Mathematical Algorithms

[**Learn more about Mathematical Algorithms in DSA Self Paced Course**](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=geeksforgeeks&utm_medium=articles+mathematical_lp+header_link_click&utm_campaign=dsa+course+tracker)

[**Practice Problems on Mathematical Algorithms**](https://practice.geeksforgeeks.org/topics/mathematical/?utm_source=geeksforgeeks&utm_medium=articles+mathematical_lp+header_link_click&utm_campaign=practice+tracker)

[**Recent Articles on Mathematical Algorithms**](https://www.geeksforgeeks.org/category/mathematical/)

**Topics:**

* [Divisibility & Large Numbers](https://www.geeksforgeeks.org/mathematical-algorithms/#divisibility)
* [GCD and LCM](https://www.geeksforgeeks.org/mathematical-algorithms/#gcd)
* [Series](https://www.geeksforgeeks.org/mathematical-algorithms/#series)
* [Number Digits](https://www.geeksforgeeks.org/mathematical-algorithms/#number)
* [Algebra](https://www.geeksforgeeks.org/mathematical-algorithms/#algebra)
* [Number System](https://www.geeksforgeeks.org/mathematical-algorithms/#system)
* [Prime numbers & Primality Tests](https://www.geeksforgeeks.org/mathematical-algorithms/#prime)
* [Prime Factorization & Divisors](https://www.geeksforgeeks.org/mathematical-algorithms/#factors)
* [Modular Arithmetic](https://www.geeksforgeeks.org/mathematical-algorithms/#modular)
* [Factorial](https://www.geeksforgeeks.org/mathematical-algorithms/#factorial)
* [Fibonacci Numbers](https://www.geeksforgeeks.org/mathematical-algorithms/#fibonacci)
* [Catalan Numbers](https://www.geeksforgeeks.org/mathematical-algorithms/#catalan)
* [nCr Computations](https://www.geeksforgeeks.org/mathematical-algorithms/#computation)
* [Set Theory](https://www.geeksforgeeks.org/mathematical-algorithms/#set)
* [Sieve Algorithms](https://www.geeksforgeeks.org/mathematical-algorithms/#sieve)
* [Euler Totient Function](https://www.geeksforgeeks.org/mathematical-algorithms/#euler)
* [Chinese Remainder Theorem](https://www.geeksforgeeks.org/mathematical-algorithms/#chinese)
* [Some practice Problems](https://www.geeksforgeeks.org/mathematical-algorithms/#practice)

**Divisibility & Large Numbers:**

1. [Check if a large number is divisible by 3 or not](https://www.geeksforgeeks.org/check-large-number-divisible-3-not/)
2. [Check if a large number is divisible by 4 or not](https://www.geeksforgeeks.org/check-large-number-divisible-4-not/)
3. [Check if a large number is divisible by 6 or not](https://www.geeksforgeeks.org/check-large-number-divisible-6-not/)
4. [Check divisibility by 7](https://www.geeksforgeeks.org/divisibility-by-7/)
5. [Check if a large number is divisible by 9 or not](https://www.geeksforgeeks.org/check-large-number-divisible-9-not/)
6. [Check if a large number is divisible by 11 or not](https://www.geeksforgeeks.org/check-large-number-divisible-11-not/)
7. [Divisibility by 12 for a large number](https://www.geeksforgeeks.org/divisibility-by-12-for-a-large-number/)
8. [Check if a large number is divisible by 13 or not](https://www.geeksforgeeks.org/check-large-number-divisible-13-not/)
9. [Check if a large number is divisibility by 15](https://www.geeksforgeeks.org/check-large-number-divisibility-15/)
10. [Number is divisible by 29 or not](https://www.geeksforgeeks.org/number-is-divisible-by-29-or-not/)

