CS2223
HW#1

DUE: Monday, March 20

1. (5 points) Suppose you want to generate an arbitrarily long stream of random bits on your computer. Describe how you will do this. Compute how much time it takes you to generate each bit. Describe your implementation.

2 (15 points) Assume you are given a set of *n* points in 2-dimensional space, $P_1,...,P_n$, where each P_i is actually a pair of numbers (x_i, y_i). We want to compute δ_{min}, the distance between a closest pair of distinct points, where the Euclidean distance between points P_i and P_j is $\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$. Consider the following algorithm to compute δ_{min}.

$\delta_{min} \leftarrow \infty$
for $i \leftarrow 1$ **to** *n* **do**
  for $j \leftarrow 1$ **to** *n* **do**
    if $i \neq j$ **and** $\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} < \delta_{min}$
    then $\delta_{min} \leftarrow \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$
return δ_{min}

a Exactly how many times is $\sqrt{}$ computed by the above algorithm?

b Write a program to implement the above algorithm, and run and time the program for *n* randomly generated points for several values of *n*, including *n*=1000 and *n*=10000. Describe your implementation, and the run time of your program as a function of *n*. For example, if the above algorithm takes quadratic time, then you'd say something like "My program takes about $1.635 * n^2$ seconds".

c Describe some changes to the above algorithm to make your program faster. For each change, describe the run time of your new program as a function of *n*.

CS2223
HW#1 SOLUTIONS

1. In Maple 10 on my Windows machine, after defining *roll*:=*rand*(0..1): every invocation of *roll*() returns a random bit. In order to time the execution of *roll*(), we first time an empty **for**-loop

```
> st := time( ) : for n from 1 to 10000000 do od : time( ) - st;
```

6.09

which took 6.09 seconds. The routine with a **for**-loop loaded with a random bit generator

```
> roll := rand(0..1) : st := time( ) : for n from 1 to 10000000
    do roll( ) : od : time( ) - st;
```

30.885

The difference of 24.795 seconds means that on average each invocation of *roll*() took 2.4795 μ-seconds.

2.

c Computing $\sqrt{\ }$ is expensive and is monotonic, $x \geq y \Rightarrow \sqrt{x} \leq \sqrt{y}$, so a faster algorithm which only computes one $\sqrt{\ }$ is

$\delta_{min} \leftarrow \infty$
for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow 1$ **to** n **do**
    if $i \neq j$ **and** $\left(x_j - x_i\right)^2 + \left(y_j - y_i\right)^2 < \delta_{min}$
    then $\delta_{min} \leftarrow \left(x_j - x_i\right)^2 + \left(y_j - y_i\right)^2$
return $\sqrt{\delta_{min}}$

Since distance is a symmetric function, a faster algorithm is

$\delta_{min} \leftarrow \infty$
for $i \leftarrow 1$ **to** n-1 **do**
  for $j \leftarrow i+1$ **to** n **do**
    if $\left(x_j - x_i\right)^2 + \left(y_j - y_i\right)^2 < \delta_{min}$
    then $\delta_{min} \leftarrow \left(x_j - x_i\right)^2 + \left(y_j - y_i\right)^2$
return $\sqrt{\delta_{min}}$

Rather than computing $\left(x_j - x_i\right)^2 + \left(y_j - y_i\right)^2$ twice each time in the **for**-loop, it is faster to compute it once.

$\delta_{min} \leftarrow \infty$

for $i \leftarrow 1$ **to** n-1 **do**

    for $j \leftarrow i+1$ **to** n **do**

        $temp \leftarrow \left(x_j - x_i\right)^2 + \left(y_j - y_i\right)^2$

        if $temp < \delta_{min}$

        then $\delta_{min} \leftarrow temp$

return $\sqrt{\delta_{min}}$