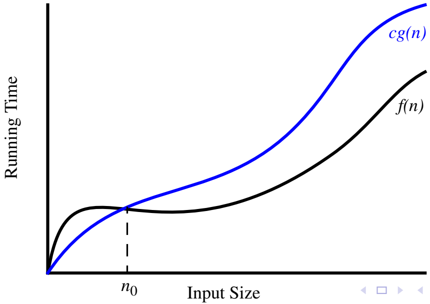
Algorithm Analysis

Jianguo Lu



Overview

What is an Algorithm

Definition: Algorithm

- An algorithm is any well-defined computational procedure that takes some value(s) as input, and produce some value(s) as output.
- A sequence of computational steps that transform the input into the output.

Algorithm 1: Selection Sort

Input: Array A of length n

Output: Sorted A

```
for int i =0; i < n-1; i++ do
    min= minimal element in array[i+1:n];
    swap array[i] with min;
```

Algorithm differs from a problem specification

A formal specification of the sorting problem:

Input: A sequence of numbers $(a_1, a_2, \ldots, a_n)$.

Output: A permutation(reordering) $(a_{x1}, a_{x2}, \ldots, a_{xn})$ of the input, such that

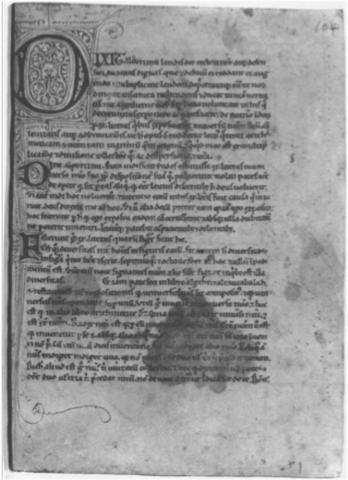
$$a_{x1} \leq a_{x2} \leq \cdots \leq a_{xn} \tag{1}$$

where $x_i \in \{1, 2, \ldots n\}$ and $x_i \neq x_j$ for all $i, j \in \{1, 2, \ldots n\}$

An algorithm is a solution to the problem

Etymology of “Algorithm”

- Al-Khwarizmi was a 9th-century scholar, born in present-day Uzbekistan, who studied and worked in Baghdad during the Abbassid Caliphate.
- Among many other contributions in mathematics, astronomy, and geography, he wrote a book about how to multiply with Arabic numerals.
- His ideas came to Europe in the 12th century.
- Originally, “Algorisme” [old French] referred to just the Arabic number system, but eventually it came to mean “Algorithm” as we know today.



What to analyse

- Correctness: the algorithm satisfies its specification. Also involves termination (the algorithm stops).
    - Formal verification/proof of the correctness of program (Comp-4400)
    - Software testing
- Performance
    - Run time
    - Space
- Algorithm analysis is to determine the computational complexity.
- Mostly on running time.

But running time depends on data size....

How to evaluate/analyze an algorithm

- The running time of an algorithm typically grows with the input size.
- Hence we evaluate algorithms in terms of *functions*.

Even for the same data size, every-run is different.

How to evaluate/analyze an algorithm

- The running time of an algorithm typically grows with the input size.
- Hence we evaluate algorithms in terms of *functions*.

Even for the same data size, every-run is different.

- Average case time is often difficult to determine.
- We often focus on the worst case running time.
    - Easier to analyze
    - Crucial to applications such as games, finance and robotics

How to evaluate/analyze an algorithm

- The running time of an algorithm typically grows with the input size.
- Hence we evaluate algorithms in terms of *functions*.

Even for the same data size, every-run is different.

- Average case time is often difficult to determine.
- We often focus on the worst case running time.
  - Easier to analyze
  - Crucial to applications such as games, finance and robotics

How to know the run time?

Approach 1: Experimental analysis

- Write a program that implements the algorithm
- Run the program with inputs of varying size and composition
- Keep track of the CPU time used by the program on each input size
- Plot the results on a two-dimensional plot

How to measure the speed of your code?

```
long start = System.currentTimeMillis();
long end =  System.currentTimeMillis();
long elapsed-time=end-start
```

Approach 1: Experimental analysis

- Write a program that implements the algorithm
- Run the program with inputs of varying size and composition
- Keep track of the CPU time used by the program on each input size
- Plot the results on a two-dimensional plot

How to measure the speed of your code?

```
long start = System.currentTimeMillis();
long end =  System.currentTimeMillis();
long elapsed-time=end-start
```

Limitations?

Disadvantage of experimental analysis

- Need to implement the algorithm and debug the programs
- Can't predict for very large data (we can't run for 31 years)
- Experimental evaluation depends on
  - Hardware;
  - Programming language;
  - Data (e.g., partially sorted data may favour one algorithm);
  - If all above are the same, whether the run time is the same?

Disadvantage of experimental analysis

- Need to implement the algorithm and debug the programs
- Can't predict for very large data (we can't run for 31 years)
- Experimental evaluation depends on
  - Hardware;
  - Programming language;
  - Data (e.g., partially sorted data may favour one algorithm);
  - If all above are the same, whether the run time is the same?
  - Each run is different (e.g., garbage collection)

The impact of data on algorithms

Animation can be viewed using Acrobat. Preview won't play the sorting steps.