**GCD and LCM:**

1. [LCM of array](https://www.geeksforgeeks.org/lcm-of-given-array-elements/)
2. [GCD of array](https://www.geeksforgeeks.org/gcd-two-array-numbers/)
3. [Basic and Extended Euclidean algorithms](https://www.geeksforgeeks.org/basic-and-extended-euclidean-algorithms/)
4. [Stein’s Algorithm for finding GCD](https://www.geeksforgeeks.org/steins-algorithm-for-finding-gcd/)
5. [GCD, LCM and Distributive Property](https://www.geeksforgeeks.org/gcd-lcm-distributive-property/)
6. [Count number of pairs (A <= N, B <= N) such that gcd (A, B) is B](https://www.geeksforgeeks.org/count-number-pairs-n-b-n-gcd-b-b/)
7. [Program to find GCD of floating point numbers](https://www.geeksforgeeks.org/program-find-gcd-floating-point-numbers/)
8. [Series with largest GCD and sum equals to n](https://www.geeksforgeeks.org/series-largest-gcd-sum-equals-n/)
9. [Largest Subset with GCD 1](https://www.geeksforgeeks.org/largest-subset-gcd-1/)
10. [Summation of GCD of all the pairs up to N](https://www.geeksforgeeks.org/summation-gcd-pairs-n/)

**Series:**

1. [Juggler Sequence](https://www.geeksforgeeks.org/juggler-sequence/)
2. [Padovan Sequence](https://www.geeksforgeeks.org/padovan-sequence/)
3. [Aliquot Sequence](https://www.geeksforgeeks.org/aliquot-sequence/)
4. [Moser-de Bruijn Sequence](https://www.geeksforgeeks.org/moser-de-bruijn-sequence/)
5. [Stern-Brocot Sequence](https://www.geeksforgeeks.org/stern-brocot-sequence/)
6. [Newman-Conway Sequence](https://www.geeksforgeeks.org/newman-conway-sequence/)
7. [Sylvester’s sequence](https://www.geeksforgeeks.org/sylvesters-sequence/)
8. [Recaman’s sequence](https://www.geeksforgeeks.org/recamans-sequence/)
9. [Sum of the sequence 2, 22, 222, ………](https://www.geeksforgeeks.org/sum-sequence-2-22-222/)
10. [Sum of series 1^2 + 3^2 + 5^2 + . . . + (2\*n – 1)^2](https://www.geeksforgeeks.org/sum-series-12-32-52-2n-12/)
11. [Sum of the series 0.6, 0.06, 0.006, 0.0006, …to n terms](https://www.geeksforgeeks.org/sum-series-0-6-0-06-0-006-0-0006-n-terms/)
12. [n-th term in series 2, 12, 36, 80, 150….](https://www.geeksforgeeks.org/n-th-term-series-2-12-36-80-150/)

**Number Digits:**

1. [Minimum digits to remove to make a number Perfect Square](https://www.geeksforgeeks.org/required-minimum-digits-remove-number-make-perfect-square/)
2. [Print first k digits of 1/n where n is a positive integer](https://www.geeksforgeeks.org/print-first-k-digits-1n-n-positive-integer/)
3. [Check if a given number can be represented in given a no. of digits in any base](https://www.geeksforgeeks.org/check-if-a-given-number-can-be-represented-in-given-a-no-of-digits-in-any-base/)
4. [Find element using minimum segments in Seven Segment Display](https://www.geeksforgeeks.org/find-element-using-minimum-segments-seven-segment-display/)
5. [Find next greater number with same set of digits](https://www.geeksforgeeks.org/find-next-greater-number-set-digits/)
6. [Check if a number is jumbled or not](https://www.geeksforgeeks.org/check-if-a-number-is-jumbled-or-not/)
7. [Numbers having difference with digit sum more than s](https://www.geeksforgeeks.org/numbers-difference-digit-sum-s/)
8. [Total numbers with no repeated digits in a range](https://www.geeksforgeeks.org/total-numbers-no-repeated-digits-range/)
9. [K-th digit in ‘a’ raised to power ‘b’](https://www.geeksforgeeks.org/k-th-digit-raised-power-b/)

