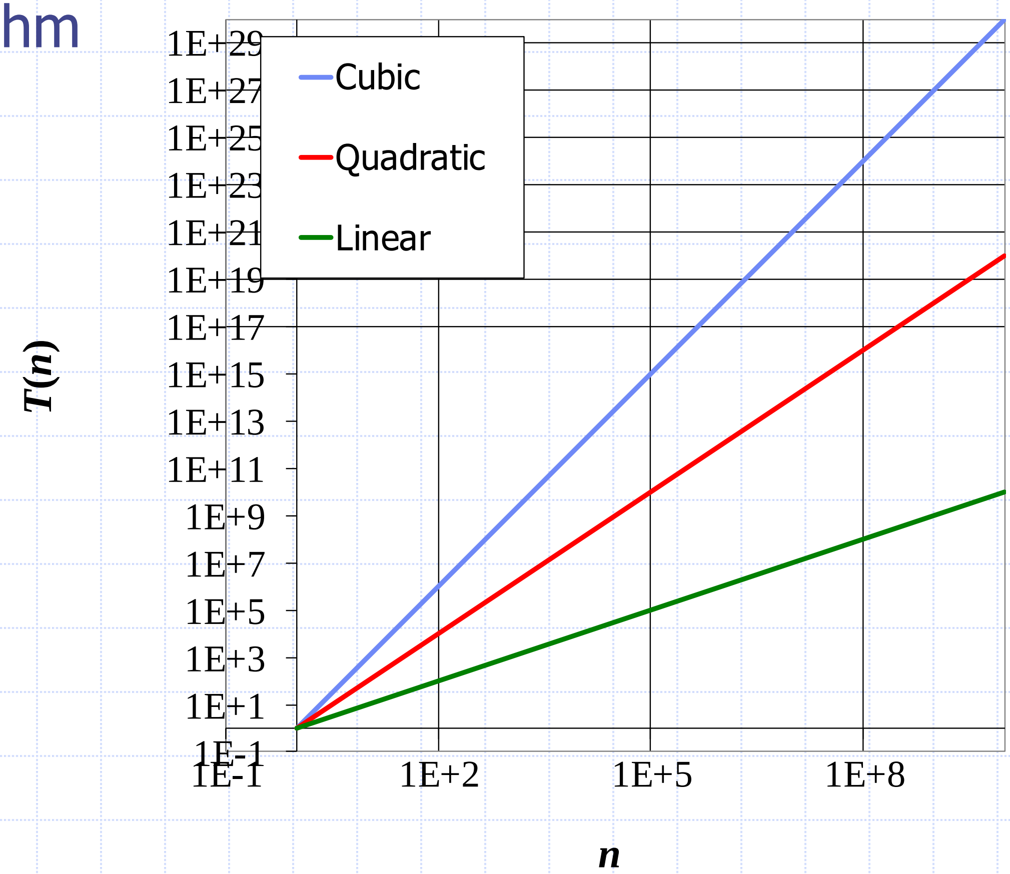
AE2ADS: Algorithms Data Structures and Efficiency

Lecturer: Heshan Du
Email: Heshan.Du@nottingham.edu.cn

University of Nottingham Ningbo China

Seven Important Functions

- Seven functions that often appear in algorithm analysis:
  - Constant ≈ 1
  - Logarithmic ≈ $\log n$
  - Linear ≈ n
  - N-Log-N ≈ $n \log n$
  - Quadratic ≈ n^2
  - Cubic ≈ n^3
  - Exponential ≈ 2^n
- In a log-log chart, the slope of the line corresponds to the growth rate



Seven Common Functions

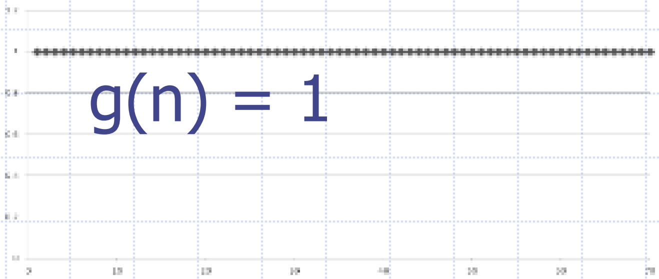
- Ideally, we would like data structure operations to run in times proportional to the constant or logarithm function
- We would like our algorithms to run in linear or n-log-n time.

