Exam 2

Due Mar 21, 2017 at 9:15pm **Points** 28 **Questions** 28
Available until Mar 21, 2017 at 9:16pm **Time Limit** 75 Minutes

Instructions

Choose the **best** answer from the options provided.

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

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	21 minutes	22 out of 28

Score for this quiz: **22** out of 28
Submitted Mar 21, 2017 at 8:21pm
This attempt took 21 minutes.

Question 1 — 1 / 1 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 **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 $O(1)$, 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 $O(N)$

Correct!

- (•) A

B

C

D

Question 2

1 / 1 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 $O(1)$, 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

Correct!

C

D

Question 3

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

A. linear search

B. binary search

C. depth-first search

D. breadth-first search

- A
- B

Correct!

- C
- D

Question 4

1 / 1 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(1); q.enqueue(2); q.enqueue(3);
q.enqueue(q.dequeue());
q.dequeue();
q.enqueue (4); q.enqueue(5);
q.dequeue();
```

A. *front* | 1, 2, 4 | *rear*

B. *front* | 1, 4, 5 | *rear*

C. *front* | 5, 4, 1 | *rear*

D. *front* | 4, 2, 1 | *rear*

- A

Correct!

- B

C

D

Question 5

1 / 1 pts

According to the definitions discussed in class, which of the following terms would be most appropriate when describing a specific physical way of storing and connecting data in a program?

A. abstract data type

B. collection

C. data structure

D. interface

A

B

Correct!

C

D

Question 6

0 / 1 pts

Which of the following statements correctly creates the object `array` as an array of 10 elements of type `T`, where `T` is a generic type variable?

A. `T[] array = new T[10];`

B. `T[] array = new Object[10];`

C. `Object<T>[] array = new Object<T>[10];`

D. `T[] array = (T[]) new Object[10];`

A

B

'ou Answered

C

orrect Answer

D

Question 7

1 / 1 pts

Both the JCF and the course notes are careful to design and implement a collection in terms of an interface and one or more classes that implement that interface. Which of the following is **not** a benefit that this design pattern offers?

A. Interfaces offer a more efficient solution than classes.

B. Interfaces provide a way of separating the collection specification from the collection implementation.

C. Interfaces allow client code to be largely independent of any specific implementing class.

D. Interfaces promote generic, reusable code.

Correct!

A

B

C

D

Question 8

1 / 1 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 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 $O(N)$, 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 $O(N)$

Correct!

- (•) A
- ( ) B
- ( ) C
- ( ) D

Question 9

1 / 1 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;
for (int i = 0; i < 5; i++) {
   n = new Node(i, n);
}
Node m = n.next.next;
n = null;
```

A. 5

B. 4

C. 3

D. 2

- ( ) A

B

C

Correct!

D

Question 10

1 / 1 pts

Recall the stack-based postfix evaluation algorithm from class (the "shunting yard algorithm"). If it were applied to the following postfix expression, what would the stack contain **after** the `6` has been processed but **before** the `/` has been read? The top of the stack is the left-most element listed.

Postfix: `8 4 * 2 3 7 + * - 6 /`

A. *top* | 2

B. *top* | 12

C. *top* | 6, 12

D. *top* | 20, 32

A

B

Correct!

C

D

Question 11

0 / 1 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. Assuming the index of the right spot has already been found, what is the worst-case time complexity of then physically adding the element to the array?

A. $O(1)$

B. $O(\log N)$

C. $O(N)$

D. $O(N^2)$

'ou Answered

- (•) A

- ( ) B

orrect Answer

- ( ) C

- ( ) D

Question 12

0 / 1 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 $O(1)$, 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

'ou Answered

- (•) B
- ( ) C

orrect Answer

- ( ) D

Question 13

1 / 1 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(1); s.push(2); s.push(3);
s.push(s.pop());
s.pop();
s.push (4); s.push(5);
s.pop();
```

A. *top* | 1, 4, 5

B. *top* | 5, 4, 1

C. *top* | 4, 2, 1

D. *top* | 1, 2, 4

- ( ) A
- ( ) B

Correct!

- (•) C
- ( ) D

Question 14

1 / 1 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 b = new ArrayBag();
for (int i = 1; i <= 18; i++) {
   b.add(i);
}
for (int i = 1; i <= 15; i++) {
   b.remove(i);
}
```

A. 4

B. 8

C. 16

D. 32

- A
- B

Correct!

