Assignment 2a. **Running time and application of sorting, second chance.**

Given February 28, due March 7 (Wednesday).

1. We have a list of $n$ numbers, $a(k)$. We want to determine whether there are any three that add up to $L$ (a given number). Duplicates are allowed (e.g. $a(4) + a(4) + a(35)$).

   a. Write an algorithm that generates (but does not store!) all possible sums and checks whether the sum is equal to $L$. Be careful that your algorithm does not test unnecessary repeats. For example, once you know that $a(2) + a(4) + a(5) \neq L$ you also know that $a(4) + a(2) + a(5) \neq L$. Eliminating duplicates reduces the running time by roughly a factor of 6. What is the asymptotic running time of your algorithm?

   b. Write a faster algorithm that sorts $a$ first. What is the running time of this one. It should be roughly, but possibly not exactly, a power of $n$ faster.

2. Write an algorithm to generate all possible permutations of $n$ numbers. Use this as part of an algorithm to solve the *traveling salesman* problem. In this problem, we have an $n \times n$ table of costs, one for each pair of cities from a list of $n$ cities. We want to find a “tour” that visits each city once while minimizing the total travel cost as given by the table. What is the asymptotic running time of your algorithm. What size tour would you expect to be able to optimize in ten minutes on a 500 MHz computer this way? For this, you should make a reasonable guess at the number of cycles needed for the steps in your algorithm.

3. Find the asymptotic running time for the following as a function of $n$, and don’t worry that this code doesn’t do anything.

   ```
   L = 0
   for (k=1,n) {
     L = L + k
     for(i=0, L-1)
       for(j=0,L-1)
         a[i][j] = (i + j) / (i*i + j*j)
   }
   ```

4. An array of numbers, $a$, is “unimodal” if has only one “local maximum”. We say that $a(k)$ is a local maximum if $a(k) > a(k+1)$ and $a(k) > a(k-1)$. The array [1, 2, 3, 4, 5, 3, 1] is unimodal while [1, 3, 2, 4, 5, 3, 1] is not. Write an algorithm that finds the maximum in a unimodal array in $O(\log(n))$ time using some sort of bisection. Note, the best possible algorithm of this kind is called “golden section search”. Feel free to look this up in a book on optimization algorithms.