**Algebra:**

1. [Program to add two polynomials](https://www.geeksforgeeks.org/program-add-two-polynomials/)
2. [Multiply two polynomials](https://www.geeksforgeeks.org/multiply-two-polynomials-2/)
3. [Find number of solutions of a linear equation of n variables](https://www.geeksforgeeks.org/find-number-of-solutions-of-a-linear-equation-of-n-variables/)
4. [Calculate the Discriminant Value](https://www.geeksforgeeks.org/calculate-discriminant-value/)
5. [Program for dot product and cross product of two vectors](https://www.geeksforgeeks.org/program-dot-product-cross-product-two-vector/)
6. [Iterated Logarithm log\*(n)](https://www.geeksforgeeks.org/iterated-logarithm-logn/)
7. [Program to find correlation coefficient](https://www.geeksforgeeks.org/program-find-correlation-coefficient/)
8. [Program for Muller Method](https://www.geeksforgeeks.org/program-muller-method/)
9. [Number of non-negative integral solutions of a + b + c = n](https://www.geeksforgeeks.org/number-non-negative-integral-solutions-b-c-n/)
10. [Generate Pythagorean Triplets](https://www.geeksforgeeks.org/generate-pythagorean-triplets/)

**Number System:**

1. [Exponential notation of a decimal number](https://www.geeksforgeeks.org/exponential-notation-decimal-number/)
2. [Check if a number is power of k using base changing method](https://www.geeksforgeeks.org/check-number-power-k-using-base-changing-method/)
3. [Convert a binary number to hexadecimal number](https://www.geeksforgeeks.org/convert-binary-number-hexadecimal-number/)
4. [Program for decimal to hexadecimal conversion](https://www.geeksforgeeks.org/program-decimal-hexadecimal-conversion/)
5. [Converting a Real Number (between 0 and 1) to Binary String](https://www.geeksforgeeks.org/converting-a-real-number-between-0-and-1-to-binary-string/)
6. [Convert from any base to decimal and vice versa](https://www.geeksforgeeks.org/convert-base-decimal-vice-versa/)
7. [Decimal to binary conversion without using arithmetic operators](https://www.geeksforgeeks.org/decimal-binary-conversion-without-using-arithmetic-operators/)

**Prime Numbers & Primality Tests:**

1. [Prime Numbers](https://www.geeksforgeeks.org/prime-numbers/)
2. [Left-Truncatable Prime](https://www.geeksforgeeks.org/left-truncatable-prime/)
3. [Mersenne Prime](https://www.geeksforgeeks.org/mersenne-prime/)
4. [Super Prime](https://www.geeksforgeeks.org/super-prime/)
5. [Hardy-Ramanujan Theorem](https://www.geeksforgeeks.org/hardy-ramanujan-theorem/)
6. [Rosser’s Theorem](https://www.geeksforgeeks.org/rossers-theorem/)
7. [Fermat’s little theorem](https://www.geeksforgeeks.org/fermats-little-theorem/)
8. [Introduction to Primality Test and School Method](https://www.geeksforgeeks.org/introduction-to-primality-test-and-school-method/)
9. [Vantieghems Theorem for Primality Test](https://www.geeksforgeeks.org/vantieghems-theorem-primality-test/)
10. [AKS Primality Test](https://www.geeksforgeeks.org/aks-primality-test/)
11. [Lucas Primality Test](https://www.geeksforgeeks.org/lucas-primality-test/)

**Prime Factorization & Divisors:**

1. [Prime factors](https://www.geeksforgeeks.org/print-all-prime-factors-of-a-given-number/)
2. [Smith Numbers](https://www.geeksforgeeks.org/smith-number/)
3. [Sphenic Number](https://www.geeksforgeeks.org/sphenic-number/)
4. [Hoax Number](https://www.geeksforgeeks.org/hoax-number/)
5. [k-th prime factor of a given number](https://www.geeksforgeeks.org/k-th-prime-factor-given-number/)
6. [Pollard’s Rho Algorithm for Prime Factorization](https://www.geeksforgeeks.org/pollards-rho-algorithm-prime-factorization/)
7. [Finding power of prime number p in n!](https://www.geeksforgeeks.org/finding-power-prime-number-p-n/)
8. [Find all divisors of a natural number](https://www.geeksforgeeks.org/find-divisors-natural-number-set-1/)
9. [Find numbers with n-divisors in a given range](https://www.geeksforgeeks.org/find-numbers-n-divisors-given-range/)

