Computer Science II Section 1 Exam 1 Date: 2/14/2018

Last Name: _____, First Name: _____

1) (15 pts) In class we've looked at some sorts. To verify that our code has worked, we've written a utility function, isSorted, shown below:

```
public static boolean isSorted(int[] arr) {
     for (int i=0; i<arr.length-1; i++)
          if (arr[i] > arr[i+1])
               return false;
     return true;
}
```

In our particular use case, unless our sort was incorrect, the run time of this method ends up being O(n), when run in an array of length n because the for loop runs to completion when the input array is sorted.

Now consider the case of running this method on a randomly arranged array of n unique items, where each of the possible n! permutations is equally likely.

In this question, we'll set up the summation equal to the ***average case*** run-time of the method under these particular assumptions.

(a - 2 pts) What is the probability that the first two items are in order? What is the probability that the first two items are out of order? (Note that in this case, the method does one single comparison before returning.)

(b - 3 pts) What is the probability that the second item is larger than the first, but the third item is smaller than the second? (This corresponds to the situation where the method does 2 comparisons before returning a result in the negative.)

(c - 6 pts) In general, what is the probability that the first k-1 items are in sorted order but the k^{th} item is less than the $(k-1)^{th}$ item? (Hint: Consider the possible k! permutations of any of the first k chosen items and how many of them fit this mold, then divide that value by k!.)

(d - 4 pts) Using the information in part (c), set up a summation equal to the average case run-time of the isSorted method when run on a random array where we expect each permutation of items to be equally likely. You may leave one term (corresponding to sorted input) out of the sum.

2) (20 pts) Consider arranging 16 values into a 4 x 4 square such that no two adjacent values (directly up, down, left or right) differ by more than the absolute value of a given maximum. If the input values are spaced out enough and the maximum is small enough, we can utilize backtracking to speed up a solution to count the number of different ways to arrange the values in the square. (In particular, we skip trying a value in a square if its difference with the neighbor above or to the left the square is greater than the given maximum.) Below is an incomplete implementation of 2 methods of a program that solves this problem. Complete the 2 methods.

```
final public static int N = 4;
final public static int[] DX = {-1,0};
final public static int[] DY = {0,-1};
public static int go(int[][] grid, boolean[] used, int[] list, int k, int
max) {

      if (k == N*N) return ____;

      int res = 0;
      for (int i=0; i<N*N; i++) {

            if (used[i]) continue;

            if ( ____________________________________________________ ) continue;

            grid[ ______ ][ ______ ] = list[i];

            used[i] = ________ ;

            ____________________________________________________________ ;

            used[i] = _________ ;
      }

      return res;
}

public static boolean conflict(int[][] grid, int pos, int value, int max) {

      int cX = pos/N;
      int cY = pos%N;

      for (int i=0; i<DX.length; i++) {

            if (!inbounds( _______________ , _____________ )) continue;

            if (Math.abs( ____________________________________________ ) > max)
                  return true;
      }
      return false;
}

public static boolean inbounds(int x, int y) {
      return x >= 0 && x < N && y >= 0 && y < N;
}
```

3) (12 pts) Complete the implementation of bucket sort shown below. In this implementation we assume that the values range from 0 to Integer.MAX_VALUE, inclusive when forming our buckets. To be safe, we add 1 to the size of the range obtained by integer division.

```
public static void bSort(int[] arr) {

      int range = Integer.MAX_VALUE/arr.length + 1;

      ArrayList[] buckets = new ArrayList[arr.length];
      for (int i=0; i<buckets.length; i++)
            buckets[i] = new ArrayList<Integer>(5);

      for (int i=0; i<arr.length; i++)

            insert( _______________________________ , arr[i]);

      int idx = 0;
      for (int i=0; i < _____________________; i++)

            for (int j=0; j < ____________________________________; j++)

                  arr[________] = ((ArrayList<Integer>)buckets[__]).get(__);
}

public static void insert(ArrayList<Integer> list, int item) {
      int i = 0;
      while (i < list.size() && list.get(i) < item) i++;
      list.add(i, item);
}
```

