**Recitation Worksheet: Algorithm Analysis, Lower Bound Sorting, Counting/Radix Sorts Solutions**

1) 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) 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.)

**50% or 1/2. For each permutation starting with a, b where a < b, there is a corresponding permutation starting with b, a. Thus, half of all permutations have the first two items out of order.**

(b) 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.)

**Given 3 items, a, b, c with a < b < c, there are 2 orderings of these three items with the first less than the second and the third less than the second: a, c, b and b, c, a. Since there are 6 orderings total of a, b and c, it follows that the desired probability is 2/6 = 1/3.**

(c) 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!.)

**Given k items, with $a_1 < a_2 < a_3 \ldots < a_k$, we want to count the number of permutations of these items with the largest term appearing second to last and all the terms before the largest appearing in ascending order. In all of these situations, the algorithm will make k-1 comparisons before returning false.**

**In essence, we are forced to place the largest item in a particular slot. Then, we are free to choose any of the remaining k-1 items to the right of the largest item. So, there are k-1 ways to do this. Once we pick that item, the following k-2 items are forced into place, since they have to be in order. It follows that the corresponding probability is $\frac{k-1}{k!}$. Note that for k = 3, we do indeed get the consistent answer of $\frac{3-1}{3!} = \frac{2}{6} = \frac{1}{3}$.**

(d) 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.

**The value of k (where we find the incorrect pair) can range from k = 2 to k = n. To this, we just need to add the probability of the sorted list (1/n!) and multiply this by the number of comparisons the algorithm does in this case. The full summation is:**

$$\left(\sum_{k=2}^{n}(k-1) \times \frac{(k-1)}{k!}\right) + \frac{n-1}{n!}$$

**We can simplify this a bit as follows:**

$$\left(\sum_{k=2}^{n}\frac{(k-1)^2}{k!}\right) + \frac{n-1}{n!}$$

2) One way to analyze this is to write a simulation, run it many times and take the average. The latter portion of this is fairly trivial, so for this question, you will simply write a single Java method that takes in n, the number of dice, and k, the number of sides on each dice, simulates this process, and returns the number of turns it took to complete a single simulation of the game. Fill in the method signature given below. Assume that you have access to a static class variable r, that is of type Random, which you can use to generate random integers.

```
public static Random r;

public static int numTurnsSim(int n, int k) {

     int turns = 0;
     while (n > 0) {

          int numK = 0;

          for (int i=0; i<n; i++) {
               int die = r.nextInt(k);
               if (die == k-1) numK++;
          }

          turns++;
          n -= numK;
     }

     return turns;
}
```

3) Let the dice have k sides each and let T(n) equal the expected number of turns to complete the simulation described previous. A recurrence relation that T(n) satisfies is as follows:

$$T(0) = 0$$

$$T(n) = 1 + \sum_{i=0}^{n} [\binom{n}{i} \left(\frac{1}{k}\right)^{i} \left(\frac{k-1}{k}\right)^{n-i} T(n-i)]$$

In words, explain why this formula is correct. In your explanation, please explain what 1 represents, what the summation index i, represents and what T(n-i) represents. (Of course, explain what each part represents and why their interaction is the way that it is.) Note: One issue with this formula is that a T(n) term appears on the right hand side. But, if one were to solve this recurrence, we can easily take care of this problem by subtracting that term to the left hand side and factoring out T(n). Then, we would have a factor by which we could divide both sides of the resulting equation.

**The 1 represents a turn throwing the the n dice. Of these dice, any where from 0 to n of them could turn up to land showing k. The variable i in the summation index represents the number of dice that show k out of the n dice that are rolled. For each of these possibilities, we must calculate the probability of that happening and multiply that by the expected number of turns of the simulation which will now have n - i dice left. The probability that i dice show up with k is based on the binomial distribution. In general, if we repeat a trial n times where the probability of success is p, the probability of getting exactly i success is $\binom{n}{i} (p)^{i}(1-p)^{n-i}$. For this particular problem, the probability of success, ie. rolling a k, is $\frac{1}{k}$. This explains the first three terms in the sum; these are just the product that represents the probability that i of the dice will show k. We must multiply this probability by the expected number of future turns, but this is just T(n - i), since T(x) represents the expected number of turns in the game starting with x die for any non-negative integer x. If i of the dice show k, then that means we continue to play the game with n - i dice, and using the definition of expectation, it follows that the appropriate term to multiply the aforementioned probability by is just T(n - i). This explains the recurrence relation above in full.**

4) You are given a composite number n, but are asked to give proof that it is composite. You have a test that you can run such that, given a composite number, it proves that it is composite 50% of the time. Thus, to achieve your goal, you'll simply keep on repeating your test until you obtain the proof that the given number is composite. Fill out the chart below indicating the probability your algorithm will run the test various number of times.

| Number of Times Test is Run | Probability |
|---|---|
| 1 | ½ |
| 2 | **¼** |
| 3 | **1/8** |
| 4 | **1/16** |
| k | **$1/2^k$** |

Using the chart above, write an infinite summation, in summation notation equaling the expect number of times your test will have to be run to prove that a given composite number n is composite.

$$\sum_{i=1}^{\infty}[i\left(\frac{1}{2}\right)^{i}]$$

Solve the summation on the following page. You should get a constant value as your answer. (Use the scratch page if you need more room to complete the summation.) Also, no need to use summation notation here. Feel free to write out the pattern of the sum so that it's easier to see your work.

$$S = 1(.5) + 2(.5)^2 + 3(.5)^3 + \cdots$$

Multiply through the equation by .5:

$$S(.5) = 1(.5)^2 + 2(.5)^3 + \cdots$$

Now, subtract this second equation form the first:

$$S - .5S = .5 + (.5)^2 + (.5)^3 + \cdots$$

The right hand side is an infinite geometric sequence. Solve it and then solve for S:

$$.5S = \frac{.5}{1-.5} = 1, so\ S = 2.$$

5) What is the fewest number of comparisons necessary to sort an array of size 9? You may use the fact that 9! = 362880 and that $2^{16} = 65536$.

**We need to solve for the minimum value of k such that**

**$2^k > 9!$**
**$2^k > 362880$**

**We can list powers of 2 as follows:**

**$2^{16} = 65536.$**
**$2^{17} = 131072$**
**$2^{18} = 262144$**
**$2^{19} = 524288$**

**It follows that the minimal value of k we seek is 19.**

**6)** 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 | **2381** | **3967** | **1287** | 1287 |
| 2381 | **1391** | **2567** | **2381** | 1391 |
| 3967 | **3394** | **2381** | **1391** | 2381 |
| 1391 | **2987** | **2987** | **3394** | 2567 |
| 2567 | **3967** | **1287** | **2567** | 2987 |
| 3394 | **2567** | **1391** | **3967** | 3394 |
| 1287 | **1287** | **3394** | **2987** | 3967 |