**Modular Arithmetic:**

1. [Modular Exponentiation (Power in Modular Arithmetic)](https://www.geeksforgeeks.org/modular-exponentiation-power-in-modular-arithmetic/)
2. [Modular multiplicative inverse](https://www.geeksforgeeks.org/multiplicative-inverse-under-modulo-m/)
3. [Modular Division](https://www.geeksforgeeks.org/modular-division/)
4. [Euler’s criterion (Check if square root under modulo p exists)](https://www.geeksforgeeks.org/eulers-criterion-check-if-square-root-under-modulo-p-exists/)
5. [Find sum of modulo K of first N natural number](https://www.geeksforgeeks.org/find-sum-modulo-k-first-n-natural-number/)
6. [How to compute mod of a big number?](https://www.geeksforgeeks.org/how-to-compute-mod-of-a-big-number/)
7. [Exponential Squaring (Fast Modulo Multiplication)](https://www.geeksforgeeks.org/exponential-squaring-fast-modulo-multiplication/)
8. [Trick for modular division ( (x1 \* x2 …. xn) / b ) mod (m)](https://www.geeksforgeeks.org/trick-modular-division-x1x2-xnbmodm/)

**Factorial:**

1. [Program for factorial of a number](https://www.geeksforgeeks.org/program-for-factorial-of-a-number/)
2. [Legendre’s formula (Given p and n, find the largest x such that p^x divides n!)](https://www.geeksforgeeks.org/given-p-and-n-find-the-largest-x-such-that-px-divides-n-2/)
3. [Count trailing zeroes in factorial of a number](https://www.geeksforgeeks.org/count-trailing-zeroes-factorial-number/)
4. [Factorial of a large number](https://www.geeksforgeeks.org/factorial-large-number/)
5. [Primorial of a number](https://www.geeksforgeeks.org/primorial-of-a-number/)
6. [Find maximum power of a number that divides a factorial](https://www.geeksforgeeks.org/find-maximum-power-number-divides-factorial/)
7. [Largest power of k in n! (factorial) where k may not be prime](https://www.geeksforgeeks.org/largest-power-k-n-factorial-k-may-not-prime/)
8. [Check if a number is a Krishnamurthy Number or not](https://www.geeksforgeeks.org/check-if-a-number-is-a-krishnamurthy-number-or-not-2/)
9. [Last non-zero digit of a factorial](https://www.geeksforgeeks.org/last-non-zero-digit-factorial/)
10. [Count digits in a factorial using Logarithm](https://www.geeksforgeeks.org/count-digits-in-a-factorial-using-logarithm/)

**Fibonacci Numbers:**

1. [Fibonacci Numbers](https://www.geeksforgeeks.org/program-for-nth-fibonacci-number/)
2. [Interesting facts about Fibonacci numbers](https://www.geeksforgeeks.org/interesting-facts-fibonacci-numbers/)
3. [Zeckendorf’s Theorem (Non-Neighbouring Fibonacci Representation)](https://www.geeksforgeeks.org/zeckendorfs-theorem-non-neighbouring-fibonacci-representation/)
4. [Finding nth Fibonacci Number using Golden Ratio](https://www.geeksforgeeks.org/find-nth-fibonacci-number-using-golden-ratio/)
5. [Matrix Exponentiation](https://www.geeksforgeeks.org/matrix-exponentiation/)
6. [Fibonacci Coding](https://www.geeksforgeeks.org/fibonacci-coding/)
7. [Cassini’s Identity](https://www.geeksforgeeks.org/cassinis-identity/)
8. [Tail Recursion for Fibonacci](https://www.geeksforgeeks.org/tail-recursion-fibonacci/)

**Catalan Numbers:**

1. [Catalan numbers](https://www.geeksforgeeks.org/program-nth-catalan-number/)
2. [Applications of Catalan Numbers](https://www.geeksforgeeks.org/applications-of-catalan-numbers/)
3. [Dyck path](https://www.geeksforgeeks.org/dyck-path/)
4. [Succinct Encoding of Binary Tree](https://www.geeksforgeeks.org/succinct-encoding-of-binary-tree/)
5. [Find the number of valid parentheses expressions of given length](https://www.geeksforgeeks.org/find-number-valid-parentheses-expressions-given-length/)

