# Algorithm Analysis

Class 4

# Introduction

- in general, the amount of time a program takes to run is proportional to the amount of input it must process
- the more input, the more time it takes
- we care how much time a program takes to run because time is money
- any program will run quickly with small input size
- so the real question is:

**Scaling**

How does the time taken by a program vary, or **scale**, as the amount of input grows?

# Program vs Algorithm

- actually timing a program with a stopwatch has little value
- the number of seconds depends on
  - the language used
  - the speed and architecture of the computer it runs on
  - how heavily loaded the computer is with other jobs
- instead we wish to analyze the algorithm in a way that is
  - language-neutral
  - independent of hardware
  - independent of OS and load
- we will use theory for analysis
- we will sometimes confirm the theory using empirical tests

# What is the Input Size?

- what is the size of the input that affects the algorithm?
- for array summation, it is the number of array elements
- for array sorting, ditto
- for computing $n!$ $n$ is not the amount of input, $n$ is the input
    - clearly the bigger $n$ is, the longer it takes to compute $n!$

# What is the Input Size?

- what is the size of the input that affects the algorithm?
- for array summation, it is the number of array elements
- for array sorting, ditto
- for computing $n!$ $n$ is not the amount of input, $n$ is the input
  - clearly the bigger $n$ is, the longer it takes to compute $n!$
  - in this case, the input size is the number of bits required to encode $n$, i.e., the magnitude of $n$, $|n|$

# What is the Input Size?

- what is the size of the input that affects the algorithm?
- for array summation, it is the number of array elements
- for array sorting, ditto
- for computing $n!$ $n$ is not the amount of input, $n$ is the input
  - clearly the bigger $n$ is, the longer it takes to compute $n!$
  - in this case, the input size is the number of bits required to encode $n$, i.e., the magnitude of $n$, $|n|$
- for many graph algorithms, input size is both the number of nodes $n$ and the number of edges $m$: the input size is a tuple

# Input Arrangement

- sometimes the arrangement of input matters
- for some algorithms, the time taken varies depending on the particular arrangement of the input
- consider an unordered array of $n$ integers and an algorithm to find the first even value
  - what is the best case?
  - what is the worst case?

- an algorithm to sum array elements does not have a best or worst case — it does not depend on the arrangement
- if an algorithm varies depending on the particular arrangement of input, this must be stated

# What to Count

- to analyze an algorithm we must count something
- to physically time a program we count seconds
- to analyze an algorithm we count basic operations

- basic operations typically correspond to single program-language statements or operations
  - arithmetic ($+ \; - \; \times \; \div$)
  - scalar assignment
  - scalar comparison

# What to Count

- to analyze an algorithm we must count something
- to physically time a program we count seconds
- to analyze an algorithm we count basic operations

- basic operations typically correspond to single program-language statements or operations
    - arithmetic $(+ \; - \; \times \; \div)$
    - scalar assignment
    - scalar comparison

- we cannot count a statement that implies a complex algorithm or a loop, e.g., sort or find or exponentiation
- sort is not a basic operation

# What Not to Count

- we will not count:
  - a return statement
    (but determining what to return is counted)
  - a recursive call
    (but computing the arguments of recursive calls is counted)

- if a function does a simple, constant amount of work we can count its operations directly
- if a function is complex, we cannot count it directly but must drill inside and count its basic operations as they occur

## Function: input size $\longrightarrow$ operations

- the analysis of an algorithm is the determination of the number of basic operations performed as a function of input size
- we always state an analysis in terms of standard functions
- independent variable (horizontal axis) is always the input size, always a non-negative (unsigned) integer, denoted $n$
- dependent variable (vertical axis) is always a non-negative count of basic operations, denoted $T(n)$

- what are the standard functions?

# Standard Functions

$f(n) = 1$ constant: the count of basic operations does not vary with the size of input

$f(n) = \log n$ logarithmic: the count grows more slowly than the size of input

$f(n) = n$ linear: the count grows at the same rate as the size of input

$f(n) = n \log n$ n-log-n: the count grows somewhat faster than the size of input (very good performance for a program)

$f(n) = n^2$ quadratic: the count grows as the square of the input size (very common)

$f(n) = n^k$ polynomial: the count grows to the exponent $k$

$f(n) = C^n$ exponential; C a constant: very rapid growth

$f(n) = n!$ factorial: even more rapid growth

$f(n) = n^n$ bad exponential: even more rapid growth (Beck's term; rare, but does occur)

## Example

- consider the following two functions

$$T(n) = \frac{n^2}{10} + n + 1 \tag{1}$$

$$S(n) = 100n + 1 \tag{2}$$

- which one is "bigger" (i.e., has larger $y$-axis value)?
- which one grows faster?
- investigate with gnuplot

# Example

- consider the following two functions

$$T(n) = \frac{n^2}{10} + n + 1 \quad (1)$$

$$S(n) = 100n + 1 \quad (2)$$

- which one is "bigger" (i.e., has larger $y$-axis value)?
- which one grows faster?
- investigate with gnuplot
- gnuplot illustrates two points
  - we are not interested in behavior at small values of input size only the asymptotic behavior as input size gets very large
  - the high-order term of a polynomial always dominates for sufficiently large input size

# Big-$O$

- a definition you must know

**Definition 1: Big-$O$**

$T(n) \in O(f(n))$ if there are positive constants real $c$ and integer $n_0$ such that $T(n) \leq cf(n)$ when $n \geq n_0$

# Notation

Warning! You will often see the notation

$$T(n) = O(\text{foo})$$

this is wrong because $T(n)$ is a single thing and $O(\text{foo})$ is a set

instead we use the correct set notation

$$T(n) \in O(\text{foo})$$

Warning! You will often hear the phrase "T of n is of order foo". This is old-fashioned, deprecated language.

