

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 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 items 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 bag
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 LinkedListStack where the top of the stack is the tail of the linked list:

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

// remove the top most element
pop()

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

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

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.

Examples:

input: () []	([] []) ()	(() [] [] ())] () []	() ([]]	() [] [] ()
output: FTFT	FFFFFFTFT	FFFFFFFFFT	FFFFF	FTFFFF	FTFTFFF

What are iterators? Why are they useful? In terms of code, what is required to make use of them?

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

Entries in a stack are "ordered". What is the meaning of this statement?

- A. A collection of Stacks can be sorted.
- B. Stack entries may be compared with the '<' operation.
- C. The entries must be stored in a linked list.
- D. There is a first entry, a second entry, and so on.
- E. The entries rely on a Comparator class
- F. None of the above

Which of the following stack operations could result in stack underflow?

- A. isEmpty
- B. pop
- C. push
- D. clone
- E. Two or more of the above answers
- F. None of the above

When using a doubly linked list for a stack, what is the run-time behavior for the pop method?

- A. linear - but only if the head is used as the stack top
- B. linear - but only if the tail is used as the stack top
- C. linear - regardless of whether stack top is the tail or the head
- D. constant - but only if the tail is used as the stack top
- E. constant - regardless of whether stack top is the tail or the head
- F. none of the above

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