**nCr Computations:**

1. [Binomial Coefficient](https://www.geeksforgeeks.org/dynamic-programming-set-9-binomial-coefficient/)
2. [Introduction and Dynamic Programming solution to compute nCr%p](https://www.geeksforgeeks.org/introduction-and-dynamic-programming-solution-to-compute-ncrp/)
3. [Program to calculate value of nCr](https://www.geeksforgeeks.org/program-calculate-value-ncr/)
4. [Rencontres Number (Counting partial derangements)](https://www.geeksforgeeks.org/rencontres-number-counting-partial-derangements/)
5. [Sum of squares of binomial coefficients](https://www.geeksforgeeks.org/sum-squares-binomial-coefficients/)
6. [Space and time efficient Binomial Coefficient](https://www.geeksforgeeks.org/space-and-time-efficient-binomial-coefficient/)
7. [Horner’s Method for Polynomial Evaluation](https://www.geeksforgeeks.org/horners-method-polynomial-evaluation/)

**Set Theory:**

1. [Power Set](https://www.geeksforgeeks.org/power-set/)
2. [Minimize the absolute difference of sum of two subsets](https://www.geeksforgeeks.org/minimize-absolute-difference-sum-two-subsets/)
3. [Sum of all subsets of a set formed by first n natural numbers](https://www.geeksforgeeks.org/sum-subsets-set-formed-first-n-natural-numbers/)
4. [Sum of average of all subsets](https://www.geeksforgeeks.org/sum-average-subsets/)
5. [Bell Numbers (Number of ways to Partition a Set)](https://www.geeksforgeeks.org/bell-numbers-number-of-ways-to-partition-a-set/)

**Sieve Algorithms:**

1. [Sieve of Eratosthenes](https://www.geeksforgeeks.org/sieve-of-eratosthenes/)
2. [Segmented Sieve](https://www.geeksforgeeks.org/segmented-sieve/)
3. [Sieve of Atkin](https://www.geeksforgeeks.org/sieve-of-atkin/)
4. [Sieve of Sundaram to print all primes smaller than n](https://www.geeksforgeeks.org/sieve-sundaram-print-primes-smaller-n/)
5. [Sieve of Eratosthenes in 0(n) time complexity](https://www.geeksforgeeks.org/sieve-eratosthenes-0n-time-complexity/)
6. [Prime Factorization using Sieve O(log n) for multiple queries](https://www.geeksforgeeks.org/prime-factorization-using-sieve-olog-n-multiple-queries/)

**Euler Totient Function:**

1. [Euler’s Totient Function](https://www.geeksforgeeks.org/eulers-totient-function/)
2. [Optimized Euler Totient Function for Multiple Evaluations](https://www.geeksforgeeks.org/optimized-euler-totient-function-multiple-evaluations/)
3. [Euler’s Totient function for all numbers smaller than or equal to n](https://www.geeksforgeeks.org/eulers-totient-function-for-all-numbers-smaller-than-or-equal-to-n/)
4. [Primitive root of a prime number n modulo n](https://www.geeksforgeeks.org/primitive-root-of-a-prime-number-n-modulo-n/)
5. [Euler’s Four Square Identity](https://www.geeksforgeeks.org/eulers-four-square-identity/)

**Chinese Remainder Theorem:**

1. [Introduction to Chinese Remainder Theorem](https://www.geeksforgeeks.org/introduction-to-chinese-remainder-theorem/)
2. [Implementation of Chinese Remainder theorem (Inverse Modulo based implementation)](https://www.geeksforgeeks.org/implementation-of-chinese-remainder-theorem-inverse-modulo-based-implementation/)
3. [Cyclic Redundancy Check and Modulo-2 Division](https://www.geeksforgeeks.org/modulo-2-binary-division/)
4. [Using Chinese Remainder Theorem to Combine Modular equations](https://www.geeksforgeeks.org/using-chinese-remainder-theorem-combine-modular-equations/)