Approach 2: Theoretical analysis

- Often uses a high-level description of the algorithm instead of an actual implementation
- 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 independently of the hardware/software environment

Pseudo code

- High-level description of an algorithm
- More structured than English prose
- Less detailed than a program
- Preferred notation for describing algorithms
- Hides program design issues

Algorithm 2: Selection Sort

Input: Array A of length n
Output: Sorted A

```
for int i =0; i < n-1; i++ do
    min= minimal element in array[i+1:n];
    swap array[i] with min;
```

Primitive operations

- Basic computations performed by an algorithm
- Identifiable in pseudocode
- Largely independent from the programming language
- Exact definition not important
- Assumed to take a constant amount of time in the RAM model

Examples:

- Evaluating an expression
- Assigning a value to a variable
- Indexing into an array
- Calling a method
- Returning from a method

Counting primitive operations

```
double arrayMax(double[] data) {
    int n = data.length;          // 2 ops
    double currentMax = data[0];      // 2 ops
    for (int j=0; j < n; j++)         // 2n  ops
      if (data[j] > currentMax)       // 2n  ops
        currentMax = data[j];         // 0 to n
    return currentMax;                  //1 op
  }
```

Number of operations:

$$ops = \begin{cases} 2+2+2n+2n+0+1 = 4n+5 & \text{Best case} \\ 2+2+2n+2n+n+1 = 5n+5 & \text{Worst case} \end{cases} \quad (1)$$

- But operations have different costs. What is the total cost?

Counting primitive operations

```
double arrayMax(double[] data) {
    int n = data.length;          // 2 ops
    double currentMax = data[0];      // 2 ops
    for (int j=0; j < n; j++)         // 2n  ops
      if (data[j] > currentMax)       // 2n  ops
        currentMax = data[j];         // 0 to n
    return currentMax;                    //1 op
  }
```

Number of operations:

$$ops = \begin{cases} 2+2+2n+2n+0+1 = 4n+5 & \text{Best case} \\ 2+2+2n+2n+n+1 = 5n+5 & \text{Worst case} \end{cases} \tag{1}$$

- But operations have different costs. What is the total cost?
- We can not have an exact cost

Counting primitive operations

```
double arrayMax(double[] data) {
    int n = data.length;          // 2 ops
    double currentMax = data[0];      // 2 ops
    for (int j=0; j < n; j++)         // 2n  ops
      if (data[j] > currentMax)       // 2n  ops
        currentMax = data[j];       // 0 to n
    return currentMax;                  //1 op
  }
```

Number of operations:

$$ops = \begin{cases} 2+2+2n+2n+0+1 = 4n+5 & \text{Best case} \\ 2+2+2n+2n+n+1 = 5n+5 & \text{Worst case} \end{cases} \quad (1)$$

- But operations have different costs. What is the total cost?
- We can not have an exact cost
- Instead we give upper bounds and lower bounds

Upper and lower bounds

Define:

- a= Time taken by the fastest primitive operation
- b = Time taken by the slowest primitive operation
-

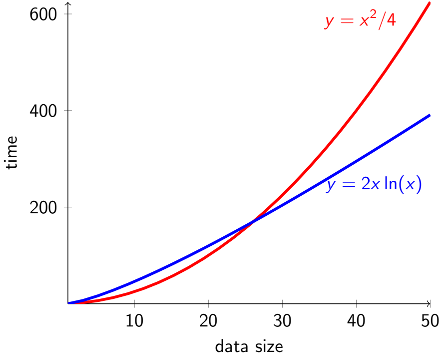
$$a(4n+5) \leq T(n) \leq b(5n+5) \tag{1}$$

- Hence, the running time T(n) is bounded by two linear functions

Growth Rate of Running Time

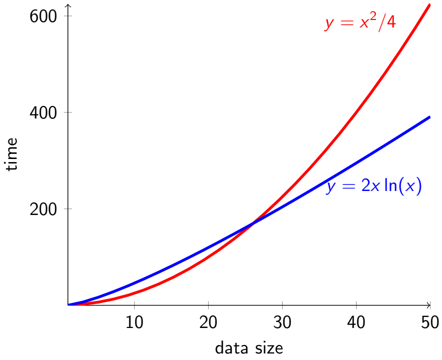
- a and b are constants determined by hardware/ software environment
- n can be very large
- What matters is how fast T(n) grows with n.
- How to compare growth functions?

Comparing Insertion sort and merge sort



- Insertion sort is $x^2/4$.
- Merge sort is $2x\log(x)$.
- Which one is faster?