Seven Common Functions

- Algorithms with quadratic or cubic running times are less practical.
- Algorithms with exponential running times are infeasible for all but the smallest sized inputs.

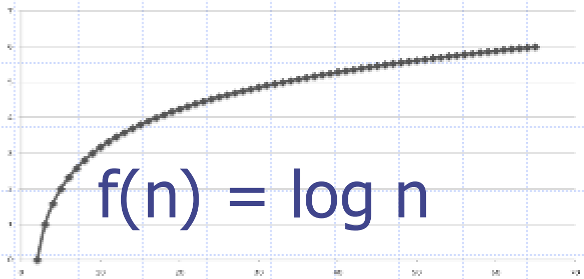
The Constant Function

- $f(n) = c,$
- where c is a fixed constant.
- e.g., $g(n) = 1$



The Logarithm Function

- $f(n) = \log_b n$,
- where b is a constant and b >1.
- $x = \log_b n$ iff $b^x = n$. The value b is known as the base of the logarithm.
- Convention in CS: $\log n = \log_2 n$



Review: Logarithm Rules

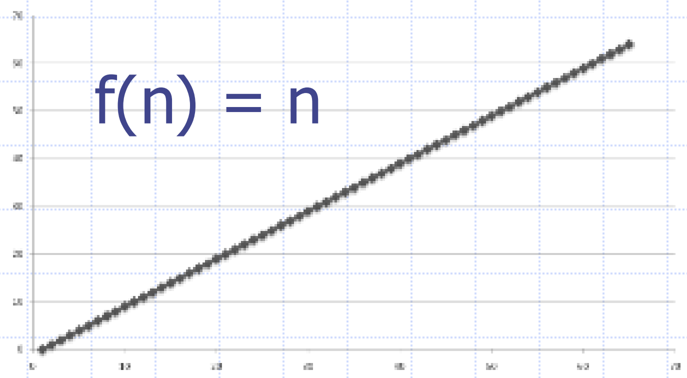
- Given real numbers $a > 0$, $b > 1$, $c > 0$, and $d > 1$, we have:
- $\log_b(ac) = ?$
- $\log_b(a/c) = ?$
- $\log_b(a^c) = ?$
- $\log_b a = ?$
- $b^{\log_d a} = ?$

Review: Logarithm Rules

- Given real numbers $a > 0$, $b > 1$, $c > 0$, and $d > 1$, we have:
- $\log_b(ac) = \log_b a + \log_b c$
- $\log_b(a/c) = \log_b a - \log_b c$
- $\log_b(a^c) = \mathrm{c} \log_b a$
- $\log_b a = \log_d a \,/\, \log_d b$
- $b^{\log_d a} = a^{\log_d b}$

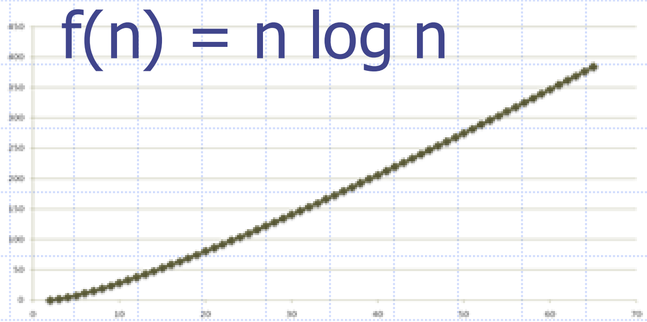
The linear Function

- $f(n) = n$,
- where n is a non-negative integer.



