Mihir Jham
CS251
Written HW#1
02/13/2014

CS251 - Spring 2014
HOMEWORK 1

Total: **50 points**

Due: Thursday, **February 13th, 2014, by 11:59 PM**. Only **PDF** files will be accepted. **No extensions or late submissions** since solutions will be released immediately after the deadline.

Submit from data.cs.purdue.edu or lore.cs.purdue.edu using the command:

```
turnin -c cs251 -p hw1 your_folder_name
```

1. Stacks (6 points)

The following list gives possible choices for using a dynamic array in a stack implementation. Write **linear** or **quadratic** in the blank following each choice to best describe the total time required in the worst case for a sequence of `push()` and `pop()` operations.

A. `push()`: always grow array by 1
`pop()`: always shrink array by 1 — Quadratic

B. `push()`: double array if it is full
`pop()`: never shrink array — Linear

C. `push()`: double array if it is full
`pop()`: halve array if it is half full — Quadratic

2. Queues (8 points)

In the *Josephus* problem from antiquity, **N** people are in dire straits and agree to the following strategy to reduce the population. They arrange themselves in a circle (at positions number 0 to N – 1) and proceed around the circle, eliminating every **M**-*th* person until only one person is left. Legend has it that Josephus figured out where to sit to avoid being eliminated.

Given the parameters **N** and **M**, provide the Java code or pseudocode of a Queue client that prints out the order in which the people are eliminated.

```
Queue<Integer> queue = new Queue<Integer>();
for(int i = 0; i < n; i++)
{
        queue.enqueue(i);
}
int k = 0;
while(k != n-1)
{
        for(int i = 1; i < m; i++)
        {
                queue.enqueue(queue.dequeue());
        }
        System.out.println(queue.dequeue());
        k++;
}
```

3. Short Answers (12 points)

Provide a short answer (5 lines or less) to each of the following questions. In each case, provide a brief justification for your answer.

1. Let **A** be an array of size **N** (≥ 2) containing integers from 1 to N-1 (inclusive), with exactly one repeated. Describe a fast algorithm (with **~N** array lookups of **A**) for finding the integer in **A** that is repeated.

   Initialize sumExpected = 0;
   Initalize sumActual = 0;
   Add values from 1 to N-1 and store it in sumExpected(eg: 1+2+3+...+N-1)
   Add values in the array A and store it in sumActual(eg: A[0]+A[1]+...+A[N-1])
   Compute sumActual - sumExpected and store result in repeatedValue(this is your answer)
   The array A is only gone through once, which corresponds to N lookups, to sum the values up.

2. Given a circularly linked list **L** containing an even number of nodes, describe how to split **L** into two circularly linked lists of half the size. Your solution should visit each node of the list only once. (*Note*: a circularly linked list is a singly linked list such that each node has a non-null next node.)

```
/*Initialize 2 pointer and point them to NULL*/
        *headList1 = NULL;  *headList2 = NULL;
/*Initialize 2 pointers and point them to the head of the list L*/
        *ptr1 = head;  *ptr2 = head;
/*Here the ptr1->next->next will become head.*/
while(ptr1->next != head && ptr1 -> next -> next != head)
{
        ptr1 = ptr1->next->next;
        ptr2 = ptr2->next;
}
/*As there are even number of nodes, we need to move ptr1.*/
if(ptr1->next->next == head)
        ptr1 = ptr1->next;
/*Setting the head pointer of the first half*/
headList1 = head;
/*Setting the head pointer of the second half*/
if(head->next != head)
        headList2 = ptr2->next;
/*Making the second half circular*/
ptr1->next = ptr2->next;
/*Making the first half circular*/
ptr2->next = head;
```

4. Analysis of Algorithms (14 points)

1. Consider the following code fragment:

```
int val = 0;
for (int i=1 ; i<=N ; ++i) {
    for (int j=1 ; j<i*i ; j*=2) {
        val++;
    }
}
```

   Suppose it takes 1 second to run this code when N = 1000. How long will it take approximately to run this code when N = 1000000 (10^6)? Provide a detailed answer. (Use Stirling's formula)

```
i          j
1          1
2          1 2
3          1 2 4 8
4          1 2 4 8
5          1 2 4 8 16
6          1 2 4 8 16 32
.             .
.             .
.             .
N          log(i*i)
= O(Nlog(N))
```

```
N=10^3 -> 1 second
N=10^6 ->
                (10^6)log(10^6) / (10^3)log(10^3)
                = (10^(6-3))* 6log(10)/ 3log(10)
                = 10^3 * 6/3
                = 10^3 * 2
                = 2000 seconds
```

2. Let **S** be a set of N lines in the plane such that no two lines are parallel and no three lines meet at the same point. Prove, by induction, that the lines in **S** determine **Θ**(N^2) intersection points. (Reminder: **Θ** / " Big Theta" designates an approximation of the asymptotic complexity to leading order without the constant term).

If N = 1 line, 0 intersections
if N = 2 lines, 1 intersection
if N = 3 lines, 3 intersections
if N= 4 lines, 6 intersections
if N = 5 lines, 10 intersections
.
.
.
if N = N lines, N(N-1)/2 intersections
As it has to be the previous number of intersections + the new number of intersections(n -1 intersections).

1/2 * (N^2 - N) intersections
=> θ(N^2) intersections

5. Application of Sorting (10 points)

Given two sets **A** and **B** represented as sorted sequences, give Java code or pseudocode of an efficient algorithm for computing **A** ⊕ **B**, which is the set of elements that are in **A** or **B**, but not in both. Explain why your method is correct.

Assuming, the sequences are Arrays(and we can use A.length) and the new sequence will not be bigger than 100 elements. The A and B are created just for example use.

```
int A[] = {1,2,3,3};;
int B[] = {1,2,4,6,7,8};
int C[] = new int [100];
int i = 0;
int j = 0;
int k = 0;
while(i < A.length && j < B.length)
{
        if(A[i] < B[j])
        {
            C[k] = A[i];
            i++;
            k++;
        }
        else if(A[i] > B[j])
        {
            C[k] = B[j];
            j++;
            k++;
        }
        else
        {
            i++;
            j++;
        }
}
```

```
while(i < A.length)
        {
            C[k] = A[i];
            k++;
            i++;
        }
        while(j < B.length)
        {
            C[k] = B[j];
            j++;
            k++;
        }
`       return C;
```

END