Use "big-Oh"; do not use the term "order" for this

# Big-Omega

**Definition 2: Big-$\Omega$**

$T(n) \in \Omega(f(n))$ if there are positive constants real $c$ and integer $n_0$ such that $T(n) \geq cf(n)$ when $n \geq n_0$

## Big-Theta

**Definition 3: Big-$\Theta$**

$T(n) \in \Theta(f(n))$ iff $T(n) \in O(f(n))$ and $T(n) \in \Omega(f(n))$

- the $c$ and $n_0$ for the big-Oh and for the big-Omega are distinct
- it makes life easier if we choose both $n_0$ to be the same

# Big-O

$T(n) \in O(f(n))$ if there are positive constant $c$ and nonnegative integer $n_0$ such that $T(n) \leq cf(n)$ when $n \geq n_0$

- $O(f(n))$ is a set of functions defined by $f(n)$
- let $f(n) = n^2$ for example
- what functions are in $O(f(n))$?
  - $n^2, 2n^2, \frac{n^2}{2}, n^2 + 10n - 5$, all other polynomial functions of degree 2
  - $n, 4n, \frac{n}{2}, 10^5 n + 100$, all other polynomial functions of degree less than 2
  - $\log n, 10 \lg n, n \lg n$, all other functions whose highest-order term is less than or equal to $n^2$

$O(n^2)$

$3 \lg n$

$n^2 + n \lg n$

$5n$

$5n^2 + 3$

$2n \lg n$

$\frac{n^2}{2}$

$1$

$n^2 + 10n - 5$

# Big-Omega

$T(n) \in \Omega(f(n))$ if there are positive constant $c$ and nonnegative integer $n_0$ such that $T(n) \geq cf(n)$ when $n \geq n_0$

- $\Omega(f(n))$ is a set of functions defined by $f(n)$
- let $f(n) = n^2$ for example
- what functions are in $\Omega(f(n))$?
  - $n^2, 2n^2, \frac{n^2}{2}, n^2 + 10n - 5$, all other polynomial functions of degree 2
  - $n^3, 4n^5, \frac{n^4}{2}, n^{4.5} - 1000$, all other polynomial functions of degree greater than 2
  - $n^3 \log n$, all other functions whose high-order term is greater than or equal to $n^2$

$\Omega(n^2)$

$n^2 + n \lg n$

$n^3 \lg n$

$5n^2 + 3$

$4n^5$

$\frac{n^2}{2}$

$\frac{n^4}{2}$

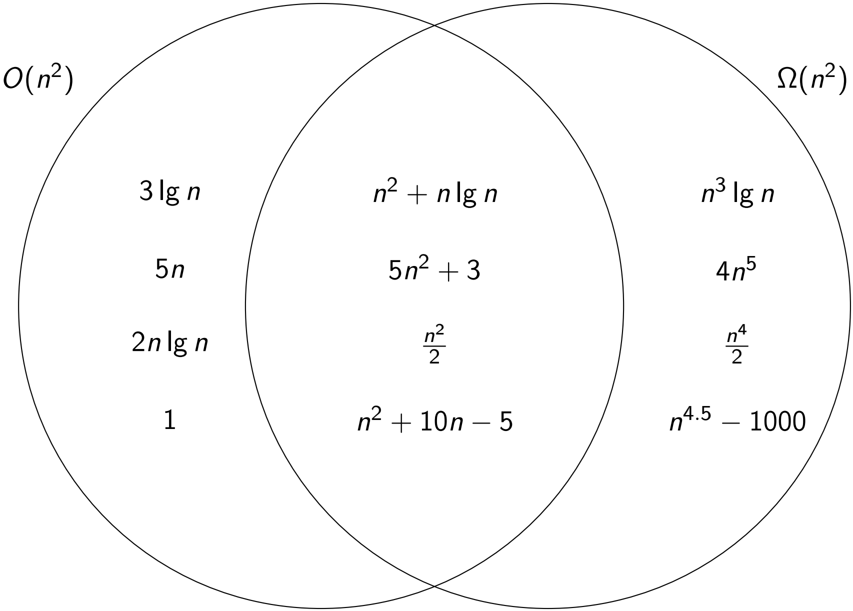
$n^2 + 10n - 5$

$n^{4.5} - 1000$

# Big-Theta

$$T(n) \in \Theta(f(n)) \text{ iff } T(n) \in O(f(n)) \cap \Omega(f(n))$$

- $\Theta(f(n))$ is a set of functions defined by $f(n)$
- let $f(n) = n^2$ for example
- what functions are in $\Theta(n^2)$?
- the functions in the intersection of the sets $O(f(n))$ and $\Omega(f(n))$
- exactly the functions whose high-order term is a non-zero multiple of $n^2$


$O(n^2)$
$\Omega(n^2)$
$3 \lg n$
$5n$
$2n \lg n$
$1$
$n^2 + n \lg n$
$5n^2 + 3$
$\frac{n^2}{2}$
$n^2 + 10n - 5$
$n^3 \lg n$
$4n^5$
$\frac{n^4}{2}$
$n^{4.5} - 1000$