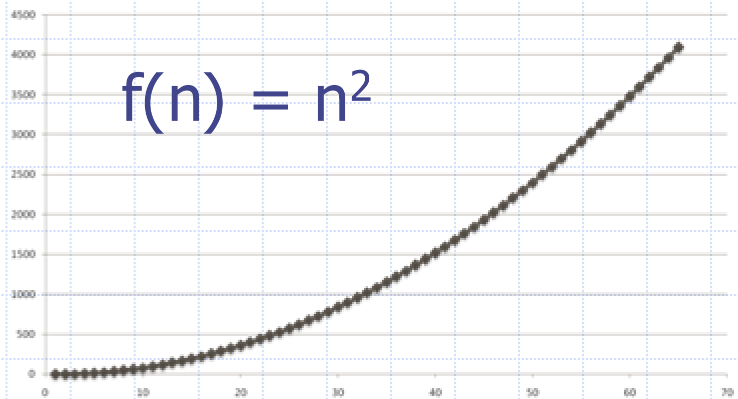
The N-Log-N Function

- $f(n) = n \log n$
- where n is a non-negative integer.



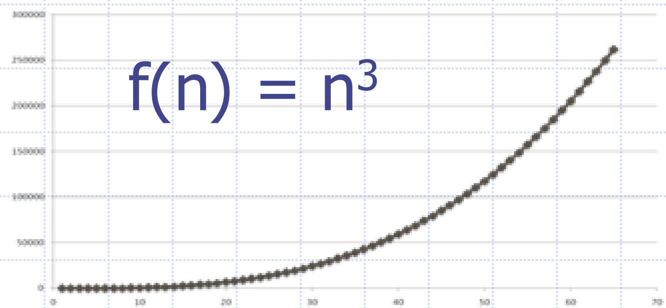
The Quadratic Function

- $f(n) = n^2$
- where n is a non-negative integer.



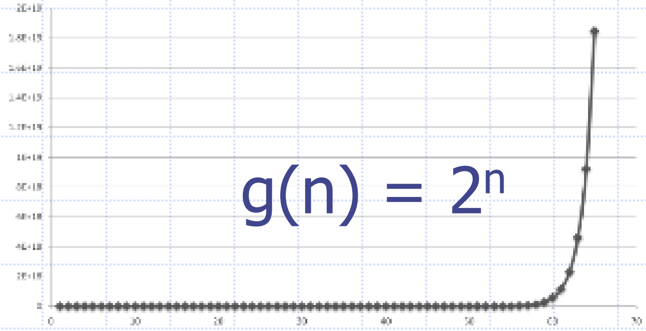
The Cubic Function

- $f(n) = n^3$
- where n is a non-negative integer.



The Exponential Function

- $f(n) = b^n$
- Where b is a positive constant, called the **base**, and the argument n is the **exponent**.
- e.g., $g(n) = 2^n$



Review: Exponent Rules

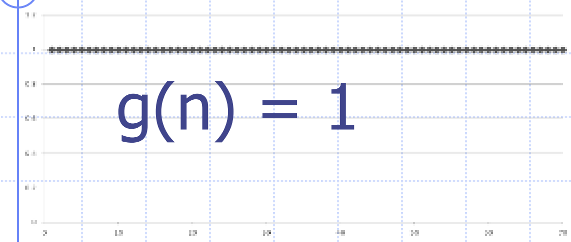
- Given positive integer a, b and c, we have
- $(b^a)^c = ?$
- $b^a b^c = ?$
- $b^a / b^c = ?$

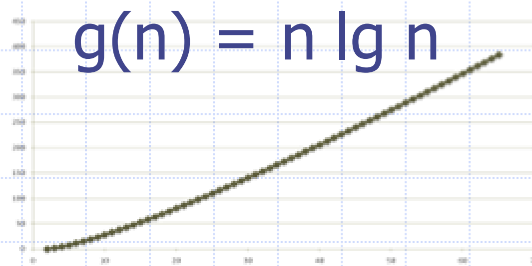
Review: Exponent Rules

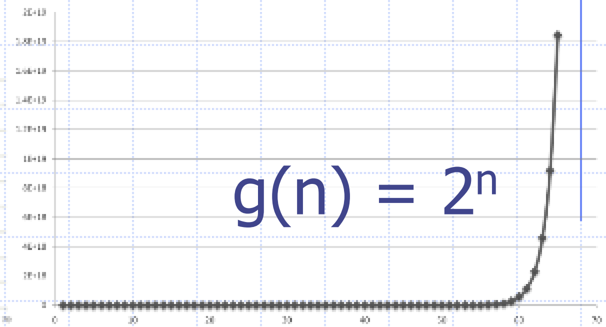
- Given positive integer a, b and c, we have
- $(b^a)^c = b^{ac}$
- $b^a b^c = b^{a+c}$
- $b^a / b^c = b^{a-c}$

