Summing

Time for Summin' Different

Aidan Irwin, Dylan Prior, Finn Riches, and Melvin Page

EXETER MATHS SCHOOL
FREEDOM FOR THINKERS

Introduction

Our project began with the observation that taking a cumulative sum (running total) of the odd numbers seemed to return the square numbers: $1 = 1^2$, $1 + 3 = 4 = 2^2$, $1 + 3 + 5 = 9 = 3^2$ and so on. Then, it became apparent that a similar process worked to get numbers of any power, as explained below.

Methods

First, we start with the natural numbers ($\mathbb{N}$). These are the positive whole numbers starting from 1. $\mathbb{N}$ = 1, 2, 3, 4, 5, 6, 7, 8, 9 ...
We then remove every 2nd number, beginning with 2, giving R_1. And finally, we find the cumulative sum (running total) of this, giving C_{R_1}.

$\mathbb{N}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
R_1	1		3		5		7		9		11		13		15	...
C_{R_1}	1		4		9		16		25		36		49		64	...

This algorithm gives the square numbers, and similar can be done to reach the cube numbers:
Remove every 3rd number beginning with 3 from $\mathbb{N}$, giving R_1, then find the cumulative sum of that (C_{R_1}), then remove every 2nd number starting with the 2nd value, giving R_2, and finally find the cumulative sum of that (C_{R_2}).

$\mathbb{N}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
R_1	1	2		4	5		7	8		10	11		13	14		...
C_{R_1}	1	3		7	12		19	27		37	48		61	75		...
R_2	1			7			19			37			61			...
C_{R_2}	1			8			27			64			125			...

Similar can be done in order to find numbers of any power, however the scope of this project only encompasses up to the powers of 3 for the majority of the project.

Main Objectives

1. Prove that you can use summations to calculate the powers of 2
2. Prove that you can use summations to calculate the powers of 3
3. Explore the efficiency of this method compared to traditional approaches
4. Develop a GUI to show how our method works
5. Use summation to calculate Euler's Number
6. Explore other applications of the method, including looking into what happens with modular arithmetic or when we vary the gap removed in a non-linear fashion

Proof of the Process for the Square Numbers

Let's consider the natural numbers from 1 to n:

$$\mathbb{N}_n = 1, 2, 3, ..., n-1, n$$

By removing every other number, we can get an arithmetic progression (a sequence of numbers that begin at a set value and where the difference between each term is a constant) of first term $a = 1$ and common difference $d = 2$.

$$R_1 = a, a + d, a + 2d, ..., a + (n-2)d, a + (n-1)d$$

Now, let's consider the cumulative sum of the first n terms of a general arithmetic progression with first term a and common difference d - the cumulative sum of this arithmetic progression, C_{R_1}.

$$C_{R_1} = a + (a + d) + (a + 2d) + ... + (a + (n-2)d) + (a + (n-1)d)$$

And by reversing the order:

$$C_{R_1} = (a + (n-1)d) + (a + (n-2)d) + ... + (a + 2d) + (a + d) + a$$

And now if we list these side-by-side and find the sum of each term:

Sum to n (C_{R_1})		**C_{R_1} reversed**		$2C_{R_1}$
a	+	$a + (n-1)d$	=	$2a + (n-1)d$
+		+		+
$(a + d)$	+	$a + (n-2)d$	=	$2a + (n-1)d$
+		+		+
...		...		... (continuing on n times)
+		+		+
$a + (n-2)d$	+	$(a + d)$	=	$2a + (n-1)d$
+		+		+
$a + (n-1)d$	+	a	=	$2a + (n-1)d$
=		=		=
C_{R_1}	+	C_{R_1}		$n(2a + (n-1)d)$

Hence,

$$2C_{R_1} = n(2a + (n-1)d)$$

$$C_{R_1} = \frac{n}{2}(2a + (n-1)d) \quad (1)$$

And this works for any arithmetic progression. Using $a = 1$ and $d = 2$ as outlined previously for every 2nd number:

$$C_{R_1} = \frac{n}{2}(2 + 2(n-1))$$

$$C_{R_1} = \frac{1}{2}(2n^2)$$

$$C_{R_1} = n^2$$

Hence we have proved that the first n numbers with every 2nd number removed will always sum to the nth square number.

Proof of the Process for the Cube Numbers

First, we start with the natural numbers:

$$\mathbb{N} = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...$$

Next, remove every 3rd number, starting with 3:

$$R_1 = 1, 2, 4, 5, 7, 8, 10, 11...$$

Then split the resulting sequence into two, one with the first of each "pair" of numbers between each gap and one with the second:

$$P_1 = 1, 4, 7, 10, ...$$

$$P_2 = 2, 5, 8, 11, ...$$

Then find the cumulative sum of these:

$$C_{P_1} = 1, 5, 12, 22, ...$$

$$C_{P_2} = 2, 7, 15, 26, ...$$

These can be represented as equations to find any term within them, using the summation formula (as proven earlier)

$$\frac{n}{2}(2a + (n-1)d) \quad (1)$$

yielding the following:

$$C_{P_1} = \frac{n}{2}(2 + 3(n-1)) = \frac{3n^2 - n}{2}$$

$$C_{P_2} = \frac{n}{2}(4 + 3(n-1)) = \frac{3n^2 + n}{2}$$

To find n^3, we must find $(n-1)^3$ + $(C_{P_1})_n$ + $(C_{P_2})_n$ - $(P_2)_n$
For instance when $n = 2$, $(n-1)^3 = 1$, $(C_{P_1})_n = 5$, $(C_{P_2})_n = 7$, and $(P_2)_n = 5$, so $2^3 = 1 + 5 + 7 - 5 = 8$
Let's find what a generalised $C_{P_1} + C_{P_2} - P_2$ equals:

$$P_2 = 3n - 1$$

$$C_{P_1} + C_{P_2} - P_2 = 3n^2 - 3n + 1$$

Now, let's find a generalised form of the previous cube number, $(n-1)^3$:

$$(n-1)^3 = n^3 - 3n^2 + 3n - 1$$

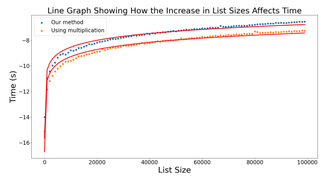
And finally, let's combine these two:

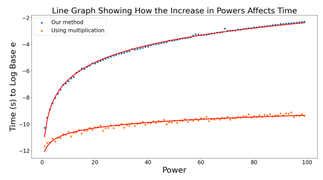
$$(n^3 - 3n^2 + 3n - 1) + (3n^2 - 3n + 1) = n^3$$

Hence by mathematical induction n^3 can be found by applying the algorithm on the previous cube number $(n-1)^3$ for any n.

Efficiency Compared to Traditional Methods

Throughout this project, we have looked at how our method of generating the powers of two and beyond compares with more traditional methods of achieving the same outcome. The first comparison we attempted was seeing how our summation stacked up against simply multiplying numbers together - also known as the multiplicative approach - e.g. doing 2^2 as $2 \times 2 = 4$.
In order to compare the efficiency of the two approaches, we have used Big O notation, which is a way of showing how much more time the processing of an algorithm takes as the size of its input increases.
The two scatter plots below show the outcomes of this comparison:





Graph 1 illustrates that while our code shares the same Big O complexity - $O(n)$ - as the multiplicative approach, it performs significantly slower in generating square numbers as the size of the list increases. This highlights the inefficiency of our implementation compared to direct multiplication.
Graph 2 demonstrates that, for generating lists of increasing powers, our code exhibits a much higher computational cost, with a Big O complexity of $O(n^2)$ compared to the $O(n)$ complexity of the multiplicative approach. This difference becomes more greater as the list size grows.

Another Application of Summation: Calculating Euler's Number

Euler's number, e, was originally encountered by the Swiss mathematician Jacob Bernoulli while studying compound interest. Across mathematics, it is a very important number - and it can be calculated using summations.
A definition for e goes as follows:

$$e = \lim_{n \to \infty} \left(1 + \frac{1}{n}\right)^n$$

This expression arises from the computation of compound interest, where n is the number of intervals in the year on which the compound interest is evaluated. For example $n = 12$ for monthly compounding.

Euler proved that e is the infinite sum:

$$e = \sum_{k=0}^{\infty} \frac{1}{k!}$$

He did this by first expanding the original expression by using the binomial theorem to result in:

$$\left(1 + \frac{1}{n}\right)^n = \sum_{k=0}^{n} \frac{n!}{k!(n-k)!} \times \frac{1}{n^k}$$

If we consider the expression:

$$\frac{n!}{(n-k)!}$$

This can be written in the form:

$$\frac{n \times (n-1) \times (n-2) \times ... \times (n-k+1) \times (n-k) \times ... \times 3 \times 2 \times 1}{(n-k) \times (n-k-1) \times (n-k-2) \times ... \times 3 \times 2 \times 1}$$

This means that the fraction will cancel up to $(n-k+1)$.

This means that the expression can then be simplified into:

$$\sum_{k=0}^{n} \frac{n \times (n-1) \times (n-2)...(n-k+1)}{k! \times n^k}$$

As n grows larger, the expression below tends to 1.

$$\frac{n \times (n-1) \times (n-2)...(n-k+1)}{n^k} \to 1$$

This is because if we break up the terms individually to create the expression:

$$\frac{n}{n} \times \frac{n-1}{n} \times \frac{n-2}{n} \times ... \times \frac{n-k+1}{n}$$

Each factor can be represented as the form $\frac{n-j}{n}$
for $j = 0, 1, 2, ..., k-1$. This means that as $n \to \infty$, $\frac{n-j}{n} \to 1$.

Therefore the entire expression of

$$\sum_{k=0}^{n} \frac{n \times (n-1) \times (n-2)...(n-k+1)}{k! \times n^k}$$

can be simplified into:

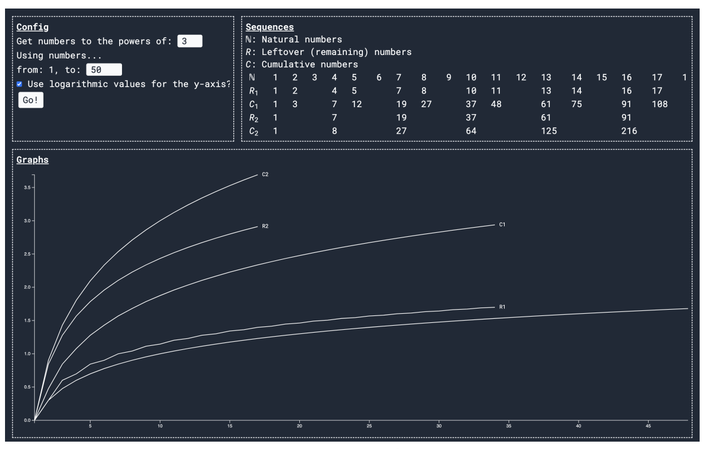
$$\sum_{k=0}^{n} \frac{1}{k!}$$

And as previously stated, $n \to \infty$. Therefore, e can be represented by the form:

$$e = \sum_{k=0}^{\infty} \frac{1}{k!}$$

A GUI Showcasing Our Algorithm

As part of this project, we set out to create a graphical user interface that visually represents our algorithm. This was coded in the form of an online web application, which uses JavaScript, Svelte/SvelteKit, TailwindCSS, D3.js, and Vercel.



The above image shows the app calculating n^3 using our algorithm and plotting the values on a graph with a logarithmic y-axis.

Accessible at `https://summingemc.com`, the site allows you to input a value to get numbers to the power of (i.e. calculate n^m by selecting the value of m) and up to what number it uses. It then creates a table with the each step to the final sequence of n^m by using our algorithm. A graph is plotted of each row in the table, showing visually how the final sequence is obtained. There is also the option to use logarithmic values for the y-axis on this graph.

Further Research

Due to the time constraints we faced during the project we were left with a few things we ran out of time to investigate in further depth. These include:

- Prove that our algorithm will work to calculate n^m for any integer m.
- Research further applications of this method to find square numbers, such as using it to easily get from one square to the next.

Conclusions

- Throughout the project, we have shown unconventional uses for summation in mathematics, mainly focusing on their applications to generate numbers to a power.
- We proved that this algorithm works for computing the powers of 2 and 3.
- We investigated the efficiency of our algorithm when compared to more traditional techniques, showing that in theory they should be approximately the same.
- We also demonstrated how summation can be used to derive Euler's number.
- Additionally, we explored the use of the algorithm on modular arithmetic and looked at what happens when we vary the gap of numbers removed in a non-linear fashion, but did not find anything of note.

Acknowledgements

We would like to thank Professor Kyle Wedgwood for setting us this project and providing support throughout, as well as Dr Ed Horncastle for leading the Exeter Mathematics Certificate and providing support to us throughout. We would also like to thank Chris Child, Alex Toogood-Johnson, and Lyall Stewart for their invaluable support in proofreading our poster. We would also like to acknowledge Orlando Graham and Emily Millington for their help in completing this poster.

Contact Details
team@summingemc.com
Aidan Irwin: aidan@summingemc.com / aidanirwin@exe-coll.ac.uk
Dylan Prior: dylan@summingemc.com / dylanprior@exe-coll.ac.uk
Finn Riches: finn@summingemc.com / finnriches@exe-coll.ac.uk
Melvin Page: melvin@summingemc.com / melvinpage@exe-coll.ac.uk
See `https://summingemc.com/poster` for an online copy of this poster.