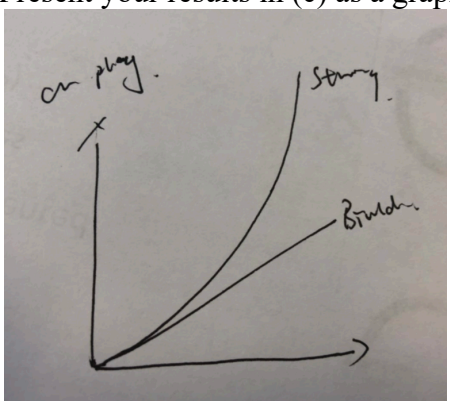
Data Structures and Algorithms
INFO 6205
Homework 3
Due: February 2, 2019
Hongxi li

Put all your java, compiled class files and documentation files into a zip file named Homework3.zip and submit it via the drop box on the blackboard before the END of due date. Put your name on all .java files. There will be a short quiz on this homework.

1. Write a Java program that generates random text string with a length of 500 bytes for 200,000 iterations. For each iteration, reverse the string using:
a) String operations, and
b) StringBuilder operation(s). Then
c) What is the running time complexity of (a) and (b) after all iterations?
time complexity of (a) is O(n^2), time complexity of (b) is O(n).
d) Present your results in (c) as a graph showing the running times.



2. Consider String "Test is a hard test".
a) Generate a binary Huffman Tree
b) Show binary data both before and after compression. Analyze difference.
Before: T(1010100) e(1100101) s(1110011) t(1110100) i(1101001) s(1110011) a(1100001) h(1101000) a(1100001) r(1110010) d(1100100) t(1110100) e(1100101) s(1110011) t(1110100) " "(1000000)
1010100 1100101 1110011 1110100 1000000 1101001 1110011 1000000 1100001 1000000 1101000 1100001 1110010 1100100 1000000 1110100 1100101 1110011 1110100
After: t(010) s(011) e(100) a(101) i(1100) h(1101) r(1110) d(1111) " "(000)
010 100 011 010 000 1100 011 000 101 000 1101 101 1110 1111 000 010 100 011 010
Different(Huffman Code Func):

- The leaf nodes of the tree contain frequency and characters.
- The frequency is higher near the root node, and the lower frequency is at the bottom of the tree.
- The root node frequency value is equal to the number of characters in the input.
- The tree represents more compression than other trees and is an optimal prefix code.

c) Consider Java code: https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/
Write Pseudo-Code for the Huffman algorithm

d) Compile Java code and run it with the input string provided above.
Test is a hard test
Huffman Code: 00100001101110110101100110001101111001011000110111100011011 1
BitStream:
00010110010101010101001011001001001000000010110000101011010010101110010101101 0
000101110011101110100
Huffman Code length(bit): 60
Length of text(bit): 152
Compression ratio: 39.473684210526315%

3. Consider signed byte X, and unsigned byte Y. What are the possible values for both X and Y can have?

Answer:

A signed byte, its value range from-128 to 127. An UnsignedByte is like a Byte, but its values range from 0 to 255 instead of -128 to 127. Because byte stores signed values (both positive and negative). If you count from -128 to +127, it is 256 values. Java provides only signed byte values.Also, it is 2^8 because a byte consist of 8 bits. Each bit can be either '0' or a '1', hence two possible values per bit (2^8).

1 bit -> 2^1 = 2 values
2 bits -> 2^2 = 4 values
8 bits -> 2^8 = 256 values
n bits -> 2^n.

4. Write Java Factorial program for n! where n=7 and n=14.
   a) Compile the code and run.
   b) Step through each recursive call and show the Stack for push and pop. For each pop operation, show the STATE of push operation (ie: how "n" is being kept track of).
   c) What is the Time and Space Complexity of Factorial function?

n=7; do Factorial(7-1); 720*7=5040
n=6; do Factorial(6-1); 120*6=720
n=5; do Factorial(5-1); 24*5=120
n=4; do Factorial(4-1); 6*4=24
n=3; do Factorial(3-1); 2*3=6
n=2; do Factorial(2-1); 1*2=2
n=1; do Factorial(1); 1*1=1
The Time and Space Complexity are O(n).

5. Java is Pass-by-Value, what does that mean?
The method parameter values are copied to another variable and then the copied object is passed, that's why it's called pass by value.

Consider the following two programs,

Program-1:

```
public static void main(String[] args) {
   Dog aDog = new Dog(“Bella");
   Dog oldDog = aDog;

   changeName(aDog);

   aDog.getName().equals("Bella");       True
   aDog.getName().equals("Molly");       False
   aDog == oldDog;                       True
}

public static void changeName(Dog d) {
   d.getName().equals("Bella");          True
   d = new Dog("Molly");
   d.getName().equals("Molly");
}
```

Program-2:

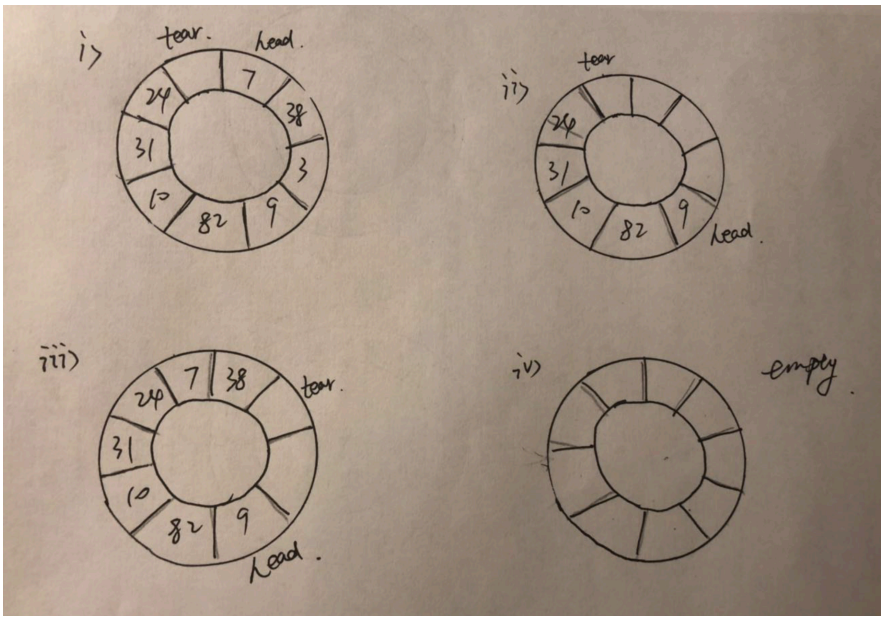
```
public static void main(String[] args) {
   Dog aDog = new Dog("Bella");
   Dog oldDog = aDog;

    changeName(aDog);

   aDog.getName().equals("Molly");       True
   aDog == oldDog;                       True
}

public static void changeName(Dog d) {
  d.getName().equals(“Bella");           True
  d.setName("Molly");
}
```

6. Consider the following, Input Data: {7, 38, 3, 9, 82, 10, 31, 24}
   a) Graphically build a Circular queue for input data. Discuss and show the Head and Tail pointers at each step.
      i) enqueue all input data
      ii) dequeue three elements
      iii) enqueue two elements
      iv) dequeue all elements



b) Write Java code for the Circular queue, provide enqueue, dequeue, isEmpty, isFull, and displayQueue methods, to show the status of the queue with steps described in (a). Compile code and Run with input data.