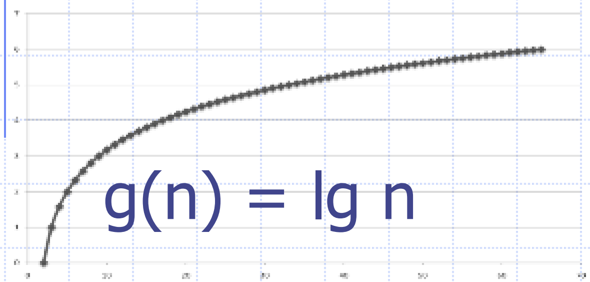
Functions Graphed Using “Normal” Scale

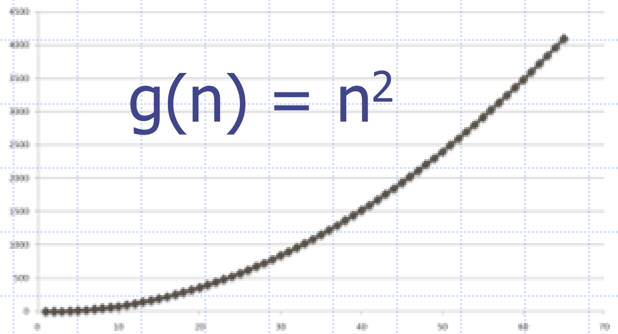
Slide by Matt Stallmann included with permission.