Comparing Insertion sort and merge sort



- Insertion sort is $x^2/4$.
- Merge sort is $2x\log(x)$.
- Which one is faster?
- What matters is "which one is faster **aymptotically**"

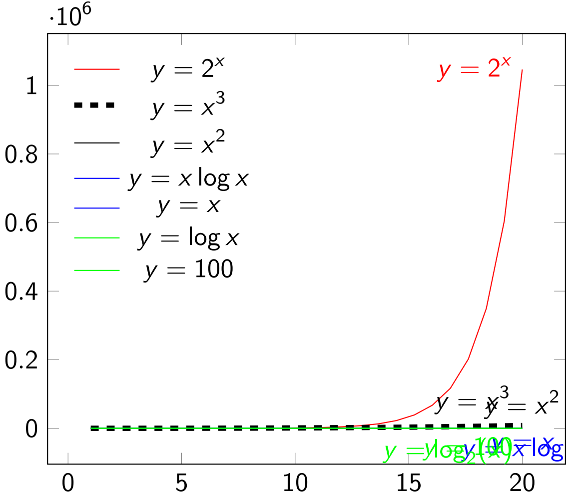
Asymptotic:

(of a function) approaching a given value as an expression containing a variable tends to infinity.

Seven functions

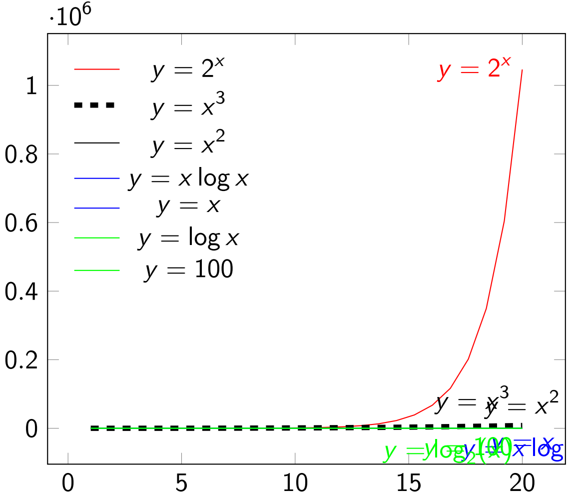
Constant	$f(x) = C$
Logarithmic	$f(x) = \log_2(x)$
Linear	$f(x) = x$
Linearithmic	$f(x) = x\log_2(x)$
Quadratic	$f(x) = x^2$
Cubic	$f(x) = x^3$
Exponential	$f(x) = 2^x$

