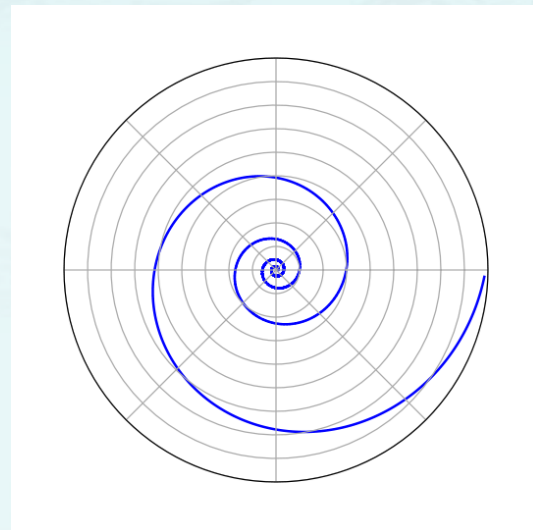


Discrete math

Data Structures
C++ for C Coders

한동대학교 김영섭 교수
idebtor@gmail.com



- **Logarithmic spiral**
- **miraculous spiral**
- ***Spira mirabilis* [Latin]**

1. Logarithms

- **Exponents:**

- X^Y , or "**X to the Yth power**";
X multiplied by itself Y times

- **Some useful identities:**

- $X^0 = 1$, provided $x \neq 0$.
- $X^A X^B = X^{A+B}$
- $X^A / X^B = X^{A-B}$
- $X^{-B} = \frac{1}{X^B}$
- $X^{1/n} = \sqrt[n]{X}$
- $X^N + X^N = 2X^N$
- $2^N + 2^N = 2^{N+1}$

1. Logarithms

- **Logarithms**

- **definition:** $X^A = B$ if and only if $\log_x B = A$
- **intuition:** $\log_x B$ means: the power X must be raised to, to get B
- This is the same as asserting $X^{\log_x B} = B$.

- **Examples:**

- $\log_x 1 = 0$
- $\log_x X = 1$
- $\log_2 16 = 4$
- $\log_{10} 1000 = 3$

- Most people use base 10, written as \log_{10} or \log , base e or \ln .
- In computer science, we typically use base 2, written as \log_2 or lg .
Between us, however, we simply use \log instead of \log_2 or lg .

1. Logarithms

Exercise:

- How many bits does it take to encode 1,000,000 different values?
 - Each bit can take on one of two values (0 or 1).
 - Therefore, n bits can represent 2^n values. (ex. 8bits: 0~255)
 - So, encoding 1,000,000 values will require $\log_2 n = 20$ bits.
To be exact, $\lceil \log_2(1,000,000) \rceil = 20$. // “a little under 20”
- $\lceil x \rceil \rightarrow$ **Ceiling** function: the smallest integer $\geq x$.
 - Ex.
- $\lfloor x \rfloor \rightarrow$ **Floor** function: the largest integer $\leq x$.
 - Ex.

$$\lceil 2.3 \rceil = 3 \quad \lceil -2.3 \rceil = -2 \quad \lceil 2 \rceil = 2$$

$$\lfloor 2.7 \rfloor = 2 \quad \lfloor -2.7 \rfloor = -3 \quad \lfloor 2 \rfloor = 2$$

1. Logarithms

Powers of 2:

- A bit is 0 or 1 (just two different “letters” or “symbols”)
- A sequence of n bits can represent 2^n distinct things
 - For example, the numbers 0 through 2^n-1
- 2^{10} is 1024 (“about a thousand”, kilo in CSE speak)
- 2^{20} is “about a million”, mega in CSE speak
- 2^{30} is “about a billion”, giga in CSE speak

Java: an **int** is 32 bits and signed, so “max int” is “about 2 billion”

a **long** is 64 bits and signed, so “max long” is $2^{63}-1$

1. Logarithms

Examples:

- If we have an alphabetically **sorted list of 100 names**, how many records do we need to look at to find a given individual?
 - Since the list is sorted, we can use **binary search**.
 - Look at the middle element: if it's after than the name we're looking for, search the first half of the list. If it's before the name we're looking for, look at the second half of the list.
 - Each check cuts the size of the list in half; how many times can we do this?

