Page 26-27

1.1 - Write a program to solve the selection problem. Let k=N/2. Draw a table showing the running time of your program for various values of N (10, 100, 1000, 10000). Make sure you submit a working java program and the runtime in seconds to 5 decimal places for at least the 4 problem sizes above.

1.5 - Write a recursive method that returns the number of 1's in the binary representation of N. Use the fact that this is equal to the number of 1's in the representation of N/2, plus 1, if N is odd. For example, starting at 5, 5 is odd, so ones++, 5/2 = 2, 2 is even, 2/2 = 1, 1 is odd, so ones++, so 5 should produce an answer of 2 (101 in binary). DO NOT use the built in method for converting an integer into a binary string. Make sure you are prompting the user for a number, then outputting how many 1's are in it's binary representation.

1.15 - Define a Rectangle class that provides getLength and getWidth methods. Using the findMax routines in Figure 1.18, write a main that creates an array of Rectangle (5+) and finds the largest Rectangle, first on the basis of area, and then on the basis of perimeter. Note, you should create 2 Comparator classes, one for area, and one for perimeter, to use with the findMax method provided by the textbook. You shouldn't need to modify the findMax method.

Grading:

1.1 - Program (2pts)  
- Select middle largest number (if 10 numbers, get 5th largest)

1.1 - Runtimes (1pt)  
- Add code to program for timing (only need to time solving, not filling the array with numbers in a random order), record runtimes of at least 4 different executions (10, 100, 1000, 10000)

1.5 - Program (3pts)  
- Recursive method

1.15 - Program (4pts)  
- Rectangle Class, 2 Comparators, findMax, 5+ Rectangle objects in an array

Page 50-55

2.1 - Order the following functions by growth rate. Make note of the functions that are considered equal in Big-Oh

2.2 - Which of the following are true (a-d). Explain answers.

2.6 - Word problem  
a: what would the fine be on Day N - write this as a function of N  
b: How many days would it take for the fine to reach D dollars (Big-Oh representation)

2.7 (a,b only)  
a: Give Big-Oh runtime of programs 1-6  
b: Implement programs 1-6, calculate run times in seconds for each using the values of N (10,100,1000,10000)

2.8 (b,c only)  
b: Give as accurate as you can, the Big-Oh expected value for all 3 algorithms  
c: Write 3 separate programs (1 for each algorithm), calculate the run time in seconds of each for the 10 values of N provided

2.11 - Calculate run times based on given program run time growth rates (a-d)

2.12 - Similar to 2.11, figure out how large the input can be for the program to finish in 1 min (a-d)

2.13 - Give number of operations for performing an exponent by just multiplying the number N number of times(a) versus the method described in 2.4.4(b) (pg 47-49)

2.19 - Modify Algorithm 2 (pg 41) and Algorithm 4 (pg 44) to return an object that contains 3 values, the sum, the start index, the end index for the maximum sub-sequence sum (you can use an array, or a custom object)

2.27 - Provide a program that asks a user for the value N, creates an N x N matrix of unique values that follow the rules for the problem, then prompt the user for a number, and determine if the number exists in the matrix in O(N) run time or better.

2.31 - In the binary search algorithm, if we replaced (low = mid) with (low = mid + 1), would the program still work? Explain why or why not.

Grading:

2.1 - 1pt

2.2 - 1pt

2.6 - 1pt

2.7a - 1pt

2.7b - Programs (2pts)

2.7b - Run times (1pt)

2.8b - 1pt

2.8c - Programs (3pts)

2.8c - Run times (1pt)

2.11 - 0.5pt

2.12 - 0.5pt

2.13 - 1pt

2.19 - Programs (2pts)

2.27 - Program creates NxN matrix of unique values in correct format (1pt)

2.27 - Program determines if X is located in the matrix in O(N) or better (2pts)

2.31 - 1pt

Page 96-100

3.2 - Add a swap method to the two linked list classes we wrote in class. Make sure you are only changing the pointers of the nodes, do not touch the actual values. Make sure to account for all special swap cases.