Growth rates of the 7 functions



Observations

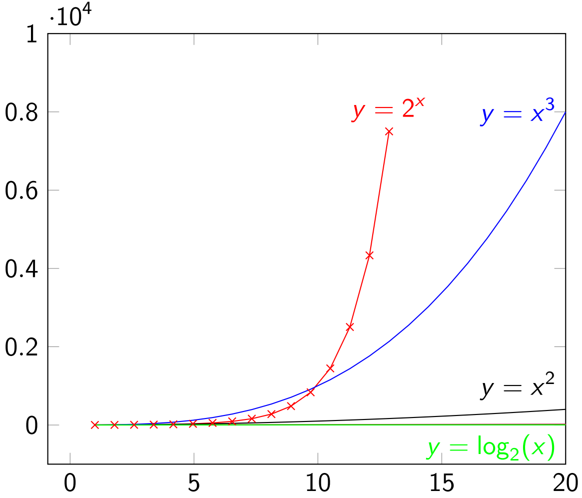
Growth rates of the 7 functions



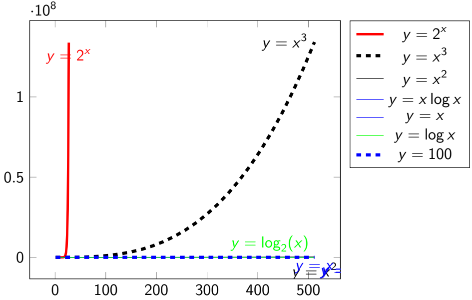
Observations

- Exponential function grows fast
- x is small

Zoom in by limiting $y < 10,000$

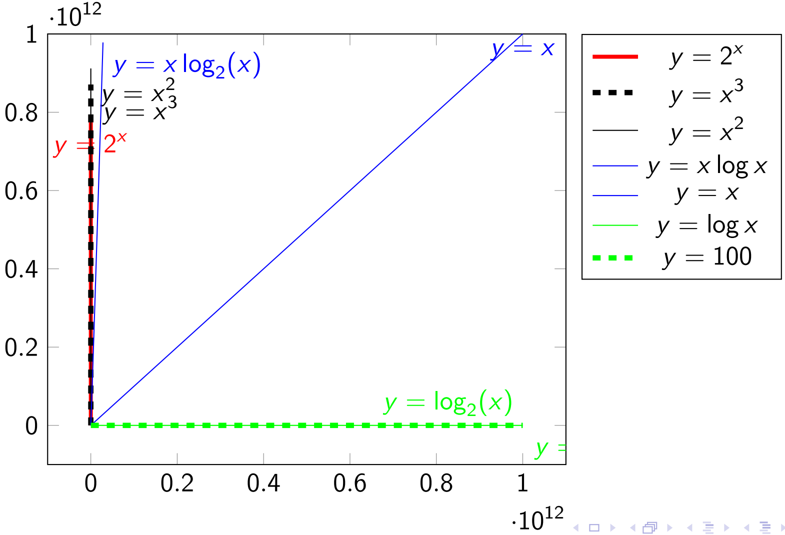


Read the numbers of the chart

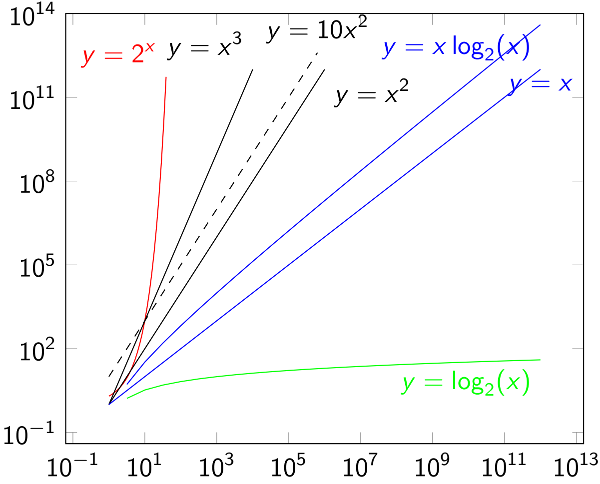


n	$\log n$	n	$n \log n$	n^2	n^3	2^n
8	3	8	24	64	512	256
16	4	16	64	256	4,096	65,536
32	5	32	160	1,024	32,768	4,294,967,296
64	6	64	384	4,096	262,144	1.84×10^{19}
128	7	128	896	16,384	2,097,152	3.40×10^{38}
256	8	256	2,048	65,536	16,777,216	1.15×10^{77}
512	9	512	4,608	262,144	134,217,728	1.34×10^{154}

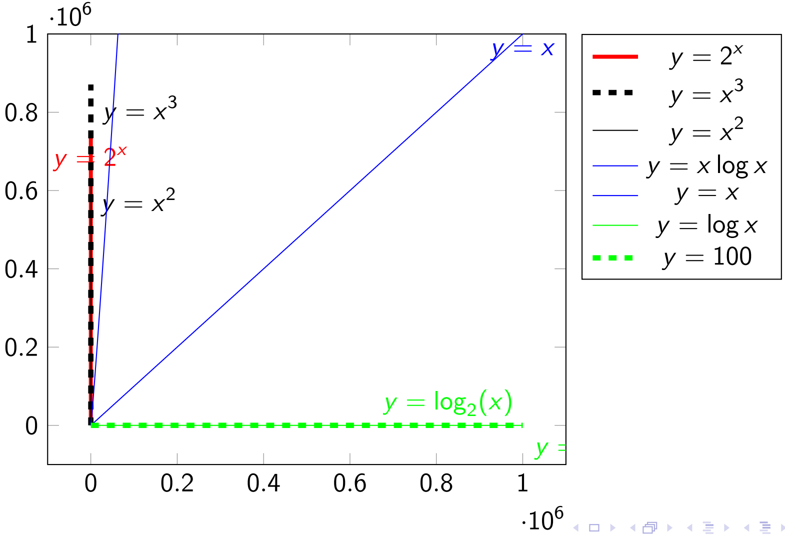
Growth rate of the 7 functions: when x becomes larger



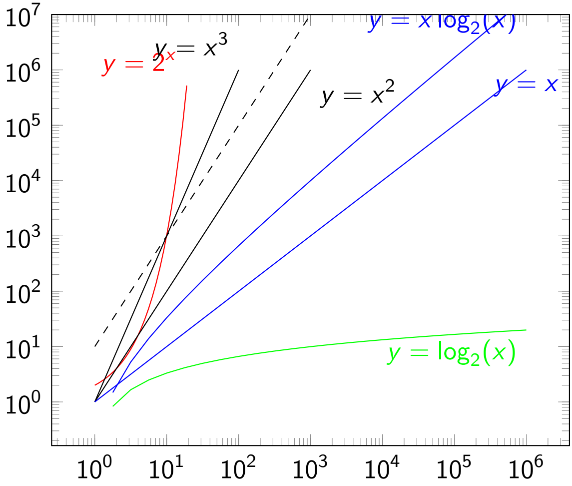
Growth rate of the 7 functions: loglog scale



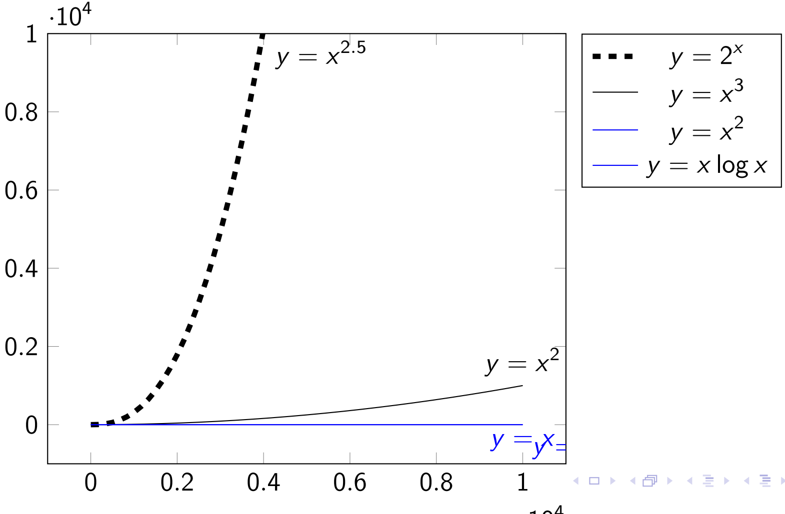
Growth rate of the 7 functions: when x limit is 10^6



