ALGORITHM ANALYSIS

Problem 1. Order the following functions by growth rate: N, $\sqrt{N}$, $N^{1.5}$, N^2, $N \log N$, $N \log \log N$, $N \log^2 N$, $N \log(N^2)$, $2/N$, 2^N, $2^{N/2}$, 37, $N^2 \log N$, N^3. Indicate which functions grow at the same rate.

Problem 2. For each of the following six program fragments:

a. Give an analysis of the running time (Big-Oh will do)
b. Implement the code in the language of your choice, and give the running time for several values of N.
c. Compare your analysis with the actual running times.

```
(1)  sum = 0;
     for( i = 0; i < n; ++i )
        ++sum;
(2)  sum = 0;
     for( i = 0; i < n; ++i )
        for( j = 0; j < n; ++j )
           ++sum;
(3)  sum = 0;
     for( i = 0; i < n; ++i )
        for( j = 0; j < n * n; ++j )
           ++sum;
(4)  sum = 0;
     for( i = 0; i < n; ++i )
        for( j = 0; j < i; ++j )
           ++sum;
(5)  sum = 0;
     for( i = 0; i < n; ++i )
        for( j = 0; j < i * i; ++j )
           for( k = 0; k < j; ++k )
              ++sum;
(6)  sum = 0;
     for( i = 1; i < n; ++i )
        for( j = 1; j < i * i; ++j )
           if( j % i == 0 ) for( k = 0; k < j; ++k )
              ++sum;
```

Problem 3. Suppose you need to generate a *random* permutation of the first N integers. For example, $\{4, 3, 1, 5, 2\}$ and $\{3, 1, 4, 2, 5\}$ are legal permutations, but $\{5, 4, 1, 2, 1\}$ is not, because one number (1) is duplicated and another (3) is missing. This routine is often used in simulation of algorithms. We assume the existence of a random number generator, `r`, with method `randInt(i,j)`, that generates integers between `i` and `j` with equal probability. Here are three algorithms:

1. Fill the array a from `a[0]` to `a[N-1]` as follows: To fill `a[i]`, generate random numbers until you get one that is not already in `a[0]`, `a[1]`,`...`, `a[i-1]`.
2. Same as algorithm `(1)`, but keep an extra array called the `used` array. When a random number, `ran`, is first put in the array a, set `used[ran] = true`. This means that when filling `a[i]` with a random number, you can test in one step to see whether the random number has been used, instead of the (possibly) `i` steps in the first algorithm.
3. Fill the array such that `a[i] = i+1`. Then

```
for( i = 1; i < n; ++i )
   swap( a[ i ], a[ randInt( 0, i ) ] );
```

   a. Prove that all three algorithms generate only legal permutations and that all permutations are equally likely.
   b. Give as accurate (Big-Oh) an analysis as you can of the *expected* running time of each algorithm.
   c. Write (separate) programs to execute each algorithm 10 times, to get a good average. Run program `(1)` for `N = 250, 500, 1,000, 2,000`; program `(2)` for `N = 25,000, 50,000, 100,000, 200,000, 400,000, 800,000`; and program `(3)` for `N = 100,000, 200,000, 400,000, 800,000, 1,600,000, 3,200,000, 6,400,000`.
   d. Compare your analysis with the actual running times.
   e. What is the worst-case running time of each algorithm?

Problem 4. Consider the following algorithm (known as *Horner's rule*) to evaluate $f(x) = \sum_{i=0}^{N} a_i x^i$:

```
poly = 0;
for( i = n; i >= 0; --i )
   poly = x * poly + a[i];
```

   a. Show how the steps are performed by this algorithm for $x = 3, f(x) = 4x^4 + 8x^3 + x + 2$.
   b. Explain why this algorithm works.
   c. What is the running time of this algorithm?

Problem 5.

   a. Write a program to determine if a positive integer, N, is prime.
   b. In terms of N,what is the worst-case running time of your program? (You should be able to do this in $O(\sqrt{N})$.)
   c. Let B equal the number of bits in the binary representation of N. What is the value of B?

d. In terms of B, what is the worst-case running time of your program?
e. Compare the running times to determine if a 20-bit number and a 40-bit number are prime.
f. Is it more reasonable to give the running time in terms of N or B? Why?