4) (4 pts) Answer the following questions about the implementation above:

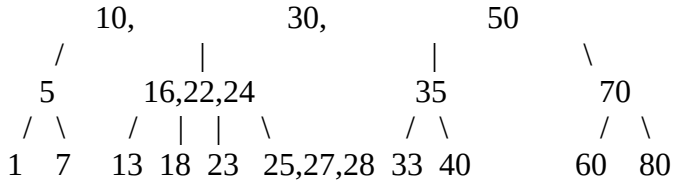
(a) Why might the ArrayList constructor call in the first for loop set the initial size of each list to 5 instead of opting for the default of 10? (Note: This is only relevant for sorting large arrays.)

(b) Why is the insert method somewhat less efficient (roughly be a factor of 2, on average) than it could be?

5) (9 pts) Show each phase of radix sort on the following integers where we sort by digit:

Original List	Phase 1	Phase 2	Phase 3	Sorted
2987				1287
2381				1391
3967				2381
1391				2567
2567				2987
3394				3394
1287				3967

6) (12 pts) Show the result of inserting 26 into the 2-4 Tree below. Note: Whenever a node overflows, please "send up" the ***third*** value out of four in the overflowed node.



7) (10 pts) In the MCSS analysis, we moved from an $O(n^3)$ solution to an $O(n^2)$ solution by realizing that the sum of all the values from index i to index j+1 in an array was simply equal to the sum of the values in the array from index i to index j, plus the value stored in index j+1. Thus, if we previously computed the sum from index i to index j, to get the sum from index i to index j+1, we just have to add 1 more value to the old sum. A similar idea can be used to solve the following problem: Given an array of size n, find the maximal sum of any k consecutive elements. The nieve solution tries adding all the values in all n-k+1 consecutive intervals of size k, yielding a run-time of O(k(n-k)). (Note we drop the +1 from the Big-Oh since it isn't a significant portion of the total function.) We can improve this run-time to O(n) by noting that if we have the sum of a single window of k values, we can get the sum of the window one space to its right by adding the new value into the old sum AND subtracting out the left-most value of the old window. Consider the array below where we know the sum of the values in indexes 2 through 8 is 31. To obtain the sum from indexex 3 to 9, we take 31, add 1 (value in index 9) and subtract 2 (value in index 2) to get 31 + 1 - 2 = 30. Indeed, 5+4+7+9+3+1+1 = 30.

index	0	1	2	3	4	5	6	7	8	9	10	11
value	3	6	2	5	4	7	9	3	1	1	6	9

Complete the method shown below that achieves this O(n) implementation (for an array of size n) for solving this problem. Assume that all possible sums of consecutive elements fits in an int.

```
// Returns the maximal sum of any k consecutive items in vals.
// We assume vals.length >= k.
public static int maxSum(int[] vals, int k) {

    int cur = 0;
    for (int i=0; i < ____ ; i++)
        cur += vals[i];
    int max = cur;

    for (int i= ______; i<vals.length; i++) {

        cur = ____________________________________________________ ;
        max = Math.max(cur, max);
    }

    return max;
}
```

8) (15 pts) Show the steps of running Prim's on the graph whose adjacency matrix (tedundant edges aren't listed) is shown below, **starting at vertex C**. **(Note: if you don't start at vertex C you'll get very little credit.)** In particular, at each step, show which edges have been added into the priority queue for consideration. Then, show the order of which edges are considered. If an edge isn't added, list what cycle it would have caused.

	A	B	C	D	E	F
A		20	13	5	7	4
B			18	19	1	11
C				12	15	9
D					14	6
E						16

Edges added before main loop: _____

Edge Selected	Was it Added?	If yes, what edges added to PQ? If no, what cycle?

Note: You may not fill out all rows on this chart before the algorithm terminates, so it's okay if you leave some rows blank.

9) (3 pts) The Fox Broadcasting Company shares its name with which animal? _____

Scratch Page – Please clearly mark any work on this page you would like graded.