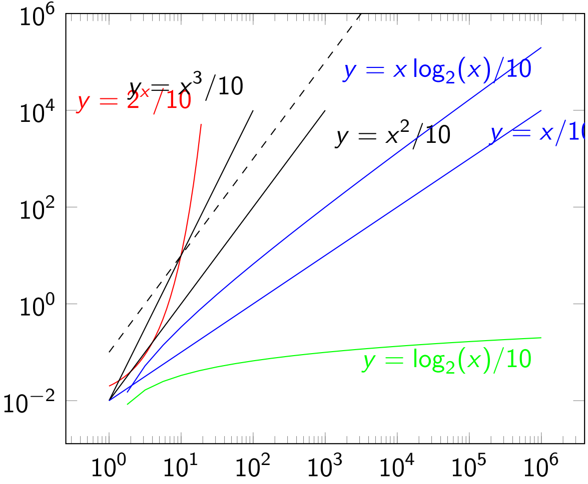
Growth rate of the 7 functions: loglog scale ($x < 10^6$)

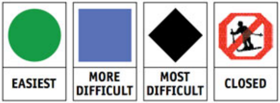


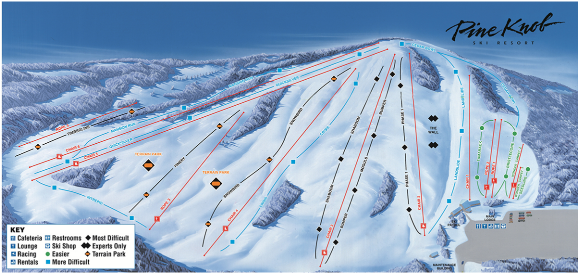
Growth rate of the 7 functions: when x limit is 10^6: x^2. $y/10^5$



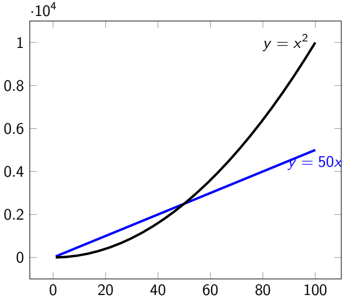
Growth rate of the 7 functions: loglog scale ($x < 10^6$)




EASIEST
MORE DIFFICULT
MOST DIFFICULT
CLOSED

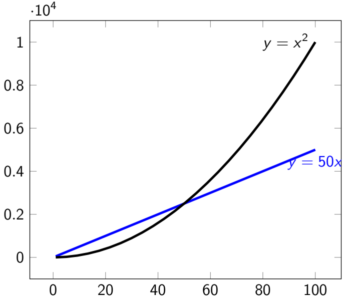

Pine Knob
SKI RESORT
KEY
Cafeteria
Restrooms
Most Difficult
Lounge
Ski Shop
Experts Only
Racing
Easier
Terrain Park
Rentals
More Difficult

Which function is better? The impact of coefficient



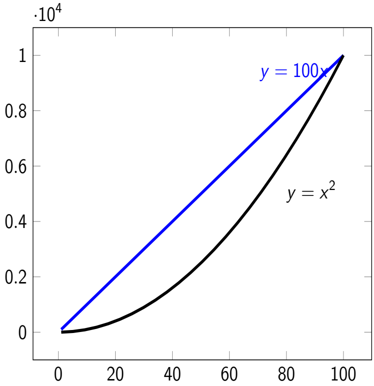
- it is not always one function is smaller than another
- what matters is when x is large

Which function is better? The impact of coefficient

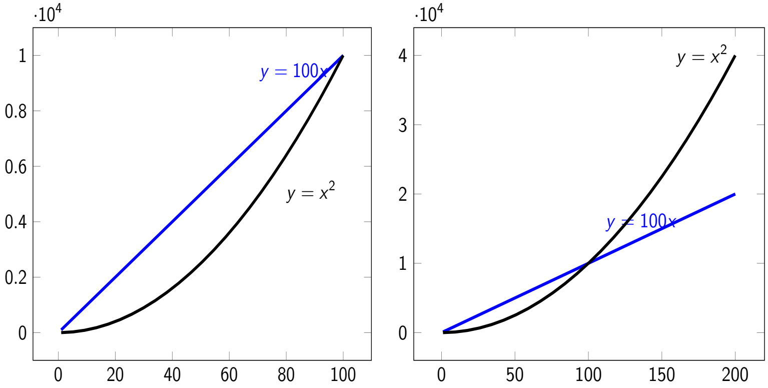


- it is not always one function is smaller than another
- what matters is when x is large
- What if we increase the constant from 50 to 100?

