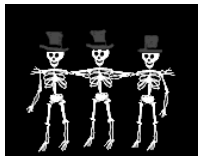
Performance Measurement



Performance Analysis

- Paper and pencil.
- Don't need a working computer program or even a computer.

Some Uses Of Performance Analysis

- determine practicality of algorithm
- predict run time on large instance
- compare 2 algorithms that have different asymptotic complexity
  - e.g., O(n) and O(n^2)

Limitations of Analysis

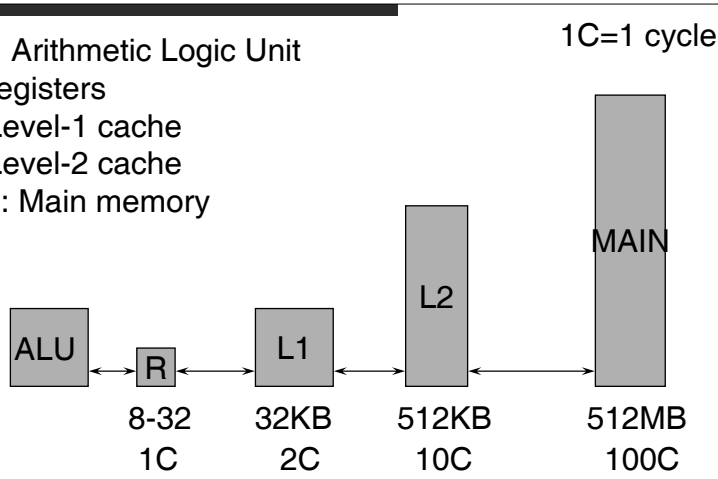
- Doesn't account for constant factors.
- but constant factor may dominate
  - 1000n vs n^2
- and we are interested only in n < 1000

Limitations of Analysis

- Modern computers have a hierarchical memory organization with different access time for memory at different levels of the hierarchy.

Memory Hierarchy

ALU: Arithmetic Logic Unit
R: Registers
L1: Level-1 cache
L2: Level-2 cache
Main: Main memory



Limitations of Analysis

- Our analysis doesn't account for this difference in memory access times.
- Programs that do more work may take less time than those that do less work.
- Compare:
  - 100 operations on the same data
  - 10 operations on the different data

Performance Measurement

- Measure actual time on an actual computer.
- What do we need?

Performance Measurement Needs

- programming language
- working program
- computer
- compiler and options to use

Performance Measurement Needs

- data to use for measurement
  worst-case data
  Insertion sort: 5 4 3 2 1
  best-case data
  insertion sort: 1 2 3 4 5
  average-case data

- timing mechanism --- clock

Timing In C++

```
long start, stop;

time(start); // set start to current time in
             // hundredths of a second

// code to be timed comes here

time(stop); // set stop to current time

long runTime = stop – start;
```


Shortcoming

- Preceding measurement code is acceptable only when the elapsed time is large relative to the accuracy of the clock.
  Clock accuracy: assume 1/100 second

If code to be timed is too small. We should repeat work many times to bring total time larger, says 1/10 sec.

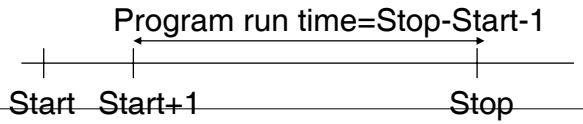
Accurate Timing

```
time(start);
long counter;
do {
        counter++;
        doSomething();
        time(stop);
     } while (stop - start < 10)
double elapsedTime = stop - start;

double timeForTask = elapsedTime/counter;
```


Accuracy

- Now accuracy is 10%.
- first reading may be just about to change to start + 1
- second reading may have just changed to stop
- so stop - start is off by 1 unit

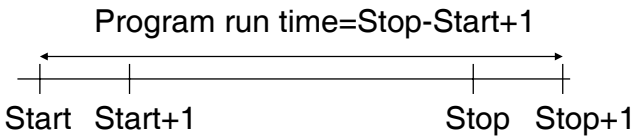


Accuracy

first reading may have just changed to start

second reading may be about to change to stop + 1

so stop - start is off by 1 unit



Accuracy

Examining remaining cases, we get

trueElapsedTime = stop - start +- 1

To ensure 10% accuracy, require

elapsedTime = stop – start
>= 10

What Went Wrong?


```
time(start);
long counter;
do {
        counter++;
        insertionSort(a,n);
        time(stop);
    } while (stop - start < 10)
double elapsedTime = (stop – start);

double timeToSort = elapsedTime/counter;
```

Recall insertion sort:
Worst case: O(n^2) (inverse order)
Best case: O(n) (already sorted)
Measure worst-case running time:
1st time→ worst case, others: best case

The Fix


```
time(start);
long counter;
do {
        counter++;
        // put code to initialize a here
        insertionSort(a,n);
        time(stop);
    } while (stop - start < 10)
```

Elapsed time=Initial time+ Sorting time

In Class Exercise: Why below code is not a good way to time?

```
do {
        counter++;
        time(start);
        doSomething();
        time(stop)
        elapsedTime += stop - start;
    } while (elapsedTime < 10)
```