CS1112-206/211/212–Fall 2009

Lab 14 Solution

Friday, December 4

Since a solution was presented in section, please refer to the solution posted on the section page for how to implement the solution discussed below.

1 Writing Efficient Code

(a) We need to initialize t_1 outside the `for` loop. Then, we need to index into t_1 correctly, e.g., index to position 1 if $n = 100$. One way to do this is to keep track of the index in another variable, say m. Be sure not to reuse any variables names already used in the program, or the program will become incorrect. Alternatively, we can calculate the index directly from n using the formula $(n - 50)/50$.

(b) We can ensure that each triangle is evaluated once by making sure that the `for` loops run through each possible point combination once. This can be done by making sure that the value of i is always less than j, and the value of j is always less than k. That is, for each i, start j at $i + 1$, and for each j, start k at $j + 1$. We can deduce further that, by doing it this way, the maximum possible value of i (that is useful) is $n - 2$, and the maximum possible value of j is $n - 1$.

Next, once we know i, we can calculate sine and cosine right away without waiting until the k value is assigned. That way we do not repeat the calculation in the innermost loop. Hence, we can move the two function calls outside to just after the beginning of the `for` loop. We can do the same for j.

(c) We can improve the running time of our program further by storing information that we frequently need in a matrix, so that we can just look it up as needed, without calculating it again. The information we will need frequently is the distance between any two points. We can simply precalculate these distances prior to finding the maximum area. This can be done by adding another nested loop before the existing nested loop and, inside the new nested loop, calculate the distances. Once we have the distance matrix, we need not call `sin` and `cos` anymore in the original nested loop. Instead, we just have to refer to the matrix when distances are needed.

(d) This part of the exercise is simply data representation, and we do not discuss this further.

(e) We can simply use ratio to calculate running time here. Let a be the running time for $n = 100$ and b be the running time for $n = 200$. That is, if the input size doubles, the running time increases by the factor of b/a. What if the input size quadrupled? Then the running time should increase by $(b/a)^2$ (because the input size doubles twice. What if the input size is k times the original input size? In that case, how many times do we "double" the input size? If we double three times, then the input size is 8 times the original size. Hence, if we double p times, the input size is 2^p times the original size. So, if the input size is k times the original size, we must have doubled $\log_2 k$ times.

Therefore, if $n = 1000$, then the input size is 5 times the original size, so we have doubled the size $\log_2 5 \approx 2.3219$ times. Therefore, the running time is approximately

$$\left(\frac{b}{a}\right)^{\log_2 5}$$

when $n = 1000$.