The impact of coefficient

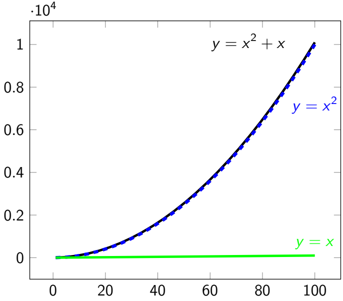


The impact of coefficient



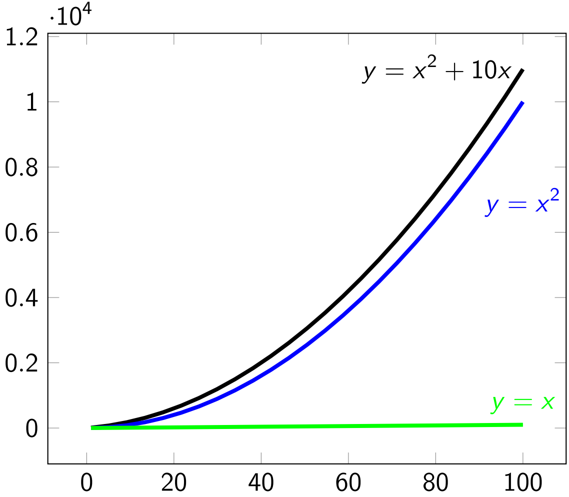
- No matter how big is the constant coefficient, x^2 always grows faster

The impact of lower order terms



- x^2 and $x^2 + x$ are in the same category, when compared with $y = x$.
- What if the coefficient for the lower order term(s) is bigger?

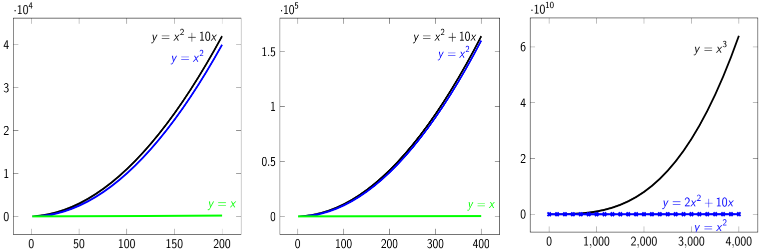
The impact of bigger lower order terms



It seems that the difference is bigger when the coefficient is 10 ...

Impact of coefficient and lower order terms

Not so obvious when x becomes larger...



We want to say that

$$y = x^2 \tag{1}$$

$$y = x^2 + 10x \tag{2}$$

$$y = 2x^2 + 10x \tag{3}$$

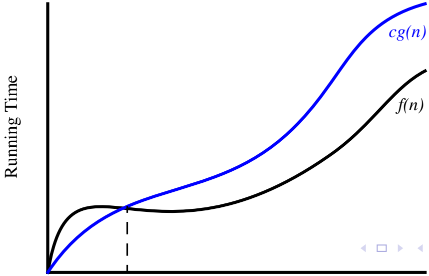
$$\ldots \tag{4}$$

are in the same category....

Big Oh

Definition: O(g(n))

$$O((g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that} \\ 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\} \quad (1)$$



Big O examples

- Prove that $7n - 2$ is $O(n)$
- Need to find $c > 0$ and $n_0 > 1$ such that

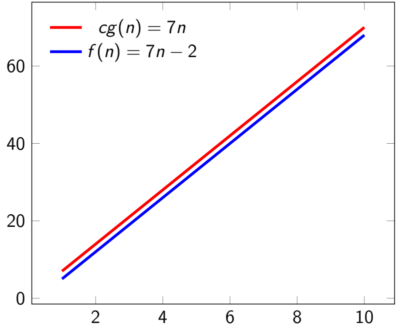
$$7n - 2 \leq cn \qquad (1)$$

for $n > n_0$.

-

$$c \geq 7 - 2/n \qquad (1)$$

- This is true when $c = 7$ and $n_0 = 1$



Prove that $7n + 2$ is $O(n)$

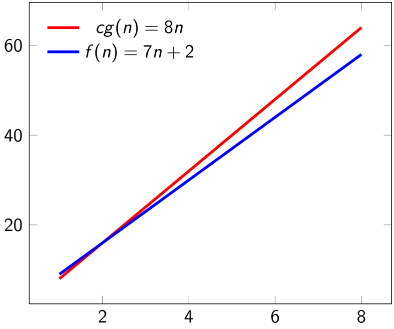
- Need to find $c > 0$ and $n_0 > 1$ such that

$$7n + 2 \leq cn \qquad (1)$$

for $n > n_0$.

$$c \geq 7 + 2/n$$

This is true when $c = 8$ and $n_0 = 2$



Prove that $f(n) = 2n^2 + 5n$ is $O(n^2)$

- Need to find $c > 0$ and $n_0 > 1$ such that

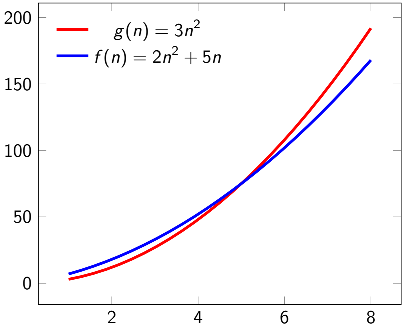
$$2n^2 + 5n \leq cn^2$$

for $n > n_0$.