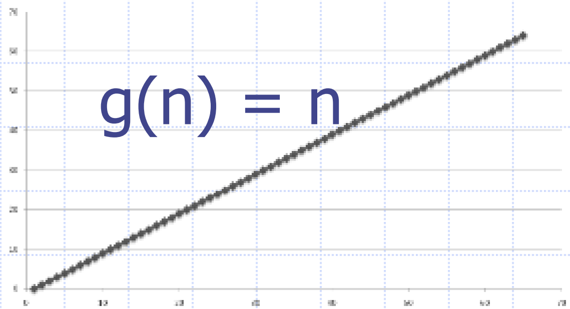


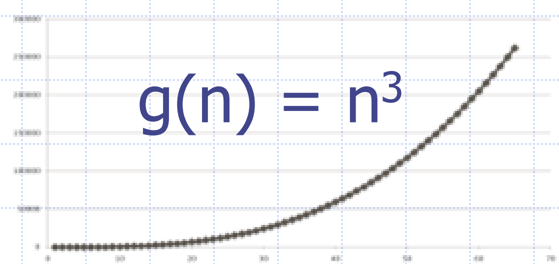












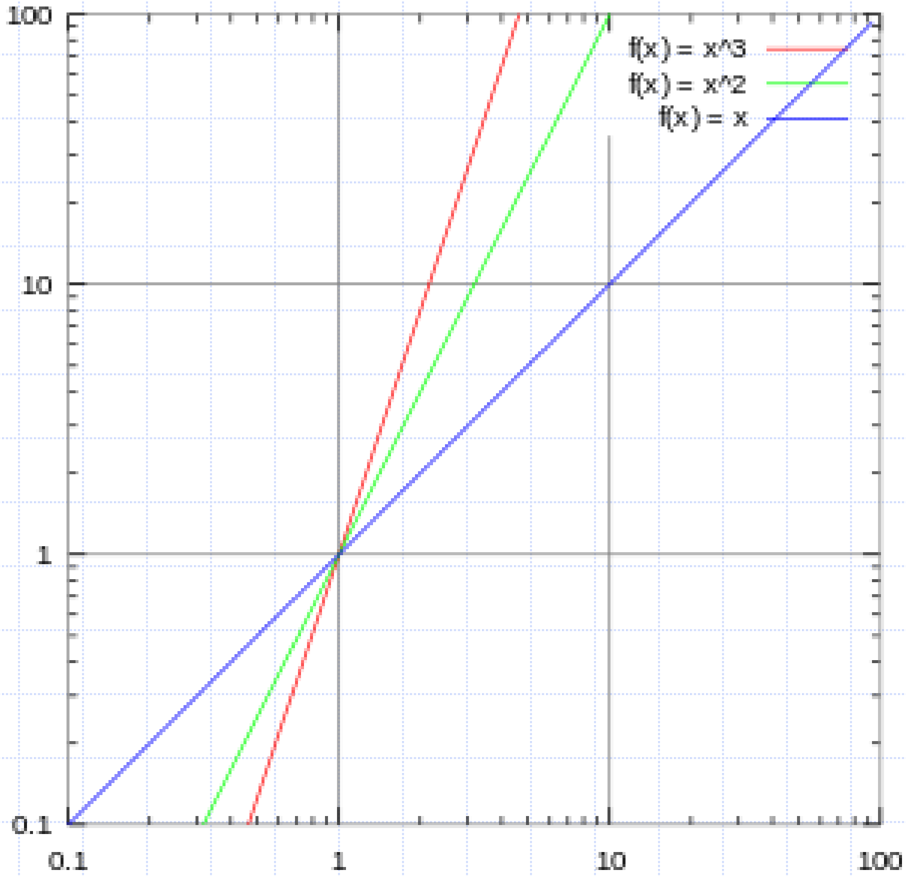
Log-log Plot

- A log–log plot is a two-dimensional graph of numerical data that uses logarithmic scales on both the horizontal and vertical axes.
- Relationships of the form $y = ax^k$ appear as straight lines in a log–log graph, with the power and constant term corresponding to slope and intercept of the line.
- Any base can be used for the logarithm, though most common are 10, e, and 2.

