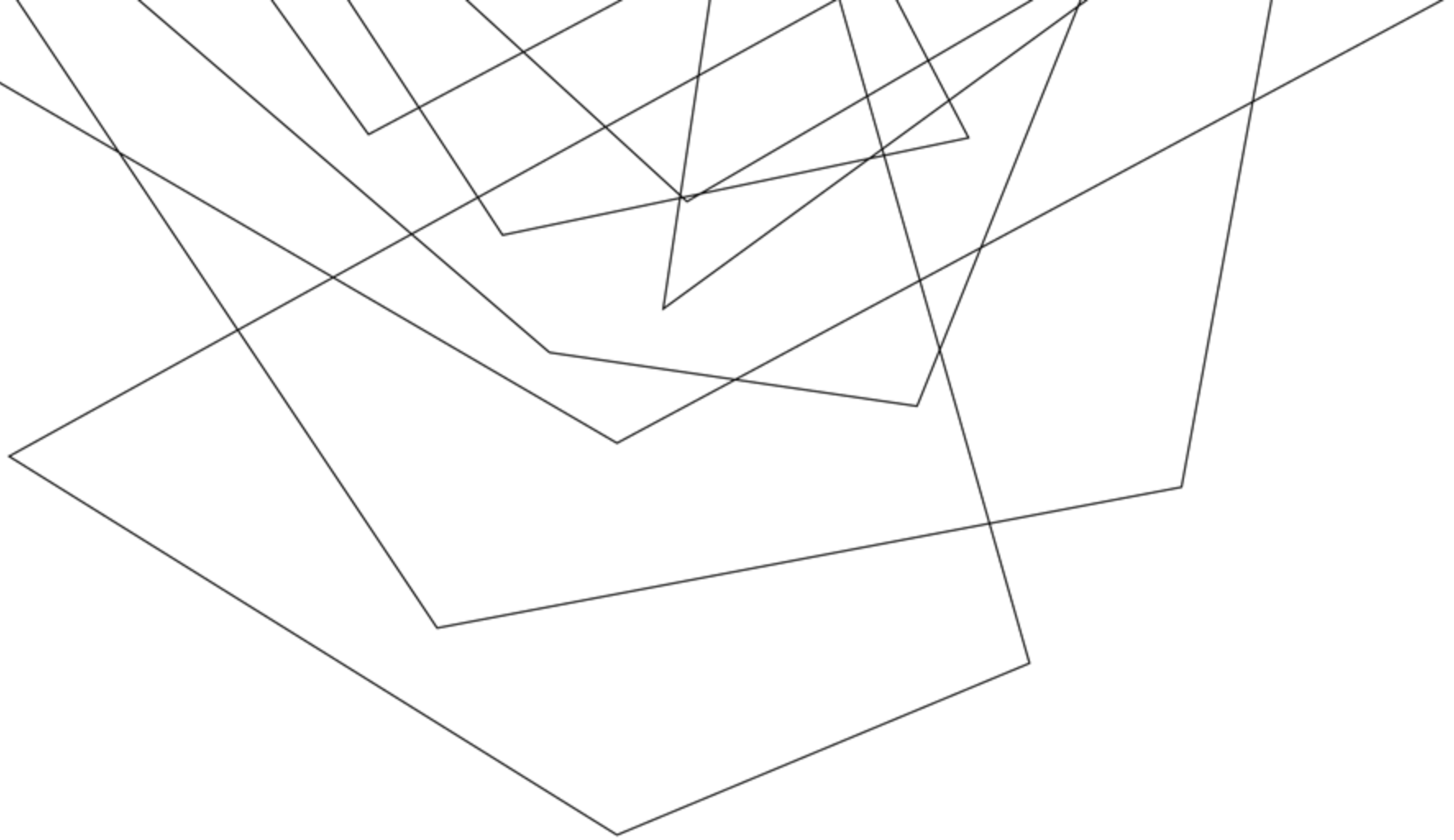

CMPT 225 D100 LAB03

TA

TOPICS FOR TODAY

Empirically measuring performance

- Using the chrono library
- Simple counting

Performance of Functions

- Given any function, we want them to have **good performance**, which can be measured in two main ways:
  - How much time it needs to produce the output (time performance)
  - How much memory space it needs to produce the output (space performance)
- Here we examine the **time performance**, which can be measured by:
  - Actual time for a function to finish
  - Simple counting of number of steps involved