$$c \geq 2 + 5/n$$

true when $c = 3$ and $n_0 = 5$

- This is the exact solution
- There are many other solutions
-



Prove that $f(n) = 2n^2 + 5n$ is $O(n^2)$

- Need to find $c > 0$ and $n_0 > 1$ such that

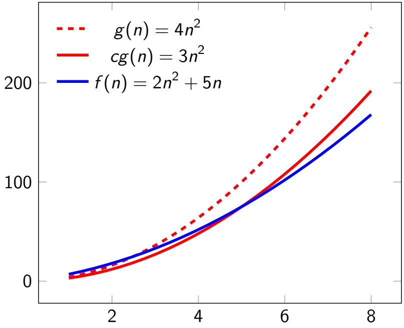
$$2n^2 + 5n \leq cn^2$$

for $n > n_0$.

$$c \geq 2 + 5/n$$

true when $c = 3$ and $n_0 = 5$

- This is the exact solution
- There are many other solutions
- e.g., when c=7, $n_0 = 1$



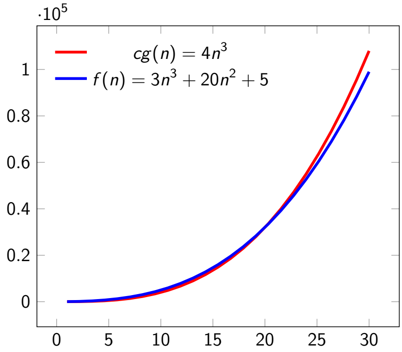
Prove that $3n^3 + 20n^2 + 5$ is $O(n^3)$

- Need to find $c > 0$ and $n_0 > 1$ such that

$$3n^3 + 20n^2 + 5 \leq cn^3$$

$$c \geq 3 + 20/n + 5/n^3$$

This is true when $c = 4$ and $n_0 = 21$



General rules for polynomials

Proposition for Polynomial

If $f(n)$ is a polynomial of degree d, that is,

$$f(n) = a_0 + a_1 n + \cdots + a_d n^d \tag{1}$$

and $a_d > 0$, then $f(n)$ is $O(n^d)$.

Justification: Note that, for $n \geq 1$, we have

$$1 \leq n \leq n^2 \leq \cdots \leq n^d;$$

hence,

$$a_0 + a_1 n + a_2 n^2 + \cdots + a_d n^d \leq (|a_0| + |a_1| + |a_2| + \cdots + |a_d|)n^d.$$

We show that $f(n)$ is $O(n^d)$ by defining

$$c = |a_0| + |a_1| + \cdots + |a_d|$$

and $n_0 = 1$.

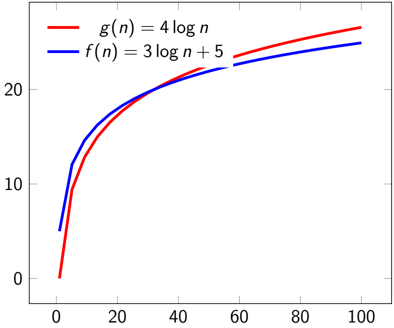
Prove that $3 \log n + 5$ is $O(\log n)$

- Need to find $c > 0$ and $n_0 > 1$ such that

$$3 \log n + 5 \leq c \log n \qquad (1)$$

$$c \geq 3 + 5/ \log n \qquad (1)$$

- This is true when $c = 4$ and $n_0 = 32$



O, Ω, Θ

Big-Oh $f(n)$ is $O(g(n))$ if
- $f(n)$ is asymptotically less than or equal to $g(n)$

big-Omega $f(n)$ is $\Omega(g(n))$ if
- f(n) is asymptotically greater than or equal to g(n)

big-Theta $f(n)$ is $\Theta(g(n))$ if
- f(n) is asymptotically equal to g(n)

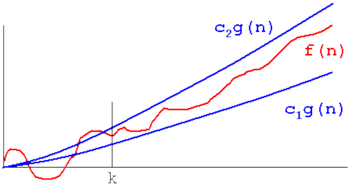
Analogy between real number comparisons

$$f(n) = O(g(n) \approx a \leq b \tag{1}$$
$$f(n) = \Omega(g(n) \approx a \geq b \tag{2}$$
$$f(n) = \Theta(g(n) \approx a = b \tag{3}$$

big-Theta

$f(n)$ is $\Theta(g(n))$ if

- f(n) is asymptotically equal to g(n)



Find max example again

```
arrayMax(int[] data)
  int n=data.length;
  int currentMax=data[0];
  for (int j=1;j<n;j++)
    if(data[j]>currentMax)
      currentMax=data[j];
  return currentMax;
```

A simplified inference

Proposition: arrayMax runs in O(n) time

Justification: Number of comparison operations are:

- Non-loop part: b
- Loop part: (n-1) a
- Total:

$$\begin{aligned} f(n) &= a(n-1)+b \\ &= an-a+b \\ &= an+(b-a) \end{aligned}$$

$$\begin{aligned} cn &\geq an+(b-a) \\ c &\geq a+(b-a)/n \end{aligned}$$

Number of updates of *currentMax*

A more challenging question:

```
static double arrayMax(double[] data)
    int n = data.length;
    double currentMax = data[0];
    for (int j=1; j < n; j++)
      if (data[j] > currentMax)
        currentMax = data[j];
    return currentMax;
```

- how many times the red line is executed?