-
-

1. Logarithms

Examples:

- Let's suppose that we begin with a value N , divide it by 2, then the result that we divide it by 2, and so on, until reaching 1 or less.
 - $N, N/2, N/4, \dots, 4, 2, 1$
- **Question:** How many times did we divide before reaching 1 or less?
 - Think of it from the other direction: How many times do I have to multiply by 2 to reach N ?
 - $1, 2, 4, \dots, N/4, N/2, N$
Call this k number of times, then $N = 2^k$, or $k = \lg(N)$.
- **Exercise:** How is this related to the idea of binary search?
 - (I leave this one for you to think about it, as an exercise.)

1. Logarithms

Logarithmic Operators:

- $\log ab = \log a + \log b$
 - $\log \frac{a}{b} = \log a - \log b$
 - $\log a^b = b \log a$
 - $\log_a n = \frac{\log_b n}{\log_b a} = \frac{\log n}{\log a}$
(this is used to change bases.)
 - $\log_a a = 1, \text{ for all } a > 0$
 - $\log_a 1 = 0, \text{ for all } a > 0$
-
- Evaluate $\log_4 4 + \log_2 2 + \log_{10} 1$
 - Evaluate $\log_2 \frac{1}{2}$
 - Plot $y = \log_2 x$

Example: Solve for x.

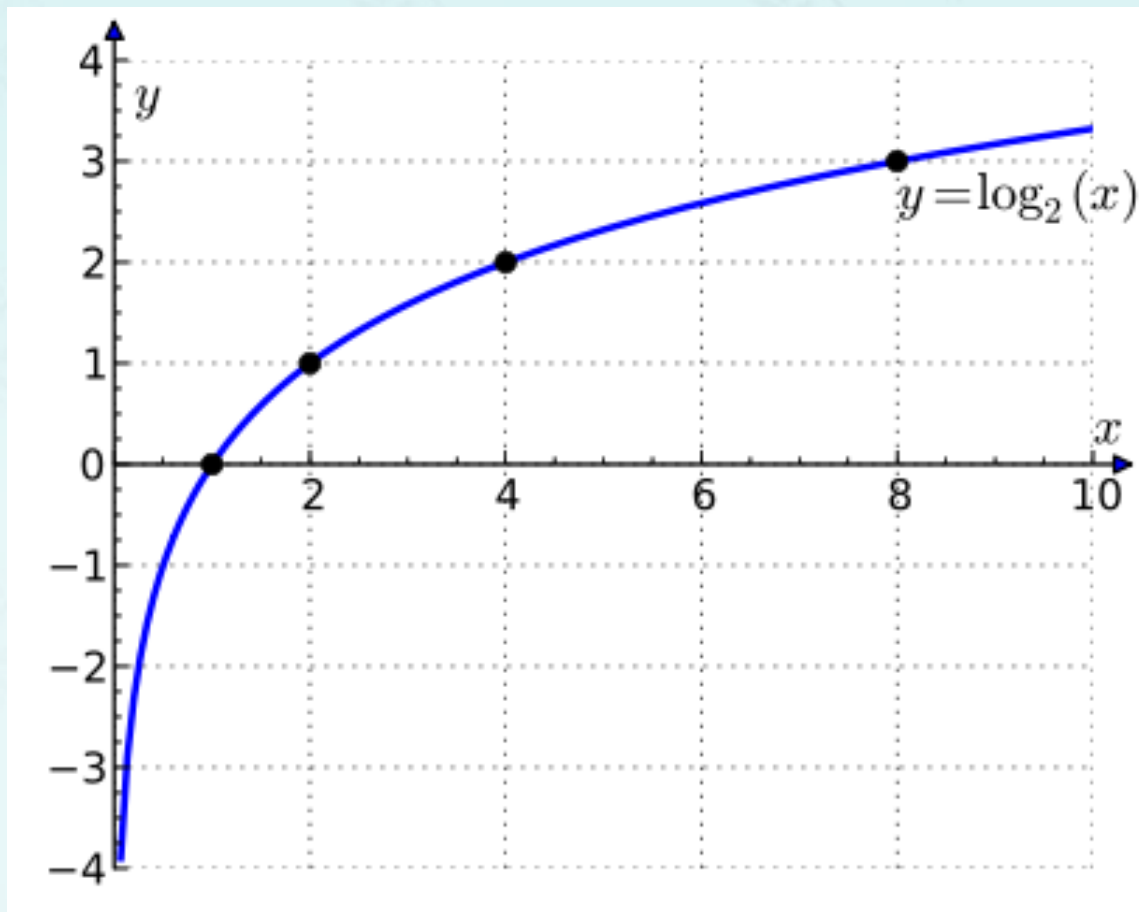
$$x^{x^{x^{\cdots}}} = 2$$

$$x^{x^{x^{\cdots}}} = 2$$



1. Logarithms

- Plot $y = \log_2 x$



1. Logarithms

Logarithmic Operators:

- $\log ab = \log a + \log b$
- $\log \frac{a}{b} = \log a - \log b$
- $\log a^b = b \log a$
- $\log_a n = \frac{\log_b n}{\log_b a}$ (this is used to change bases.)
- Evaluate $\log_4 2 + \log_4 32$
 - $\log_4 2 + \log_4 32 = \log_4 2 * 32 = \log_4 64 = (\log_4 4^3) = 3$
- Evaluate $\log_2 400$ (all most all calculators don't do base 2.... wow!)
 - $\log_2 400 = \frac{\log_e 400}{\log_e 2} = \frac{5.991}{0.69} = 8.68$
- What is x?
 - $2^x = 2^{10} + 2^{10}$
 - $x \lg 2 = \lg (2 * 2^{10}), \quad x = \lg 2 + \lg 2^{10}, \quad x = 1 + 10 = 11$

1. Logarithms

Logarithmic Operators:

- Logarithms can also be very useful for comparing very large or small numbers.
- **Exercise:** $10^{100} > 2^{256}$ is true or not?
Neither number can be calculated directly without risking overflow.
- Hint:
Since the logarithm function is monotonically increasing, if $a < b$, then $\log a < \log b$.
-
-

1. Logarithms

Exercise: Compute the order of *growth rate* b in $T(n) \cong a n^b$ of the running time as a function of n using Selection Sort of which the time complexity is $O(n^2)$.

n	time
100	0.000023
200	0.000079
300	0.000173
400	0.000299
500	0.000477
600	0.000660
700	0.000904
800	0.001174
900	0.001468
1000	0.001818

As input size changes, the growth rate b of the execution time would be

$$\left(\frac{n_2}{n_1}\right)^b = \frac{t_2}{t_1}$$

When input size increases twofold, it would be

$$(2)^b = \frac{t_2}{t_1}$$

In case of the $O(n^2)$ algorithm, the growth rate should be close to 2.0.

Let's pick up $n_1 = 500$ to $n_2 = 1000$, then $t_1 = 0.000477$ to $t_2 = 0.001818$, respectively.

The growth rate of this algorithm is

$$b = \log\left(\frac{t_2}{t_1}\right) = \log\left(\frac{0.001818}{0.000477}\right) = \log(3.81) = \mathbf{1.93}$$

1. Logarithms

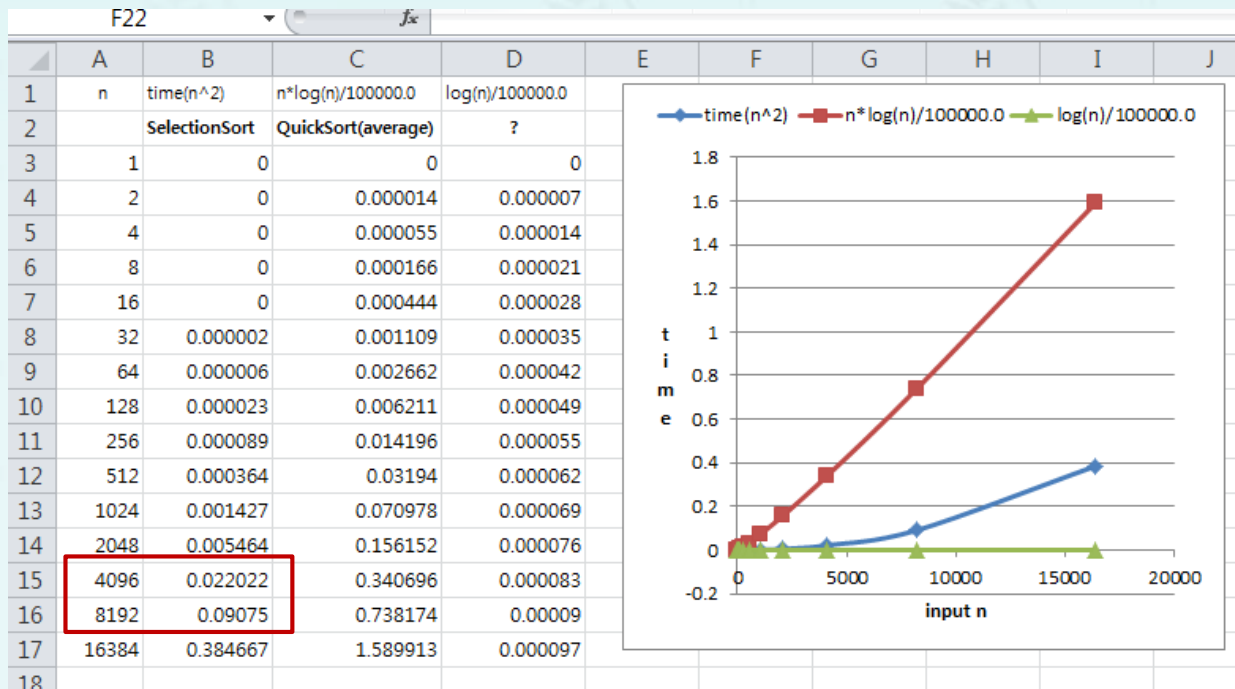
Exercise: Compute the order of growth rate b in $T(n) \cong a n^b$ of the running time as a function of n using Selection Sort of which the time complexity is $O(n^2)$.

Let's pick up

$n_1 = 4096$, $n_2 = 8192$, then $t_1 = 0.022022$, $t_2 = 0.09075$, respectively.

The **measured growth rate** b of the algorithm is

$$b = \log\left(\frac{t_2}{t_1}\right) = \log\left(\frac{0.09075}{0.022022}\right) = \log(4.12) = \mathbf{2.04}$$



2. Summations

- In analyzing a program's performance, we'll need to add up the number of times an operation is taken.
- It is typically written as:

$$\sum_{i=1}^n f(i)$$



a closed form

which is equivalent to $f(1) + f(2) + \dots + f(n)$

- **Example:**
 - summation of 1 ... 10?
 - summation of 1 ... n?

2. Summations

- We will be particularly interested in the following sum:

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}$$

- We can easily see the solution if we add the sequence twice with one of the sequences written in inverse order.

$$S = 1 + 2 + 3 + \dots + (n-1) + n$$

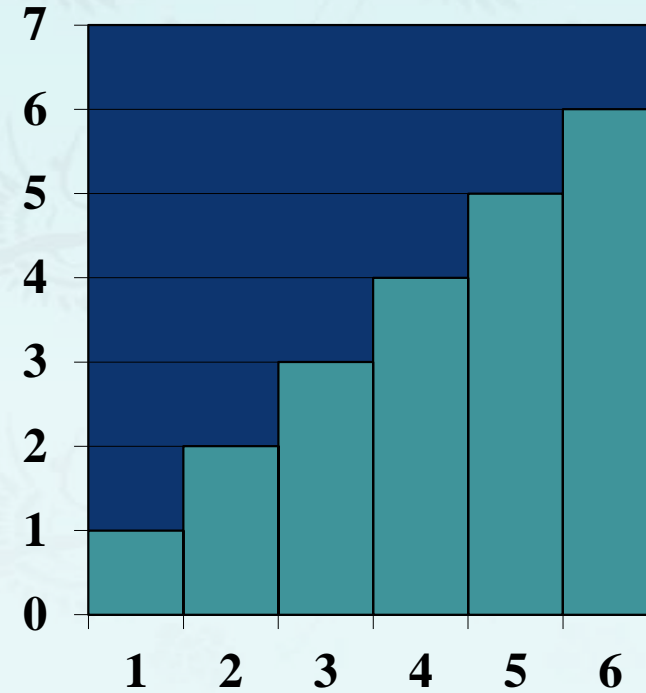
$$S = n + (n-1) + (n-2) + \dots + 2 + 1$$

- By adding two sequences, each of the pairs adds to $(n+1)$. There are n of them.
- $2S = (1+n) + (1+n) + (1+n) + \dots + (n+1) + (n+1)$
- $= n(n+1)$
- Therefore, $S = n(n+1)/2$.

2. Summations – Arithmetic Sum

- There is a simple visual proof of this fact for the following sum:

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}$$



2. Summations – Arithmetic Sum

- Some common summations and their closed-form solutions:

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^n i^2 = \frac{2n^3 + 3n^2 + n}{6}$$

2. Summations – Geometric Sum

We are also be interested in the sum, it is so called **geometric** sum;

$$\sum_{i=0}^n a^i = 1 + a^1 + a^2 + \dots + a^{n-1} + a^n$$

- **Proof:** Let's use S to denote the sum:
- $S = 1 + a^1 + a^2 + \dots + a^{n-1} + a^n$
- $aS = a^1 + a^2 + \dots + a^{n-1} + a^n + a^{n+1}$
 $= S + a^{n+1} - 1$

From $aS = S + a^{n+1} - 1$, **we solve for S , obtaining:**

$$S = \sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1}$$

2. Summations – Geometric Sum

We are also be interested in the sum, it is so called **geometric** sum;

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

Exercise: $\sum_{i=0}^{n-1} a^i =$

Exercise: $\sum_{i=0}^n 2^i =$



2. Summations – Geometric Sum

Infinite geometric series....

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

When $a < 1$ and n goes to infinity, the sum becomes

$$\sum_{i=0}^{\infty} a^i = \frac{1}{1 - a}$$

Exercise:

Compute the infinite “geometric” series intuitively and using arithmetic sum.

$$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots$$



2. Summations – Geometric Sum

Infinite geometric series....

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

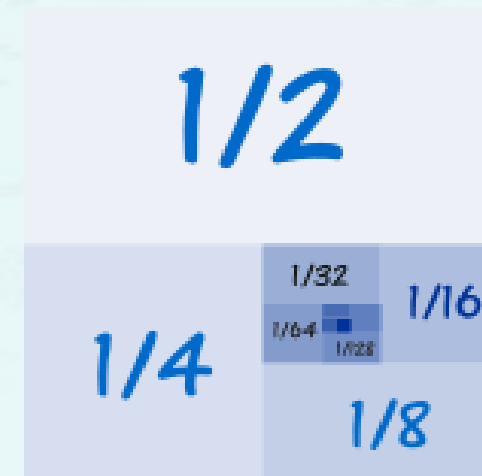
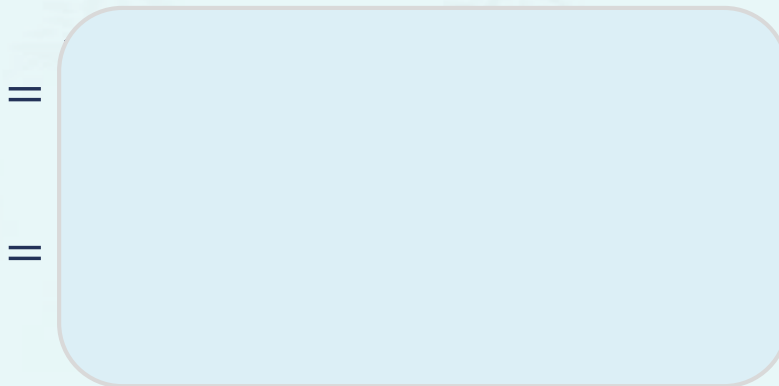
When $a < 1$ and n goes to infinity, the sum becomes

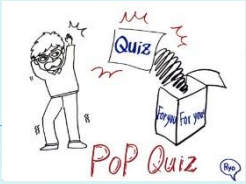
$$\sum_{i=0}^{\infty} a^i = \frac{1}{1 - a}$$

Exercise:

Compute the infinite “geometric” series intuitively and using arithmetic sum.

$$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots$$





2. Summations – Geometric Sum

Infinite geometric series....

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

When $a < 1$ and n goes to infinity, the sum becomes

$$\sum_{i=0}^{\infty} a^i = \frac{1}{1 - a}$$

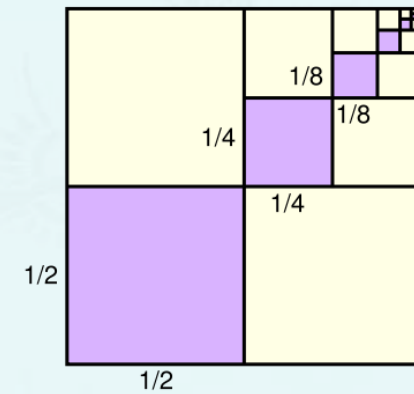
Exercise: Compute the sum of the areas of the purple squares.

Intuitively? Using arithmetic sum?

$$\frac{1}{4} + \frac{1}{16} + \frac{1}{64} \dots \dots$$

=

=





3. Mathematical Proofs

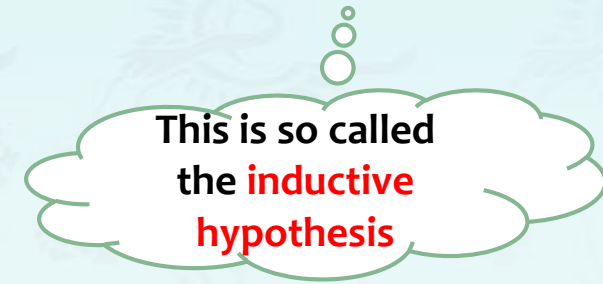
- The three most common forms of proof are:
 1. Direct proof (sometimes called a constructive proof)
 2. Indirect proof or proof by contradiction
 3. **Inductive proof**
 - It is very similar to recursion.
 - You establish a **base case** that is proved directly.
 - This is followed by an **inductive step** which shows how the **hypothesis** holds for **larger cases**.



3. Mathematical Proofs

- **Inductive proof**

- In data structures and algorithms, we often want to prove that something holds over a range of values
- The **base case** will prove the theorem for the initial c values.
- The **inductive step** will show that, if the theorem holds for $n - 1$, then it holds for n .
Alternatively, you may assume that it is true for n which is the inductive hypothesis first and then prove it for $n + 1$.
or for $2n$.



3. Mathematical Proofs

Inductive proof Example: prove $\sum_{i=1}^n i = \frac{n(n+1)}{2}$

- **Base case:** Let $n = 1$. $\frac{1(1+1)}{2} = \frac{2}{2} = 1$.
- **Inductive step:** We state the **inductive hypothesis** for $n - 1$ that:

$$\sum_{i=1}^{n-1} i = \frac{(n-1)((n-1)+1)}{2} = \frac{(n-1)(n)}{2}$$

- Assuming this is true, adding the **n th term** yields:

$$\sum_{i=1}^n i = \frac{(n-1)(n)}{2} + n = \frac{(n-1)(n)}{2} + \frac{2n}{2} = \frac{n(n+1)}{2}$$

Therefore, we have proved it.

3. Mathematical Proofs

Inductive proof Example: prove $\sum_{i=1}^n i = \frac{n(n+1)}{2}$

- **Base case:** Let $n = 1$. $\frac{1(1+1)}{2} = \frac{2}{2} = 1$.
- **Inductive step:** Assume that it holds for n , then for $2n$; that is :

$$\sum_{i=1}^{2n} i = \frac{2n(2n+1)}{2} = n(2n+1)$$

Using the **induction hypothesis** that the left side above can be rewritten and rearranged algebraically:

$$\begin{aligned}\sum_{i=1}^{2n} i &= \frac{n(n+1)}{2} + [(n+1) + (n+2) + \dots + (n+n)] \\ &= \frac{n(n+1)}{2} + n * n + \frac{n(n+1)}{2} \\ &= n(n+1) + n * n \\ &= n(2n+1)\end{aligned}$$

Therefore, we have proved it.

3. Mathematical Proofs

Inductive proof Exercise: prove $\sum_{i=1}^n i^2 = \frac{2n^3+3n^2+n}{6}$

- **Base case:** Let $n = 1$. $\frac{2+3+1}{6} = 1$.
- **Inductive step:** We state the **inductive hypothesis** for $n - 1$ that:

$$\sum_{i=1}^{n-1} i^2 = \frac{2(n-1)^3+3(n-1)^2+(n-1)}{6}$$

- Assuming this is true, adding the **n th term** yields:

$$\begin{aligned} \sum_{i=1}^{n-1} i^2 + n^2 &= \frac{2(n-1)^3+3(n-1)^2+(n-1)}{6} + \frac{6n^2}{6} \\ &= \frac{2n^3+3n^2+n}{6} \end{aligned}$$

Therefore, we have proved it.

Exercise: Prove that $8^n - 3^n$ is divisible by 5 for all $n \geq 1$.



Summary

&

quaestio quaestio qo → q ? ? ?