**Name:**

**True/False**

1. In an array-based implementation of the Stack ADT, it is more efficient to have the first array location reference the top of the stack. False
2. In a linked-chain implementation of the Stack ADT, the first node references the stack’s top entry. True
3. The first item added to a stack is the first one removed. False
4. Infix expressions are easier to evaluate than postfix expressions. False
5. You can only pop from the top of the stack but you can peek at any entry on the stack. False
6. An array potentially wastes more space than a chain. True

**Multiple choice**

1. In a linked-chain implementation of a Stack ADT, the performance of pushing an entry onto the stack is
   1. O(1)
   2. O(2)
   3. O(n)
   4. O(n2)
2. In a linked-chain implementation of a Stack ADT, the performance of popping an entry from the stack is
   1. O(1)
   2. O(2)
   3. O(n)
   4. O(n2)
3. Given the following infix expression, which one of the following is the corresponding postfix expression? a + r ^ 2 – 5
   1. a r 2 ^ + 5 -
   2. a r + ^ 5 –
   3. a r 2 5 + ^ -
   4. none of the above
4. Given the following infix expression, which one of the following is the corresponding postfix expression? w + x \* y / z
   1. w x y \* z / +
   2. w x + y z \* /
   3. w x y q + \* /
   4. none of the above
5. What is the entry returned by the *peek* method after the following stack operations.  
   push(A), push(R), pop(), push(D), pop(), push(L), pop(), push(J), push(S), pop(), pop()
   1. A
   2. S
   3. L
6. For large values of n which statement is true?
   1. 2n3 + 4n2 + 17n behaves like n3
   2. 2n3 + 4n2 behaves like n3
   3. 2n3 behaves like n3
   4. all of the above
7. The effect of doubling the input size on an algorithm with time complexity O(n3) is
   1. 8 times
   2. 3 times
   3. Double
   4. Negligible
8. What is the best-case time complexity for searching a fixed-size array-based bag ADT for a particular entry?
   1. O(1)
   2. O(n)
   3. O(n2)
   4. negligible
9. A call to the *remove* method with no arguments
   1. removes the first node
   2. removes the last node
   3. removes a random node

asks the user which node to remove

1. Which one of the following Java statements allocates an array in the bag constructor causing a compiler warning for an unchecked operation? Assume *capacity* is an integer.
   1. bag = (T[ ]) new Object[capacity];
   2. bag = new T[capacity];
   3. bag = new Object[capacity];
   4. bag = new (T[ ]) Object[capacity];
2. A fixed size array
   1. has a limited capacity
   2. can waste memory
   3. prevents expansion when the bag becomes full
   4. all of the above
3. Which of the following is a disadvantage of using an array to implement the ADT bag?
   1. increasing the size of the array requires time to copy its entries
   2. adding an entry to a bag is fast
   3. removing an unspecified entry from the array is fast
   4. all of the above
4. To accommodate entries of any class, the bag methods use \_\_\_\_\_\_ .
   1. a generic type
   2. an inherited type
   3. a sub class
   4. all of the above

**Short Answer (5)**

1. In an array based implementation of a Stack ADT, explain why it is a bad idea to use the first location of the array to reference the top of a stack.

You must move all of the entries in the array every single time you push or pop an entry causing excessive time.

1. Draw the resultant stack after the following operations. Label the top entry of your stack.  
   push(X), push(Q), push (Y), peek(), pop(), push(T), peek()

Answer: top -> T Q X

1. Convert the following infix expression to a postfix expression.  
   w \* (x + y) / z

Answer: w x y + \* z /

1. Convert the following infix expression to a postfix expression.  
   (a + b) \* (c – d) / (e – f)

Answer: a b + c d - \* e f -/

1. Using the evaluatePostfix algorithm, evaluate the following postfix expression.   
   a b c \* + d -  
   Assume that a = 10 , b = 2, c = 5, d = 3 .

Answer: 17

1. Use a stack to check whether the string/expression [a{b/(c-d) + e } – h]” is balanced or not:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |

1. Convert the following infix expression to a postfix expression.  
   (a – b \* c) / (d \* e ^ f \* g + h)
2. Order the following growth rates from smallest to largest.  
   *n2 n! n log n 2n  n log n*

Answer: *log n n n log n n2  2n  n!*

1. What is the Big-O time complexity for an algorithm to display the nth integer in an array of integers?

O(1) because the array can be indexed by n.

1. What is the Big-O time complexity for an algorithm to display the nth integer in a linked chain of integers?

O(n) because the entire chain potentially must be traversed if the index if n is the last element.

1. Given f(n) = 4n + 63 + 5n2 + 3n log n what is g(n)?

g(n) = (n2) the dominant term

1. Explain how to implement the *clear* method in a chain.

Answer: Set the *firstnode* to null.

**Code**

1. Consider the following Java statements, assuming that MyStack is a class that implements the interface StackInterface:

**int** n = 4;

Stack<Integer> stack = **new** MyStack<>();

**while**(n > 0){

stack.push(n);

n--;

}

**int** result = 1;

**while**(!stack.isEmpty()){

**int** integer = stack.pop();

result = result \* integer;

}

System.***out***.println("Result = " + result);

Result = 24

1. Write Java code that displays all the objects in a stack in the order in which they were pushed onto it. After all the objects are displayed the stack should have the same contents as when you started. (Hint: Use a helper stack)

**static** **void** display(Stack<Integer> stack){

Stack<Integer> helper = **new** ResizableArrayStack<>();

**while**(!stack.isEmpty()){

helper.push(stack.pop());

}

**while**(!helper.isEmpty()){

Integer t = helper.pop();

stack.push(t);

System.***out***.println(t);

}

}

1. Complete the pop and push methods

**public** **class** LinkedStack<T> **implements** Stack<T> {

**private** Node head;

@Override

**public** T pop() {

**if**(head == **null**)

**throw** **new** EmptyStackException();

T result = head.value;

head = head.next;

**return** result;

}

@Override

**public** **void** push(T newItem) {

Node n = **new** Node(newItem, head);

head = n;

}

}

1. Complete the following code:

**public** **class** LinkedList<T> {

Node<T> head;

**int** numOfElements;

/\*\*

\* Creates an empty linked list (no nodes)

\*/

**public** LinkedList(){

head = **null**;

numOfElements = 0;

}

/\*\*

\* Adds the specified data to the end linked list.

\* **@param** value The data to be added

\*/

**public** **void** addToTail(T value){

Node<T> curNode = **new** Node<T>(value, **null**);

**if**(head == **null**) {

head = curNode;

}**else**{

Node<T> p = head;

**while**(p.getNext() != **null**){

p = p.getNext();

}

p.setNext(curNode);

}

numOfElements++;

}

/\*\*

\* Removes the first item from the linked list, if one exists

\* **@return** the first item from the linked list, or null if the

\* linked list contains no items

\*/

**public** Node<T> removeFirst(){

**if**(head == **null**) **return** **null**;

Node<T> results = head;

head = head.getNext();

numOfElements--;

**return** results;

}

}