3.6 - Write a program for the Josephus problem (similar to hot potato), calculate the Big-Oh of your solution. Prompt the user for the value of N.

3.7, 3.8 - Calculate Big-Oh notation for these two methods. Be sure to take into account the run time of the methods being used in the loops. Show your work.

3.20 - Lazy deletion (mark items deleted instead of actually removing them), a: pro/con of lazy deletion, b: implement lazy deletion on the doubly linked list class we wrote in class (make sure the remove() method performs lazy deletion, and that all other methods honor the deleted nodes correctly, include the process for actually deleting items after half the items in the structure are considered deleted)

3.22,3.23 - Write a program that asks the user for 2 inputs, first the keyword infix/postfix, then a formula in that given format  
infix - (1+(2-3))  
postfix - 123-+  
Eval - 0  
Use the MyStack class we wrote in class for evaluating the postfix (it will also be helpful for converting the formats)  
Valid symbols: 0-9, (, ), +, -, \*, /

MyQueue - Create your own MyQueue class, be sure to use generic data types like we did with MyStack, store first/last node, make sure all methods execute in constant time, create a test class to show the Queue works correctly

Grading

3.2 - Programs (2pts)

3.6 - Program (1pt)

3.6 - Big-Oh (0.5pt)

3.7 - Big-Oh (0.5pt)

3.8 - Big-Oh (0.5pt)

3.20 - a: Pro/Con (1pt)

3.20 - b: Program (3pts)

3.22,3.23 - Convert infix->postifx (2pts)

3.22,3.23 - Convert postfix->infix (2pts)

3.22,3.23 - Evaluate postifx for answer (1pt)

MyQueue - 1.5pts

Page 161-166

4.2 - a-e: Fully describe every element in the tree (parent, children, siblings, depth, height)

4.9 - a: show the tree after each insert, b: show the tree after the delete root completes

BinaryTreeFormula - Expand on the binary tree example we did in class that reads infix formulas and stores them in a binary tree. Create methods for reading postfix and prefix formulas and storing them in the tree. DO NOT convert prefix/postfix to infix before storing in the tree. Make the methods as efficient as possible. Include a tester class and make sure the tree is printed in all 3 formats after being populated from any of the three. Include Big-Oh notation for all 3 populate methods, and all 3 print methods.

4.16 - Lazy Deletion on the binary search tree class, make sure to update all methods that are affected by this change

4.19 - Show the AVL tree after each insert

4.41 - Modify a binary tree class we created, add a method to print all the nodes in level order (root, all nodes at depth 1[root's children], all nodes at depth 2, etc) Make sure the method runs in O(N), and show that it does with comments about runtime at each step in your method.

4.46 - Modify a binary tree class we created, add a method to see if two binary trees are visually identical (if drawn on paper, would have the same structure, values don't matter).

TreeMap - Practice using Java TreeMap class. Create a program that asks the user for names until they chose to quit, insert the names as keys in the TreeMap, and the number of vowels as the value in the TreeMap. After the user choses to quit inserting names, Traverse the map and print all key/value pairs.

Grading:

4.2 - 2pts

4.9 - 2pts

BinaryTreeFormula - Populate methods (6pts)

BinaryTreeFormula - Big-Oh x 6 (3pts)

4.16 - 2pts

4.19 - 0.5pt

4.41 - 2pt

4.46 - 2pt

TreeMap - 0.5pt

Page 218-222

5.1 - a-d, show what each hash table looks like after the values are inserted

5.2 - Show the tables above after being rehashed (all 4)

5.8 - Cubic probing pro/cons versus quadratic probing, fully explain.

5.20 - Implement a generic Map class using the structure provided in the book. The QuadraticProbingHashTable is available on D2L, you should not modify this class.  
NOTE: Make sure to provide equals and hashCode methods in the private class to get this working properly

5.24 - Implement a hopscotch hash table. Do not compare with other methods. Default the maximum hops to 4, give a constructor that allows this value to be changed.

Grading:

5.1 - 2pts

5.2 - 2pts

5.8 - 1pt

5.20 - 5pts

5.24 - 5pts