Performance Measurement

Data structures

Spring 2017

Performance Analysis

Can be done with paper and pencil.

Don’t need a working computer program or even a computer (correct algorithm is enough).

Some Uses Of Performance Analysis

- determine practicality of algorithm
- predict run time on large instances
- compare two 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

E.g. $1000n$ vs n^2

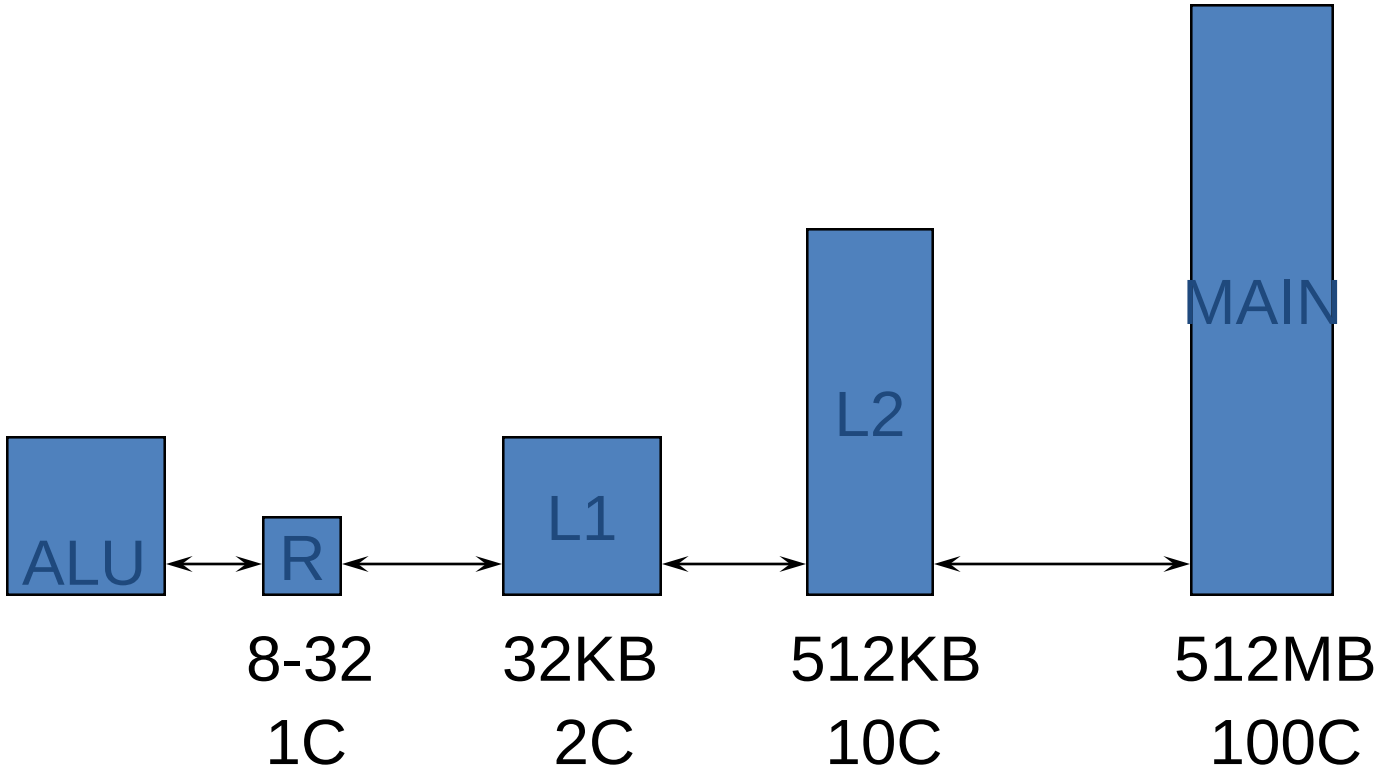
and we are interested only in

$n < 1000$

Limitations of Analysis

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

Memory Hierarchy



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.

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

  javac -o

Performance Measurement Also Needs

- data to use for measurement

  worst-case data

  best-case data

  average-case data

- timing mechanism --- clock

Timing In Java


```
long startTime = System.currentTimeMillis();
// gives time in milliseconds since 1/1/1970 GMT

// code to be timed comes here

long elapsedTime = System.currentTimeMillis()
                                    - startTime;
```

Shortcoming

Clock accuracy

assume 100 milliseconds

Repeat work many times to bring total time to be >= 1 second

Accurate Timing

```
long startTime = System.currentTimeMillis();
long counter;
do {
        counter++;
        doSomething(); // code to be timed
     } while (System.currentTimeMillis() -
                startTime < 1000)
long elapsedTime = System.currentTimeMillis()
                            - startTime;
float timeForMethod =
                ((float) elapsedTime)/counter;
```

Accuracy

Now accuracy is 90%.

first reading may be just about to change to
    startTime + 100

second reading may have just changed to finishTime

so finishTime - startTime is off by 100ms

Accuracy

first reading may have just changed to startTime

second reading may be about to change to
finishTime + 100

so finishTime - startTime is off by 100ms

Accuracy

Examining remaining cases, we get

trueElapsedTime =

finishTime - startTime +- 100ms

To ensure 90% accuracy, require

elapsedTime = finishTime – startTime

>= 1sec

What can go wrong?

```
long startTime = System.currentTimeMillis();
long counter;
do {
        counter++;
        InsertionSort.insertionSort(a);
     } while (System.currentTimeMillis() -
                startTime < 1000)
long elapsedTime = System.currentTimeMillis()
                           - startTime;
float timeForMethod =
                ((float) elapsedTime)/counter;
```

The Fix

```
long startTime = System.currentTimeMillis();
long counter;
do {
        counter++;
        // put code to initialize a[] here
        InsertionSort.insertionSort(a);
    } while (System.currentTimeMillis() -
                startTime < 1000)
```

Bad Way To Time

```
do {
        counter++;
        startTime = System.currentTimeMillis();
        doSomething();
         elapsedTime +=
                System.currentTimeMillis()
                    - startTime;
    } while (elapsedTime < 1000)
```

Exercise

Arrange the following complexity terms in the non-decreasing order of their asymptotic growth:

$2^{3\log n}$, $n^{\log n}$, $n^2(\log n)^6$, $(1.1)^n$, $(\log n)^{(1/2)\log n}$, $n^{2.5}$, $(\log n)^{n/3}$, $4^{\sqrt{n}}$, $6^{\log n}$