CS206

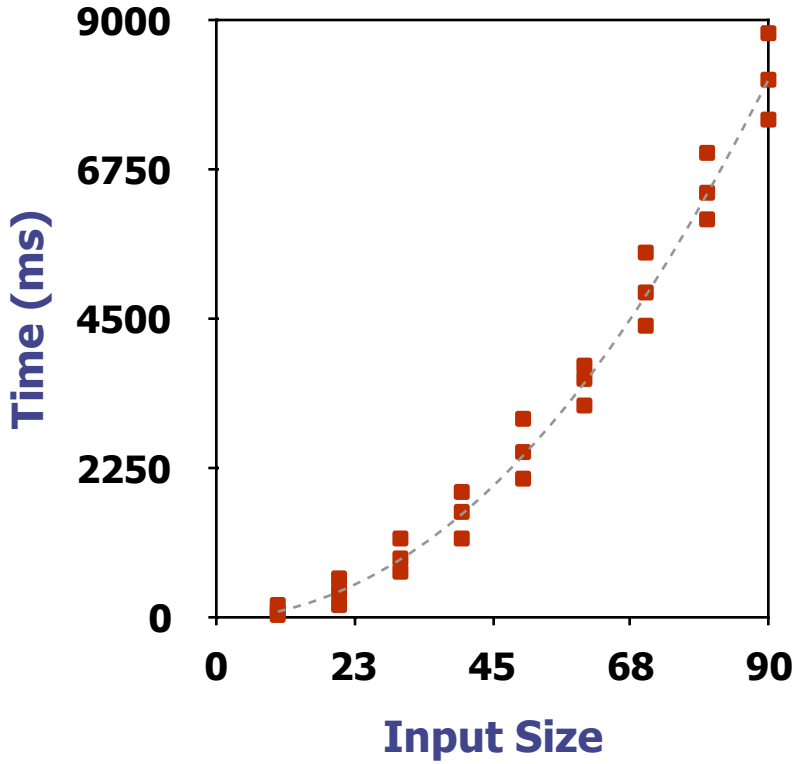
Analysis of Algorithms

Running Time

- How long a program runs depends on
  - efficiency of the algorithm/implementation
  - size of input
- The running time typically grows with input size
- Worst case analysis
  - how long will it take in the worst case?

Experimental Study

- Write a program implementing the algorithm
- Run it with different input sizes and compositions
- Record times and plot results



Timing Code

```
public class Timer {
    private static final int REPS = 5;
    private static final int NANOS_SEC = 1000000000;
    static int[] doSomething() {
     int[] karr = new int[10000];
     for (int i=1; i<100; i++)
         for (int j=1; j<1000; j++)
          for (int k=1; k<10000; k++) {
              karr[k] = i*j*k;
          }
     return karr;
    }
    public static void main(String[] args) {
     long data[] = new long[REPS];
     for (int i=0; i<REPS; i++) {
         long start = System.nanoTime();
         doSomething();
         long finish = System.nanoTime();
         data[i] = (finish-start);
         System.out.println(String.format("Run Time %2d: %12d ns", i, (finish-start)));
     }
     long tt = 0;
     for (int i=0; i<REPS; i++)
         tt += data[i];
     System.out.println(String.format("Average Time: %6.4f", ((double)(tt/REPS))/NANOS_SEC));
    }
}
```

Limitation of Experiments

- You have to implement the algorithm
- You have to generate inputs that represent all cases
- Comparing two algorithms requires exact same hardware and software environments
  - Even then timing is hard
    - multiprocessing
    - file i/o

Theoretical Analysis

- Uses high-level description of algorithm
  - pseudo-code
- Characterizes running time as a function of the input size, n
- Takes into account all possible inputs
- Allows us to evaluate the speed of an algorithm independent of the hardware/ software environment

Primitive Operations

- Basic computations
- Not looped
- Assumed to take constant time
  - exact constant is not important