- C
- D

Question 15

1 / 1 pts

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

```
Node n = new Node(1);
n.next = new Node(2, new Node(3));
n.next = n.next.next;
n = new Node(4, n);
n.next.next = new Node(5);
n.next = new Node(6, n.next);
```

A. `[1]→[2]→[3]→[4]→[5]→[6]`

B. `[6]→[5]→[4]→[3]→[2]→[1]`

C. `[4]→[6]→[1]→[5]`

D. `[4]→[5]→[6]`

- A

B

Correct!

C

D

Question 16

1 / 1 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?

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

A. linear search

B. binary search

C. depth-first search

D. breadth-first search

A

B

C

Correct!

D

Question 17

1 / 1 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. [1]⇆[2]⇆[3]⇆[4]

B. [2]⇆[1]⇆[4]⇆[3]

C. [4]⇆[3]

D. [3]

- ○ A
- Correct! ◉ B
- ○ C
- ○ D

Question 18

1 / 1 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 $O(1)$, 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 19

1 / 1 pts

According to the definitions discussed in class, which of the following terms would be most appropriate when describing a general or conceptual way of organizing and providing controlled access to data?

A. abstract data type

B. collection

C. data structure

D. interface

- ○ A
- ◉ B
- ○ C

Correct!

D

Question 20

1 / 1 pts

Suppose we were to implement a queue using an array such that both `enqueue` and `dequeue` have $O(1)$ worst-case time complexity. Recall that to ensure constant time operations, we had to treat the array as *circular*. 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 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(1); q.enqueue(2); q.enqueue(3); q.enqueue(4);
q.dequeue(); q.dequeue();q.dequeue();
q.enqueue(5); q.enqueue(6); q.enqueue(7);
q.dequeue();
```

A. `[5, 6, 7, •, •]`

B. `[6, 7, •, •, 5]`

C. `[•, •, 5, 6, 7]`

D. `[•, 7, 6, 5, •]`

A

Correct!

B

C

D

Question 21

0 / 1 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 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 $O(1)$, 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 $O(N)$

'ou Answered

- (•) A

- ( ) B

orrect Answer

- ( ) C

- ( ) D

Question 22

0 / 1 pts

According to the definitions discussed in class, which of the following terms would be most appropriate when describing a language construct that allows an object to be declared and used as a collection in a given programming language?

A. abstract data type

B. collection

C. data structure

D. interface

orrect Answer

- ( ) A

- ( ) B

C

'ou Answered

D

Question 23

1 / 1 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

Correct!

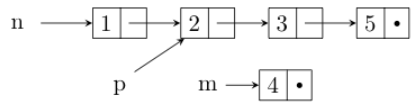
C

D

Question 24

1 / 1 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;
```

'ou Answered

- (selected) A

orrect Answer

- B
- C
- D

A is the correct answer.

Question 25

1 / 1 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. Assuming the reference to the right spot has already been found, what is the worst-case time complexity of then physically adding the element to the chain of linked nodes?

A. $O(1)$

B. $O(\log N)$

C. $O(N)$

D. $O(N^2)$

Correct!

- (●) A
- ( ) B
- ( ) C
- ( ) D

Question 26

0 / 1 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 $O(N)$

'ou Answered

- (●) A

B

C

orrect Answer

D

Question 27

1 / 1 pts

Recall the `RandomizedList` interface from Assignment 4, reproduced below.

```
public interface RandomizedList extends List {

   /**
    * Adds the specified element to this list. If the element is nul
    * method throws an IllegalArgumentException.
    */
   void add(T element);

   /**
    * Selects and removes an element selected uniformly at random fr
    * elements currently in the list. If the list is empty this meth
    * null.
    */
   T remove();

   /**
    * Selects but does not remove an element selected uniformly at r
    * the elements currently in the list. If the list is empty this
    * return null.
    */
   T sample();

}
```

Assume that `LinkedRandomizedList` is a class that implements the `RandomizedList` interface and uses a pointer chain of linked nodes as the underlying data structure. What is the best performance profile that could result from this implementation choice in terms of worst-case big-O?

A. `add`: $O(1)$, `remove`: $O(1)$, `sample`: $O(1)$

B. `add`: $O(1)$, `remove`: $O(N)$, `sample`: $O(N)$

C. `add`: $O(N)$, `remove`: $O(N)$, `sample`: $O(N)$

D. `add`: $O(N)$, `remove`: $O(1)$, `sample`: $O(1)$

A

Correct!

B

C

D

Question 28

1 / 1 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

Correct!

D

Quiz Score: **22** out of 28