Demo: Using the chrono Library

- See the example: fibonacci.cpp
- What does each line of code do?

```
std::chrono::time_point<std::chrono::system_clock> start, end;

start = std::chrono::system_clock::now();
std::cout << "f(42) = " << fibonacci(42) << '\n';
end = std::chrono::system_clock::now();

std::chrono::duration<double> elapsed_seconds = end - start;
std::cout << "elapsed time: " << elapsed_seconds.count() << "s\n";
```

Overhead

- In lecture, we said recursion is often *slightly* slower than using a loop.
- But since this delay is a constant factor, it is insignificant
- Example:
  - Let $f(n)$ be the time taken for computing the sum of an array of size *n* using an iterative function
  - Let $g(n)$ be the time taken for computing the sum of an array of size *n* using a recursive function
  - Then the relationship is $g(n) = c\ f(n)$ where **c** is a constant number

Overhead

- Similarly, we said using the standard library array is *slightly* slower than using a static array
- Example:
  - Let $f(n)$ be the time taken for computing the sum of a static array of size *n*
  - Let $g(n)$ be the time taken for computing the sum of an array object of size *n*
  - Then the relationship is $g(n) = c\ f(n)$ where **c** is a constant number

Exercise

- The **meausingTime.cpp** file has three functions:
  - linearSumLoop – computes the sum of a static array of size n using iteration
  - linearSumRec – computes the sum of a static array of size n using recursion
  - linearSumLoop – computes the sum of array object of size n using iteration

- Using the chrono library, complete the function testRec. This function should time how long it takes to run the linearSumLoop and time how long it takes to run linearSumRec for the given value of n.
- Using the chrono library, complete the function testClass. This function should time how long it takes to run the linearSumLoop with the static array and time how long it takes to run linearSumLoop with the array object for the given value of n.

Exercise

- We never stop at only one sample. We always look at many samples to watch the change as *n* grows. The main function runs tests for n = 64, 128, 256, …
- When *n* is sufficiently large, we start to notice the pattern:
  - **c** = $f(n)$ / $g(n)$
- For each test, we want to print: $f(n)$, $g(n)$ and $f(n)/g(n)$

Results for Iterative vs. Recursive test

n	iterative	recursive	difference
64	1.95e-05	6.15e-05	3.1538462
128	0.000145083	0.000243041	1.6751859
256	0.000254875	0.000402625	1.5796959
512	0.000256875	0.002280833	8.8791552
1024	0.003010667	0.014083042	4.6777149
2048	0.007035375	0.035506458	5.0468465
4096	0.017129084	0.11134671	6.5004474
8192	0.062752708	0.45847571	7.3060705
16384	0.25031242	1.8562124	7.4155827
32768	1.0004606	7.6025953	7.5990952

Given this data, what is **c**?

Results for Static Array vs. Array Object test

n	static	class	difference
64	4.709e-06	9.792e-06	2.0794224
128	1.6834e-05	3.8625e-05	2.2944636
256	6.625e-05	0.000150666	2.2742038
512	0.000253292	0.000575084	2.2704389
1024	0.000991958	0.002288333	2.306885
2048	0.003924125	0.009104542	2.3201458
4096	0.015661917	0.036697	2.3430721
8192	0.062871875	0.14567558	2.3170231
16384	0.25176996	0.58214663	2.3122164
32768	1.0048156	2.3344936	2.3233055

Given this data, what is **c**?

Simple Counting

- Another way to measure time performance is to count how many calculations a function does (e.g., adding things, assigning things, comparing things, …etc.) as each will take up some CPU time
- In CS we have a generalized way to count & report this measurement – Big-O notation, which you will learn in class