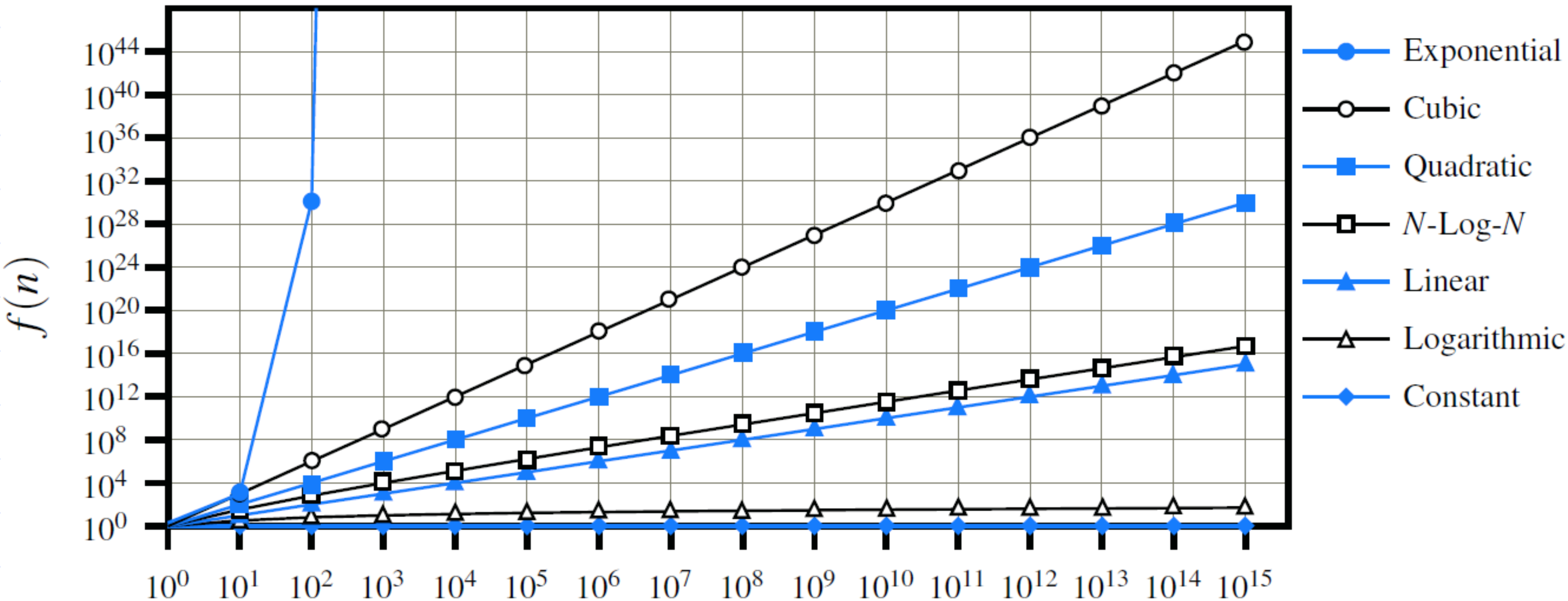
Log-log Plot

- $y = ax^k$
- taking the logarithm of the equation (with any base)
- $\log y = k \log x + \log a$
- Let $X = \log x$ and $Y = \log y$. $Y = kX + b$.
- k is the slope of the line (gradient).

Log-log Plot



Seven Common Functions

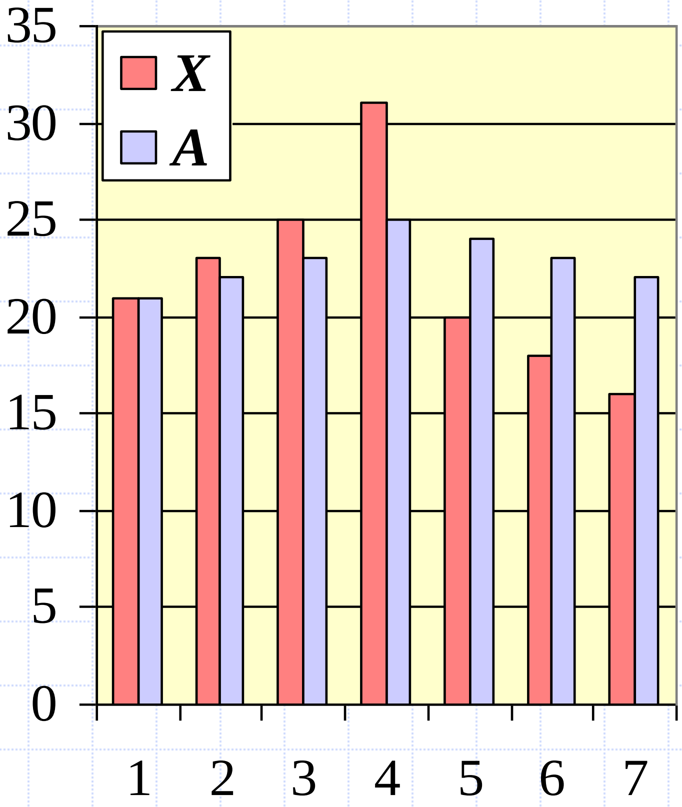


Exercise: Computing Prefix Averages

- Write pseudocode for an algorithm which computes prefix averages.
- The i-th prefix average of an array X is average of the first $(i + 1)$ elements of X:

$$A[i] = (X[0] + X[1] + \ldots + X[i])/(i+1)$$

- Computing the array A of prefix averages of another array X has applications to financial analysis