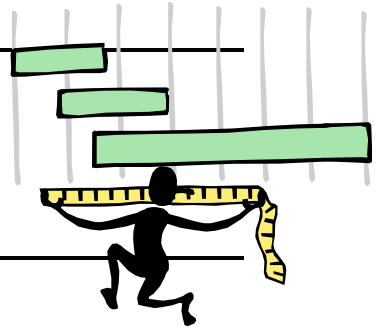
Example
Time required to compute an average

```
public static double calcAverage(long[] data)
{
     double res = 0;
     for (int i=0; i<data.length; i++)
     {
          res += data[i];
     }
     return res/data.length;
}

public static double calcAverage2(long[] data) {
     double res = 0;
     long pd = 0;
     for (long datum : data) {
         if (pd<datum) {
          res += datum;
         }
         pd=datum;
     }
     return res/data.length;
}
```

How many operations?

Estimate Running Time

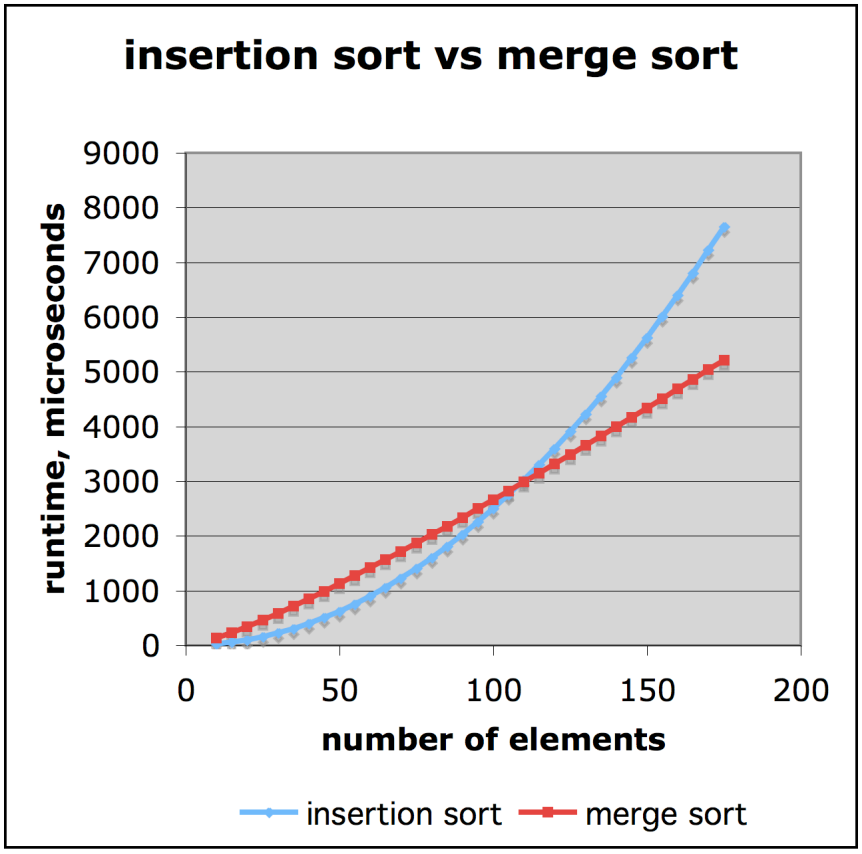


- `calcAverage2` executes a total of 6N+1 primitive operations in the worst case, 4N+1 in the best case.
- Let a = fastest primitive operation time, b = slowest primitive operation time
- Let $T(n)$ denote the worst-case time of `average2`: $a(4n+1) \leq T(n) \leq b(6n+1)$
- $T(n)$ is bounded by two linear functions – linear in terms of n

Growth Rate of Running Time

- Changing the hardware/ software environment
  - Affects $T(n)$ by a constant factor, but
  - Does not alter the growth rate of $T(n)$
- The linear growth rate of the running time $T(n)$ is an intrinsic property of algorithm `average2` (and `average`)

Comparison of Two Algorithms



- insertion sort: $n^2/4$
- merge sort: $2nlgn$
- $n = 1000000$
  - insertion sort:
    70 hours
    40 minutes
  - merge sort:
    40 seconds
    0.5 seconds

Asymptotic Notation

- Provides a way to simplify analysis
- Allows us to ignore less important elements
  - constant factors
- Focus on the largest growth of n
- Focus on the dominant term

How do these functions grow?