**Some practice Problems:**

1. [Interquartile Range (IQR)](https://www.geeksforgeeks.org/interquartile-range-iqr/)
2. [Simulated Annealing](https://www.geeksforgeeks.org/simulated-annealing/)
3. [Pseudo Random Number Generator (PRNG)](https://www.geeksforgeeks.org/pseudo-random-number-generator-prng/)
4. [Square root of a number using log](https://www.geeksforgeeks.org/square-root-number-using-log/)
5. [Find ways an Integer can be expressed as sum of n-th power of unique natural numbers](https://www.geeksforgeeks.org/find-ways-integer-can-expressed-sum-n-th-power-unique-natural-numbers/)
6. [N-th root of a number](https://www.geeksforgeeks.org/n-th-root-number/)
7. [Fast Fourier Transformation for poynomial multiplication](https://www.geeksforgeeks.org/fast-fourier-transformation-poynomial-multiplication/)
8. [Find Harmonic mean using Arithmetic mean and Geometric mean](https://www.geeksforgeeks.org/find-harmonic-mean-using-arithmetic-mean-geometric-mean/)
9. [Double Base Palindrome](https://www.geeksforgeeks.org/double-base-palindrome/)
10. [Program for Derivative of a Polynomial](https://www.geeksforgeeks.org/program-derivative-polynomial/)
11. [Sgn value of a polynomial](https://www.geeksforgeeks.org/sgn-value-polynomial/)
12. [Check if a number is a power of another number](https://www.geeksforgeeks.org/check-if-a-number-is-power-of-another-number/)
13. [Program to evaluate simple expressions](https://www.geeksforgeeks.org/program-evaluate-simple-expressions/)
14. [Make a fair coin from a biased coin](https://www.geeksforgeeks.org/print-0-and-1-with-50-probability/)
15. [Implement \*, – and / operations using only + arithmetic operator](https://www.geeksforgeeks.org/implement-and-operations-using-only-arithmetic-operator/)

**Easy Questions:**

**Interquartile Range (IQR)**

The **quartiles** of a ranked set of data values are three points which divide the data into exactly four equal parts, each part comprising of quarter data.

1. **Q1** is defined as the middle number between the smallest number and the median of the data set.
2. **Q2** is the [**median**](https://www.geeksforgeeks.org/mean-median-unsorted-array/) of the data.
3. **Q3** is the middle value between the median and the highest value of the data set.

The interquartile range IQR tells us the range   
where the bulk of the values lie. The interquartile   
range is calculated by subtracting the first quartile  
from the third quartile.   
IQR = Q3 - Q1

**Uses**

**1.** Unlike range, IQR tells where the majority of data lies and is thus preferred over range.

**2.** IQR can be used to identify [outliers](http://www.dictionary.com/browse/outlier) in a data set.

**3.** Gives the central tendency of the data.

**Examples:**

Input : 1, 19, 7, 6, 5, 9, 12, 27, 18, 2, 15  
Output : 13  
The data set after being sorted is   
1, 2, 5, 6, 7, 9, 12, 15, 18, 19, 27  
As mentioned above Q2 is the median of the data.   
Hence Q2 = 9  
Q1 is the median of lower half, taking Q2 as pivot.  
So Q1 = 5  
Q3 is the median of upper half talking Q2 as pivot.   
So Q3 = 18  
Therefore IQR for given data=Q3-Q1=18-5=13

Input : 1, 3, 4, 5, 5, 6, 7, 11  
Output : 3

# Python3 program to find IQR of

# a data set

# Function to give index of the median

**def** median(a, l, r):

    n **=** r **-** l **+** 1

    n **=** (n **+** 1) **//** 2 **-** 1

**return** n **+** l

# Function to calculate IQR

**def** IQR(a, n):

    a.sort()

    # Index of median of entire data

    mid\_index **=** median(a, 0, n)

    # Median of first half

    Q1 **=** a[median(a, 0, mid\_index)]

    # Median of second half

    Q3 **=** a[mid\_index **+** median(a, mid\_index **+** 1, n)]

    # IQR calculation

**return** (Q3 **-** Q1)

# Driver Function

**if** \_\_name\_\_**==**'\_\_main\_\_':

    a **=** [1, 19, 7, 6, 5, 9, 12, 27, 18, 2, 15]

    n **=** len(a)

    print(IQR(a, n))

# This code is contributed by

# Sanjit\_Prasad

**Output:**

13

**Time Complexity:**O(1)

**Auxiliary Space:**O(1)

**Simulated Annealing**

**Problem :** Given a cost function *f: R^n –> R*, find an *n*-tuple that minimizes the value of *f*. Note that minimizing the value of a function is algorithmically equivalent to maximization (since we can redefine the cost function as 1-f).