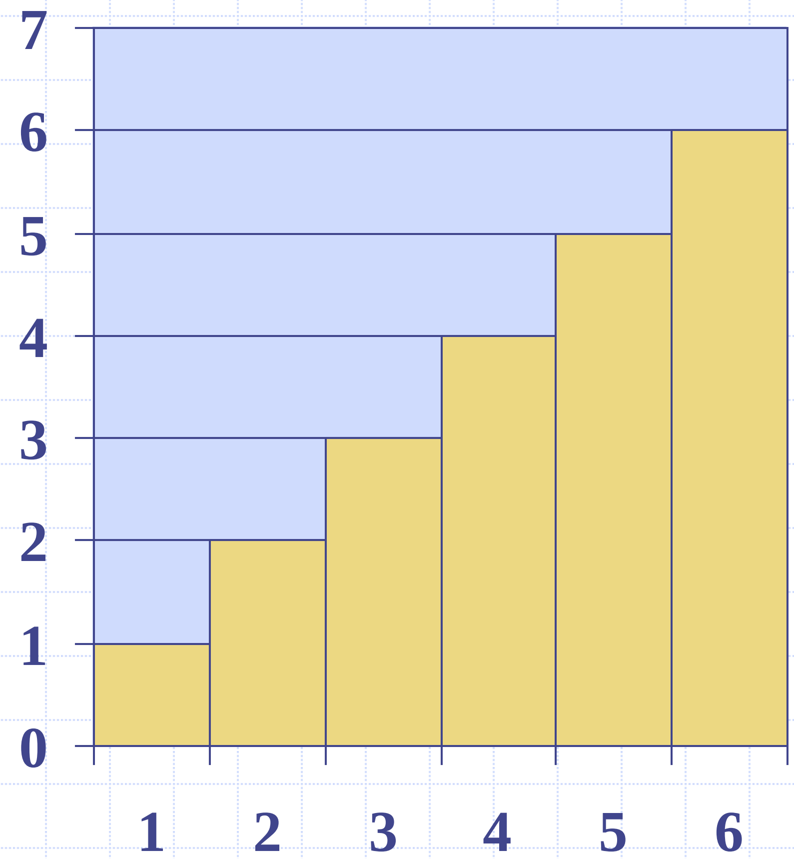
Prefix Averages (Quadratic)

The following algorithm computes prefix averages in quadratic time by applying the definition

```
/** Returns an array a such that, for all j, a[j] equals the average of x[0], ..., x[j]. */
public static double[ ] prefixAverage1(double[ ] x) {
  int n = x.length;
  double[ ] a = new double[n];                    // filled with zeros by default
  for (int j=0; j < n; j++) {
    double total = 0;                             // begin computing x[0] + ... + x[j]
    for (int i=0; i <= j; i++)
      total += x[i];
    a[j] = total / (j+1);                         // record the average
  }
  return a;
}
```

Arithmetic Progression

- The running time of prefixAverage1 is $O(1 + 2 + \ldots + n)$
- The sum of the first n integers is $n(n + 1) / 2$
  - There is a simple visual proof of this fact
- Thus, algorithm prefixAverage1 runs in $O(n^2)$ time



Prefix Averages 2 (Linear)

The following algorithm uses a running summation to improve the efficiency

```
/** Returns an array a such that, for all j, a[j] equals the average of x[0], ..., x[j]. */
public static double[ ] prefixAverage2(double[ ] x) {
  int n = x.length;
  double[ ] a = new double[n];            // filled with zeros by default
  double total = 0;                       // compute prefix sum as x[0] + x[1] + ...
  for (int j=0; j < n; j++) {
    total += x[j];                        // update prefix sum to include x[j]
    a[j] = total / (j+1);                 // compute average based on current sum
  }
  return a;
}
```

Algorithm prefixAverage2 runs in $O(n)$ time!

Pseudocode for Prefix Average

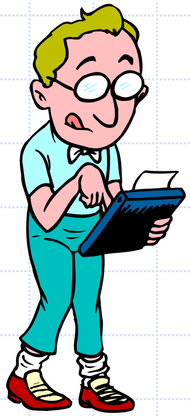
Algorithm: prefixAverage

Input: an array X of n numbers.

Output: an array A of size n such that for every integer i in $[0,\ n-1]$, $A[i] = (X[0] + X[1] + \cdots + X[i])/(i+1)$.

> Let A be an array of size n
>
> $total \leftarrow 0$
>
> **for** $i \leftarrow 0$ to $n-1$ **do**
>
> > $total \leftarrow total + X[i]$
> >
> > A$[i] \leftarrow total/(i+1)$
>
> **end for**
>
> **return** A

Math you need to Review



- Summations
- Powers
- Logarithms
- Proof techniques
- Basic probability

- Properties of powers:
  $a^{(b+c)} = a^b a^c$
  $a^{bc} = (a^b)^c$
  $a^b / a^c = a^{(b-c)}$
  $b = a^{\log_a b}$
  $b^c = a^{c*\log_a b}$
- Properties of logarithms:
  $\log_b(xy) = \log_b x + \log_b y$
  $\log_b (x/y) = \log_b x - \log_b y$
  $\log_b x^a = a\log_b x$
  $\log_b a = \log_x a / \log_x b$