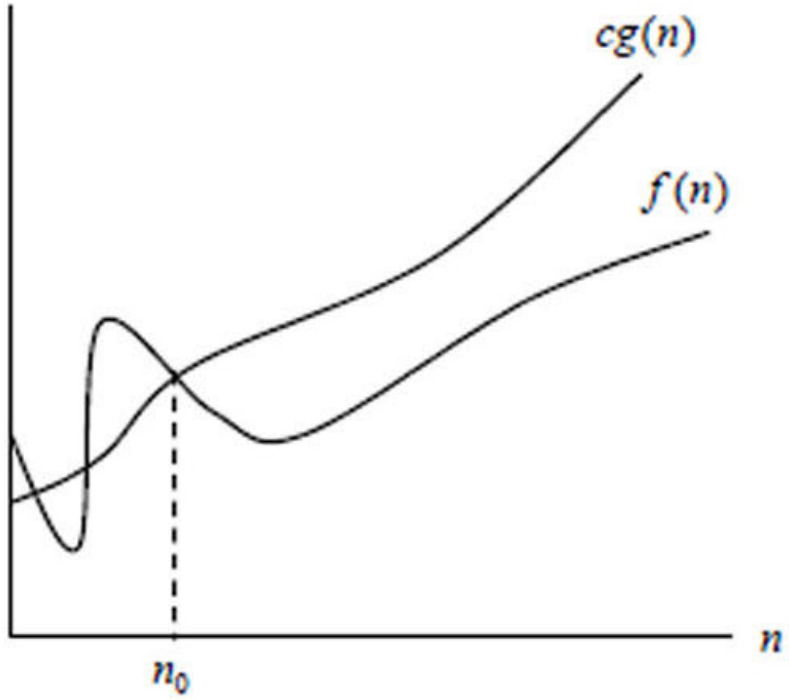
- $f_1(x) = 43n^2 \log^4 n + 12n^3 \log n + 52n \log n$
- $f_2(x) = 15n^2 + 7n \log^3 n$
- $f_3(x) = 3n + 4 \log_5 n + 91n^2$
- $f_4(x) = 13 \cdot 3^{2n+9} + 4n^9$

Big O

$\exists\ n_0 \geq 0, c > 0,$ if $f(n) \leq c \bullet g(n) \forall n \geq n_0,$

then $f(n) = O(g(n))$

- Constant factors are ignored
- Upper bound



Big O and Growth Rate

- Big-O notation gives an upper bound on the growth rate
- $f(n) = O(g(n))$ means that the growth of $f(n)$ is no more than $g(n)$
- $f(n)$ can be a complicated polynomial
- $g(n)$ is one of a few highly-recognizable simple polynomials

Examples

- $7n + 2$
- $3n^3 + 20n^2 + 5$
- $3lgn + 5$

Summations

- Constant series

$$\sum_{i=a}^{b} 1 = \max(b - a + 1, 0)$$

- Arithmetic series

$$\sum_{i=1}^{n} i = 1 + 2 + \ldots + n = \frac{n(n+1)}{2} \in \Theta\left(n^2\right)$$

- Quadratic series

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \ldots + n^2 = \frac{2n^3 + 3n^2 + n}{6} \in \Theta\left(n^3\right)$$

Linear Time Algorithms: $O(n)$

- The algorithm's running time is at most a constant factor times the input size
- Process the input in a single pass spending constant time on each item
  - max, min, sum, average, linear search
- Any single loop

$O(nlogn)$ time

Frequent running time in cases when algorithms involve:

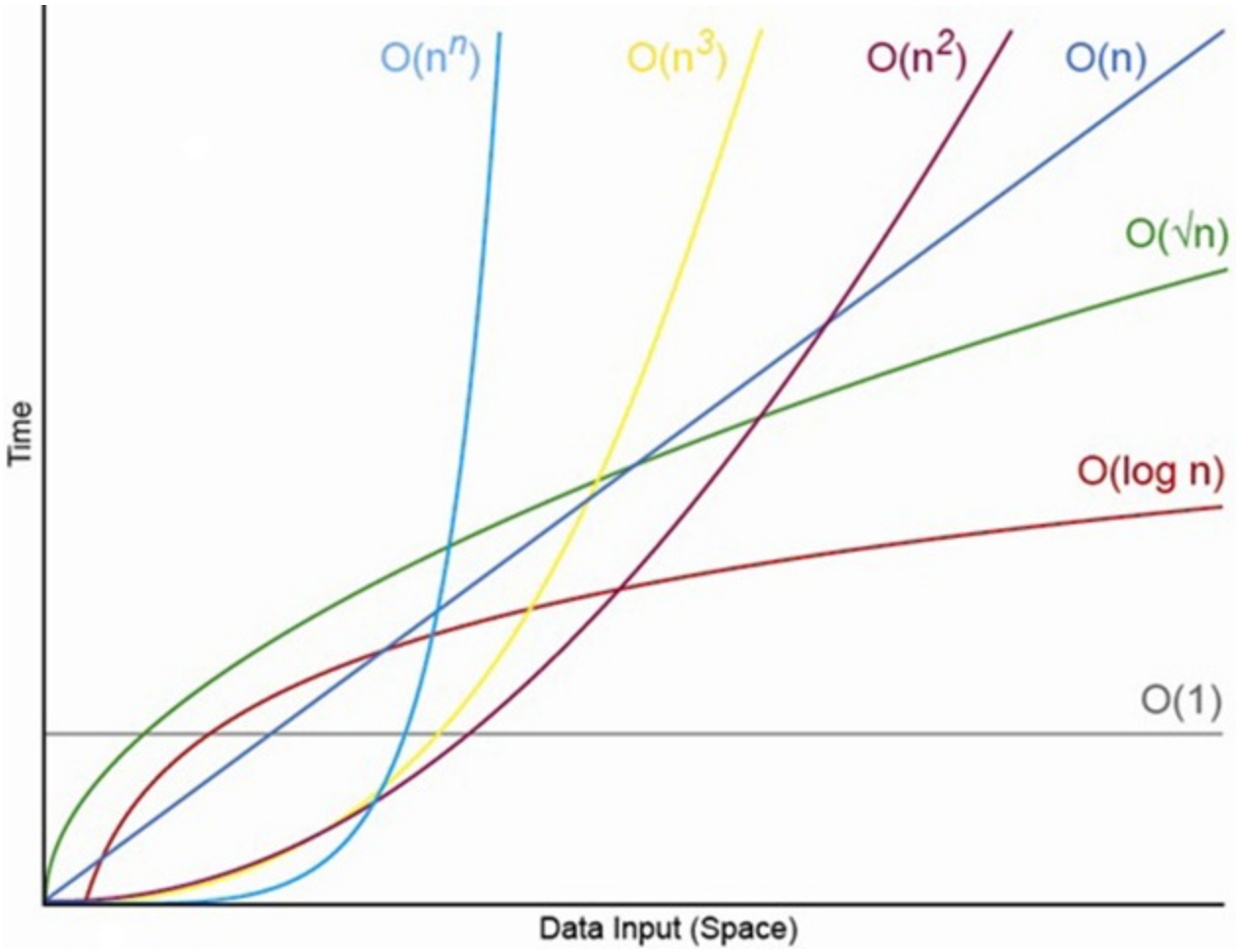
- Sorting
  - only the "good" algorithms
    - e.g. quicksort, merge sort, …
- Divide and conquer
  - binary search

Quadratic Time: $O(n^2)$

- Nested loops, double loops
  - Your assignment 2: find a zipcode match in an `ArrayList` of n elements, n times
- Processing all pairs of elements
- Other sorting algorithms
  - insertion sort

Slow Times

- All subsets of n elements of size k: $O(n^k)$
- All subsets of n elements (power set): $O(2^n)$
- All permutations of n elements: $O(n!)$


O(n^n)
O(n^3)
O(n^2)
O(n)
O(√n)
O(log n)
O(1)
Time
Data Input (Space)