Many of you with a background in calculus/analysis are likely familiar with simple optimization for single variable functions. For instance, the function *f(x) = x^2 + 2x* can be optimized setting the first derivative equal to zero, obtaining the solution *x = -1* yielding the minimum value *f(-1) = -1*. This technique suffices for simple functions with few variables. However, it is often the case that researchers are interested in optimizing functions of several variables, in which case the solution can only be obtained computationally.

One excellent example of a difficult optimization task is the chip floor planning problem. Imagine you’re working at Intel and you’re tasked with designing the layout for an integrated circuit. You have a set of modules of different shapes/sizes and a fixed area on which the modules can be placed. There are a number of objectives you want to achieve: maximizing ability for wires to connect components, minimize net area, minimize chip cost, etc. With these in mind, you create a cost function, taking all, say, *1000* variable configurations and returning a single real value representing the ‘cost’ of the input configuration. We call this the objective function, since the goal is to minimize its value.

A naive algorithm would be a complete space search — we search all possible configurations until we find the minimum. This may suffice for functions of few variables, but the problem we have in mind would entail such a brute force algorithm to fun in *O(n!)*.

Due to the computational intractability of problems like these, and other NP-hard problems, many optimization heuristics have been developed in an attempt to yield a good, albeit potentially suboptimal, value. In our case, we don’t necessarily need to find a strictly optimal value — finding a near-optimal value would satisfy our goal. One widely used technique is simulated annealing, by which we introduce a degree of stochasticity, potentially shifting from a better solution to a worse one, in an attempt to escape local minima and converge to a value closer to the global optimum.

Simulated annealing is based on metallurgical practices by which a material is heated to a high temperature and cooled. At high temperatures, atoms may shift unpredictably, often eliminating impurities as the material cools into a pure crystal. This is replicated via the simulated annealing optimization algorithm, with energy state corresponding to current solution.

In this algorithm, we define an initial temperature, often set as 1, and a minimum temperature, on the order of 10^-4. The current temperature is multiplied by some fraction alpha and thus decreased until it reaches the minimum temperature. For each distinct temperature value, we run the core optimization routine a fixed number of times. The optimization routine consists of finding a neighboring solution and accepting it with probability *e^(f(c) – f(n))* where *c* is the current solution and *n* is the neighboring solution. A neighboring solution is found by applying a slight perturbation to the current solution. This randomness is useful to escape the common pitfall of optimization heuristics — getting trapped in local minima. By potentially accepting a less optimal solution than we currently have, and accepting it with probability inverse to the increase in cost, the algorithm is more likely to converge near the global optimum. Designing a neighbor function is quite tricky and must be done on a case by case basis, but below are some ideas for finding neighbors in locational optimization problems.

* Move all points 0 or 1 units in a random direction
* Shift input elements randomly
* Swap random elements in input sequence
* Permute input sequence
* Partition input sequence into a random number of segments and permute segments

**Pseudo Random Number Generator (PRNG)**

**Pseudo Random Number Generator(PRNG)** refers to an algorithm that uses mathematical formulas to produce sequences of random numbers. PRNGs generate a sequence of numbers approximating the properties of random numbers. A PRNG starts from an arbitrary starting state using a **seed state**. Many numbers are generated in a short time and can also be reproduced later, if the starting point in the sequence is known. Hence, the numbers are **deterministic and efficient**.

**Why do we need PRNG?**

With the advent of computers, programmers recognized the need for a means of introducing randomness into a computer program. However, surprising as it may seem, it is difficult to get a computer to do something by chance as computer follows the given instructions blindly and is therefore completely predictable. It is not possible to generate truly random numbers from deterministic thing like computers so PRNG is a technique developed to generate random numbers using a computer.

