

Instructor: Alex Ryba

These problems were given on exams for this course. Some older problems did not make use of generics in Java, but generic implementations are now required in this course.

Problem 1 The generic class Queue is to be programmed with an array based implementation. A partial version of the implementation follows. The two most important methods have been omitted. The treatment of an empty Queue (as given by the constructor) is different from the one we used in class. Make sure that your methods work correctly with the choices made by the constructor.

```
public class P2<T> implements Queue<T> {
    private T data[];
    private int front, rear, size;
    public P2() { data = (T[]) new Object[100]; front = size = 0; rear = -1; }
    public int size() { return size; }
    public boolean empty() { return size == 0; }
    // methods omitted here
}
```

- (a) **Identify the two missing methods.** For each give the name, parameters and return type.
- (b) **Give a complete implementation** of **ONE** of the two missing Queue methods. (You can choose either of them.)

Problem 2 (i) Write the Java interface for a generic ADT Stack. (Just write an interface that specifies methods — **do not** implement the methods.)

(ii) Write the Java interface for a generic ADT Iterator. (Just write an interface that specifies methods — **do not** implement the methods.)

(iii) Write a client function with title line:

```
public static <T> int count(Stack<T> s, T x)
```

The function returns a count of the number of times the object x is found on the Stack s . When the method ends the Stack should store exactly the same data as at the beginning of the method. However your function will need to alter the Stack data as it runs.

(iv) Write a client function with title line:

```
public static <T> int count(Iterator<T> i, T x)
```

The function returns a count of the number of times the object x is found on the Iterator i . This function can and should empty out the Iterator and should not restore it to its original state.

Problem 3 Consider the following Java program that uses a standard Stack, Iterable Queue and Priority Queue as studied in class.

```
public static void main(String args[]) throws Exception {
    Stack<Integer> s = new ArrayStack<Integer>();
    ItQueue<Integer> q = new ItQueue<Integer>();
    PriorityQueue<String> p = new HeapPriorityQueue<String>();
    for (int i = 5; i < 10; i++) {
        s.push(i);
        q.enqueue(i + 10);
    }
    System.out.println(q.dequeue()); // line (a)
    System.out.println(s.pop()); // line (b)
```

```

    System.out.println(q.dequeue() + q.dequeue());           // line (c)
    for (Integer x:q) {
        int y = s.pop();
        p.add("" + y);
        p.add("" + x);
        System.out.print(x);                               // line (d)
    }
    System.out.println();
    for (int i = 1; i <= 4; i++)
        System.out.print(p.removeMin());                   // line (e)
    System.out.println();
}

```

(a) What is the output printed from line (a)?

Answer:

(b) What is the output printed from line (b)?

Answer:

(c) What is the output printed from line (c)?

Answer:

(d) What is the output printed from line (d)?

Answer:

(e) What is the output printed from line (e)?

Answer:

Problem 4 Consider the following method that adds some elements from an array with at least n elements to an Iterable Stack. You should assume that a standard stack iterator begins with the element at the bottom of the stack and ends with the element at the top — this is the behavior of `java.util.Stack`.

```

public static Stack<Integer> method(int array[], int n) {
    Stack<Integer> answer = new Stack<Integer>();
    if (n == 0) return answer;
    for (int i = 0; i < n; i+=2)
        answer.push(array[i]);
    Stack<Integer> temp = method(array, n/2);
    for (Integer x:temp) answer.push(x);
    return answer;
}

```

(a) What is the output when the method is called in the following:

```

public static void main(String args[]) {
    int a[]={1, 2, 3, 4, 5, 6, 7, 8};
    Stack<Integer> s = method(a, 8);
    for (Integer x:s) System.out.print(x);
    System.out.println();
}

```

Answer:

(b) Give the recurrence relation that is satisfied by the run time $t(n)$ of the method.

Answer:

(c) Apply the Master Theorem to your answer to (b) to give a useful Θ estimate for $t(n)$.

Answer:

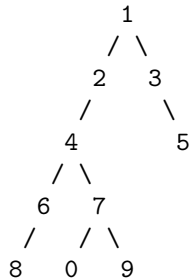
Problem 5 (a) Write complete code for the standard interface `Iterator`.

Answer:

(b) Write complete code for the class `ListAdapterStack`.

Answer:

For the following binary tree:



(c) What is the height of the tree?

Answer:

(d) What is the depth of the node that stores 7?

Answer:

(e) What is the postorder traversal?

Answer:

Problem 6 The following is part of a circular array based model for a `Deque`. The method `addLast` has logic errors and does not operate correctly. Other required methods have been entirely omitted.

```
public class Deque<T> {
    private T data[];
    private int front, rear, size;
    // the next items to be added at front and rear will be placed at indices front and rear

    public Deque() { // This is correct code.
        data = (T[]) new Object[100];
        front = size = 0; rear = 1;
    }

    public T addLast(T x) throws Exception { // This has errors.
        ++size;
        if (size > 100) throw Exception("Deque Full");
        data[++rear] = x;
        if (rear > 100) rear = 0;
    }

    // other methods omitted
}
```

(a) Correct the code for `addLast`. Your solution should not use more more than one additional line of code and should be based on that given. It must correct any errors.

Answer:

(b) Give a title line for each required method that has been omitted. (Credit will be given for the 3 most important methods only.)

Answer: