# CSC 205 Lab 5 : Bitsets, String Buffers, & Run Time Analysis

*Due by Friday, September 29th, 2:00 PM*

## Goals

After completing this lab, you should be able to:

* Use objects of the BitSet and StringBuffer types where appropriate
* Use different sorting and searching algorithms, and be able to predict their run times on different input sets

## Lab Setup

Change into your Labs directory, and let's create and change into a Lab5 directory.

Now, let's copy over some files by typing : cp /pub/digh/CSC205/Lab5/\* .

**Bitsets**

Take a look at program Sets in your directory. Now, let’s determine what the two BitSet objects, set1 and set2, will look like. The variable BOUND is a global integer constant that is equal to 8. Write the values of these two BitSet objects below.

BitSet set1 = new BitSet(BOUND);

BitSet set2 = new BitSet(BOUND);

for (int i = 1; i < set1.size(); i \*= 2)

set1.set(i);

for (int i = set2.size()-1; i > 0; i /= 2)

set2.set(i);

Now, let’s write a void class method print that can be used to print out our two bitsets so that we can check our answers from above. Print the digit 1 for each set bit, and the digit 0 for each unset bit. Compile and test your program.

Thirdly, let’s write a value-returning class method inverse that takes a BitSet object and returns its inverse. The inverse is merely the “flipping” of all set bits to 0, and the turning on of all unset bits. For example, the inverse of 11111110 is 00000001.

Uncomment out the lines in your main method that call inverse on your two bitsets. You should have the following printed to the screen.

inverse of set1 = 10010111

inverse of set2 = 10101110

###### **Sorting and Searching**

Suppose you want to use the selection sort algorithm to sort a list of integers in ascending order. Show what the array list declared below would look like after each pass through the outer loop of the selection sort. There will be a total of four passes.

int[] list = {8, 15, 12, 34, 1};

Compile and execute the file Selection.java which will give you a printout of your array list after each pass through the outer loop.

Now, given the declaration of object list above, how many checks would you need to make using the binary search algorithm for the integer 15? Remember this algorithm requires that the array is sorted first. Show all values for first, last, and middle in the space below. Remember the formula for calculating middle is

(first + last) / 2.

Compile and execute file Binary.java to check your answers.

## Run Time Analysis

First off, we want to write a program MyRand which will create two data sets of random integers of sizes 10,000 and 40,000. Display this program by typing cat myRand.java

Notice the for loop that will run *size* times where *size* is an integer typed at the command line upon execution. In the body of that loop, be sure you understand the one line of code which will print a random integer between 1 and 500 to the screen for each loop iteration. Our sequence is inclusive (i.e., includes the 500).

Compile this program, and run it using the command :

java MyRand 10000 > data1

This instructs the operating system to send all output to the file data1 rather than the screen. Check the file data1 and make sure it contains 10,000 random integers in your desired range.

Now, re-run your program using a command that will give us a file of 40,000 integers :

java MyRand 40000 > data2

Now, let's determine the actual amount of system run-time that it takes to sort the data from both of our files using the bubble sort algorithm (an algorithm). To do this, we need to use the *Linux* time command. This command will return the time in seconds that it takes the CPU to complete your program.

The program Bubble is an executable program which reads in integers from the keyboard, and sorts them using the bubble sort algorithm. This program is set up to take the desired array size you plan on sorting from the command line. So, here is the command we need to type to calculate the run time of a bubble sort on an array of size 10,000 is :

time java Bubble 10000 < data1

The total time it takes this to run will be the first float value on the first line prior to ``user." It should be around two seconds to sort 10,000 integers using a bubble sort.

Now, before we run our program on our array of size 40,000, let's see if we can predict our run-time. Remember to predict the run-time, you simply need to look at the factor of input increase, plug this increase into your growth rate (ours is  with the bubble sort), and then multiply this value by your prior run-time. Fill in your predicted value for 40,000 in the table.

Repeat the timing process now on your array of size 40,000. Record your result in your table. How close were your predicted and actual values?

Finally, delete your two data files, data1 and data2. They’re both very large files that we’ll no longer need, and can easily recreate if we do.