Number of updates of *currentMax*

A more challenging question:

```
static double arrayMax(double[] data)
    int n = data.length;
    double currentMax = data[0];
    for (int j=1; j < n; j++)
      if (data[j] > currentMax)
        currentMax = data[j];
    return currentMax;
```

- how many times the red line is executed?
- For element data[j], the probability it is greater than all proceeding ones is 1/j

$$\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \sum_{j=1}^{n} \frac{1}{j} \approx \ln n \tag{1}$$

Several useful equations

$$1 + 2 + 3 + \cdots + n = \sum_{i=1}^{n} i = n(n+1)/2 \quad \text{(triangular number)}$$

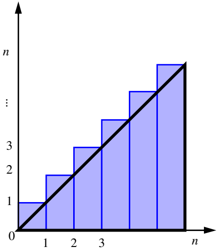
$$1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \sum_{j=1}^{n} \frac{1}{i} \approx \ln n + 0.5 \quad \text{(Hamonic number)}$$

$$2^0 + 2^1 + 2^2 + \cdots + 2^{n-1} = \sum_{i=0}^{n-1} 2^i = 2^n - 1 \quad \text{(Geometric summation)}$$

$$\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \cdots + \frac{1}{2^{n-1}} = \sum_{i=0}^{n-1} \frac{1}{2^i} = 2 - \frac{1}{2^n}$$

Triangular number

$$1 + 2 + 3 + \cdots + n = \sum_{i=1}^{n} i = n(n+1)/2 \qquad \text{(triangular number)}$$



Complexity of Selection Sort

4	6	7	1	2	5	3	n-1
1	6	7	4	2	5	3	n-2
1	2	7	4	6	5	3	n-3
1	2	3	4	6	5	7	..
1	2	3	4	6	5	7	2
1	2	3	4	5	6	7	1
1	2	3	4	5	6	7	0

Proposition

- The complexity of selection sort is $\approx n^2/2$
- Justification: Number of comparisons:

$$1 + 2 + \cdots + (n-2) + (n-1) = n(n-1)/2 \tag{1}$$

Several other examples

- String concatenation
- Three way set disjointness
- Element uniqueness
- Prefix average

String Concatenation

Repeat a char n times

```
public static String repeat1(char c, int n) {
   String answer = "";
   for (int j=0; j < n; j++)
      answer += c;
   return answer;
}
```

- What is the time complexity?

Three Way Set Disjointness

Check whether

$$A \cap B \cap C = \emptyset \tag{1}$$

```
boolean disjoint1(int[ ] groupA, int[ ] groupB, int[ ] groupC) {
   for (int a : groupA)
     for (int b : groupB)
        for (int c : groupC)
            if ((a == b) && (b == c))
                return false;
    return true;
}
```

- Time complexity?

Three Way Set Disjointness

```
boolean disjoint2(int[ ] groupA, int[ ] groupB, int[ ] groupC) {
  for (int a : groupA)
      for (int b : groupB)
         if (a == b)
            for (int c : groupC)
                if (a == c)
                   return false;
   return true;
}
```

Element Uniqueness

Returns true if there are no duplicate elements in the array.

```
public static boolean unique1(int[ ] data) {
   int n = data.length;
   for (int j=0; j<n-1;j++)
       for (int k=j+1; k < n; k++)
          if (data[j] == data[k])
              return false;
    return true;
```

Element Uniqueness (Good one)

```
public static boolean unique2(int[ ] data) {
  int n = data.length;
  int[ ] temp = Arrays.copyOf(data, n);
  Arrays.sort(temp);
  for (int j=0; j<n-1; j++)
    if (temp[j] == temp[j+1])
        return false;
  return true;
 }
```

Prefix Average

```
public static double[ ] prefixAverage1(double[ ] x) {
   int n = x.length;
   double[ ] a = new double[n];
   for (int j=0; j < n; j++) {
      double total = 0;
      for (int i=0; i <= j; i++)
          total += x[i];
      a[j] = total / (j+1);
   }
   return a;
}
```

Prefix Average (good one)

```
public static double[ ] prefixAverage2(double[ ] x) {
   int n = x.length;
   double[ ] a = new double[n];
   double total = 0;
   for (int j=0; j < n; j++) {
      total += x[j];
      a[j] = total / (j+1);
    }
    return a;
  }
```

Takeaways

- Why algorithm analysis (why empirical experiments are not enough)
- We count primitive operations
- We group growth rate into 7 functions
- Upper and lower bounds (Big Oh, Big Omega, and Big Theta)
- This is just the beginning ...
-
- Readings: Goodrich P151-P177