Exercise I – Recursion

1- Implement a recursive method *boolean isPalindrome(**String** s)* that checks whether s is a palindrome. A palindrome is a string that reads the same backwards as forwards
2- Implement a recursive method *boolean isPrime(**int** n)* that checks whether n is a prime number. A prime number is a number whose sole divisors are 1 and itself.
3- Write a recursive method called *printSquares(**int** n)* that accepts an integer parameter *n* and prints the first *n* squares separated by commas, with the odd squares in descending order followed by the even squares in ascending order. For example, *printSquares(8)* prints the following output:

49, 25, 9, 1, 4, 16, 36, 64

Exercise II – Recursion with Linked Lists

In this exercise, you can assume you are using an **int** Linked List. You can refer to the code for the integer linked list from the slides on moodle. Add the methods described below to the int LinkedList class implemented during the lecture:

1- Given a singly linked list, write a recursive function *int avgRec(**Node** n, **int** sum, **int** count)* to find the average of all nodes of the given Linked List
2- Given two singly linked lists, write a recursive function *boolean isIdentical(**Node** a, **Node** b)* that checks if two linked lists are identical or not.

Exercise III – Generic Classes

Generic methods are a very efficient way to handle multiple datatypes using a single method. This problem will test your knowledge on Java Generic methods.

Let's say you have an integer array and a string array. You must write a single method printArray that can print all the elements of both arrays. The method should be able to accept both integer arrays or string arrays.

Create a class called Printer that has an array of generic type T. Create a method print() that loops over the array and prints its elements. In the test driver make 3 instances of this class, int, string and double and call the method print.

Exercise IV – Algorithm Analysis

Part 1

Specify the running times of the following code snippets/operations in Big-Oh notation in terms of the input size n (make sure you briefly justify your answer):

1-

```
for(int i = 0; i < n; i++){
    sum += 1;
}
```

2-

```
for(int i = 0; i < n; i++){
    for(int j = 0; j < n; j++){
        sum += 1;
    }
}
```

3-

```
for(int i = 0; i < n; i*=2){
    sum += 1;
}
```

4-

```
for (int i = 0; i < n-2; i++) {
    for (int j = i+1; j < n-1; j++) {
        for (int k = j+1; k < n; k++) {
            sum += 1;
        }
    }
}
```

5-

```
for (int i = 0; i < n-2; i++) {
    for (int j = i+1; j < n-1; j*=2) {
        for (int k = j+1; k < n; k*=2) {
            sum += 1;
        }
    }
}
```

The following parts assume that you are not allowed to use auxiliary storage.

7- Adding a node to the head of a linked list (assuming a singly linked list implementation)

7- Deleting a node with a specific data value from a linked list (assuming a singly linked list implementation)

8- Deleting nodes with duplicate data from a linked list (assuming a singly linked list implementation) by keeping the closest node to the head with data d for all d present in the array.

9- Deleting the last n/2 elements from a linked list, n being the size of the linked list (assuming a singly linked list implementation)

Appendix to Part 1:

a) What are the best case, average case, and worst case running times of the add(Object) ArrayList method? Explain.

b) What are the best case, average case, and worst case running times of the remove(int) ArrayList method? Explain.

Part 2:

Specify the lowest Big-Oh complexity for the following functions:

a) $0.01n2 + 500n$

b) $5 + \log2(n100)$

c) $5n3 + \log2(nn4)$

Part 3:

Prove your answers from Part 2 (specify the values of c and n0 that make your answer work). In addition, prove that the function in a) is $\theta(g(n))$, where $g(n)$ is the answer you obtained in Part 2, a).