

Read each question carefully. When writing code, note that not all questions require a complete class, or even a complete method. Keep in mind that this review is not intended to be comprehensive. While studying for the exam, it would be prudent to consider variations on the questions presented here.

Under what circumstances would it be preferable to implement a sequence using a doubly linked list rather than an array? Specifically which operations would be more efficient & under what circumstances.

Write a generic method called **reverse** that takes an array and reverses it.

In big O notation indicate the performance of the following operations of IntLinkedListBag:

```
// add a new item to the bag
add(int item)

// remove the all occurrences of the item from the bag
// return success (if it was found) or failure (if it wasn't there)
remove(int item)

// count the number of times item is in the bag
countOccurrences(int item)

// get the total number of items in the bag
size()

// make a new bag identical to the combination of b1 & b2
union(IntArrayBag b1, IntArrayBag b2)
```

In big O notation indicate the performance of the following operations of IntLinkedListSequence (assume the list is NOT doubly linked)

```
// adds an item to the sequence after the cursor
addAfter(int item)

// adds an item to the sequence before the cursor
addBefore(int item)

// count the number of times item is in the list
countOccurrences(int item)

// create a new duplicate sequence & return it
clone()

// get the current number of items in the list
size()
```

In big O notation indicate the performance of the following operations of IntDoublyLinkedListSequence (assume the list IS doubly linked)

```
// adds an item to the sequence after the cursor
addAfter(int item)

// adds an item to the sequence before the cursor
addBefore(int item)

// count the number of times item is in the list
countOccurrences(int item)

// get the current number of items in the list
size()
```

In big O notation indicate the performance of the following operations of IntArrayStack where the top of the stack is the zero element of the array:

```
// look at the top element
peek()

// remove the top most element
pop()

// place a new element onto the stack
push()
```

In big O notation indicate the performance of the following operations of IntArrayStack where the bottom of the stack is the zero element of the array:

```
// look at the top element
peek()

// remove the top most element
pop()

// place a new element onto the stack
push()
```

Assume you were reading in a series of paren ( ) and square bracket [ ] characters one at a time. Describe the steps of an algorithm which uses a stack to determine at each step if the string is balanced.

Below are 6 example groups of input & output:

```
input:  ( )[]      ( [ ] ] ) ( )      ( ( ) [ ] [ ] )      ] ( ) [ ]      ( ) ( [ ] ]      ( ) [ ] [ ( )
output: FTFT      FFFFFTFT      FFFFFFFFFFT      FFFFF      FTFFFF      FTFTFFF
```

Assume you had a doubly linked list. Describe the steps necessary to add a new node before the cursor.

Assume you had a singly linked list. Describe the steps necessary to move the cursor back 1 element (towards the head)

What are comparators? Why are they useful? Under what conditions are they a better choice than using the Comparable interface?

What is the advantage of using a generic Bag class instead of a Bag class designed to hold Objects?

Comparators are best suited to situations where:

- A. There is a natural ordering of elements.
- B. The ordering of elements is strict I.E.  $a < b$  is allowed, but  $a \leq b$  is not
- C. You have access to the source code for the elements being compared
- D. The ordering of elements is unimportant
- E. You need to deviate from the natural ordering of elements
- F. None of the above

In linked lists, dummy nodes are:

- A. Nodes that serve as place holders for head & tail
- B. Nodes that are abandoned by the remove method
- C. Nodes that serve as place holders for cursor & precursor
- D. Nodes that only occur in a doubly linked list
- E. Nodes that link to null
- F. None of the above

To make full use of an external iterator, a container class must implement interface(s)

- A. iterator
- B. iterable
- C. both iterator & iterable are required
- E. either iterator or iterable will work
- D. no interface implementation is required
- F. none of the above

To make use of an external iterator, a class designed to print the contents of a container class must implement interface(s)

- A. iterator
- B. iterable
- C. both iterator & iterable are required
- E. either iterator or iterable will work
- D. no interface implementation is required
- F. none of the above

Compared to a linked list, an array based list is uniquely suited for:

- A. lots of insertions & removals at or after the cursor
- B. lots of insertions & removals throughout the list
- C. lots of read operations at or after the cursor
- D. lots of read operations throughout the list
- F. lots of operations of any type at the head of the list
- F. none of the above