numbers in the two-dimensional array below indicate the order in which each position in the array was examined by a search algorithm. Which search algorithm does this sequence suggest most strongly?

4	3	2	1	9
5	6	7	8	10
14	13	12	11	19
15	16	17	18	20
24	23	22	21	29
25	26	27	28	30

A. depth-first search

B. breadth-first search

- Correct! ◉ A
- ○ B

Quiz Score: **96** out of 100