**How PRNG works?**

[Linear Congruential Generator](https://en.wikipedia.org/wiki/Linear_congruential_generator) is most common and oldest algorithm for generating pseudo-randomized numbers. The generator is defined by the recurrence relation:

**Xn+1 = (aXn + c) mod m**  
where X is the sequence of pseudo-random values  
m, 0 < m - modulus   
a, 0 < a < m - multiplier  
c, 0 ≤ c < m - increment  
x0, 0 ≤ x0 < m - the seed or start value

We generate the next random integer using the previous random integer, the integer constants, and the integer modulus. To get started, the algorithm requires an initial Seed, which must be provided by some means. The appearance of randomness is provided by performing**modulo arithmetic.**.

**Characteristics of PRNG**

* **Efficient:** PRNG can produce many numbers in a short time and is advantageous for applications that need many numbers
* **Deterministic:** A given sequence of numbers can be reproduced at a later date if the starting point in the sequence is known.Determinism is handy if you need to replay the same sequence of numbers again at a later stage.
* **Periodic:** PRNGs are periodic, which means that the sequence will eventually repeat itself. While periodicity is hardly ever a desirable characteristic, modern PRNGs have a period that is so long that it can be ignored for most practical purposes

**Applications of PRNG**

PRNGs are suitable for applications where many random numbers are required and where it is useful that the same sequence can be replayed easily. Popular examples of such applications are**simulation and modeling applications**. PRNGs are not suitable for applications where it is important that the numbers are really unpredictable, such as **data encryption and gambling.**

**Pseudo Random Number Generator using**[**srand()**](https://www.geeksforgeeks.org/rand-and-srand-in-ccpp/)

# Python3 code to implement the

# approach

**import** random

**from** datetime **import** datetime

# Passing the current time as the seed value

random.seed(datetime.now())

**for** i **in** range(5):

    print(random.randint(0, 10), end**=**"\t")

# This code is contributed by phasing17

Output 1:

3 7 0 9 8

Output 2:

7 6 8 1 4

**Time Complexity:**O(1)

**Auxiliary Space:**O(1)

**Square root of a number using log**

For a given number find the square root using log function. Number may be int, float or double.

**Examples:**

***Input  :****n = 9*

***Output :****3*

***Input  :****n = 2.93*

***Output :****1.711724*

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

We can find square root of a number using sqrt() method:

# Python3 program to demonstrate finding

# square root of a number using sqrt()

**import** math

**if** \_\_name\_\_**==**'\_\_main\_\_':

    n **=** 12

    print(math.sqrt(n))

# This code is contributed by

# Sanjit\_Prasad

**Output**

3.464102

**Time complexity:** O(log2n), for using sqrt() function.

**Auxiliary space:** O(1)

**We can also find square root using log2() library function:**

# Python program to demonstrate finding

# square root of a number using sqrt()

**import** math

# function to return squareroot

**def** squareRoot(n):

**return** pow(2, 0.5 **\*** math.log2(n))

# Driver program

n **=** 12

print(squareRoot(n))

# This code is contributed by

# Sanjit\_Prasad

**Output**

3.464102

**Time complexity:** O(log2log2N), complexity of using log(N) is log(logN), and pow(x,N) is log(N), so pow(2,0.5\*log(n)) will be log(logN).

**Auxiliary space:** O(1)

**How does the above program work?**

let d be our answer for input number n  
 then n(1/2) = d   
 apply log2 on both sides  
 log2(n(1/2)) = log2(d)  
 log2(d) = 1/2 \* log2(n)  
 d = 2(1/2 \* log2(n))   
 d = pow(2, 0.5\*log2(n))

**Find ways an Integer can be expressed as sum of n-th power of unique natural numbers**

Given two numbers x and n, find a number of ways x can be expressed as sum of n-th power of unique natural numbers.

**Examples :**

***Input  :****x = 10, n = 2*

***Output :****1*

***Explanation:****10 = 12 + 32, Hence total 1 possibility*

***Input  :****x = 100, n = 2*

***Output :****3*

*E****xplanation:***

*100 = 102 OR 62 + 82 OR 12 + 32 + 42 + 52 + 72 Hence total 3 possibilities*

Express as sum of power of natural numbers

The idea is simple. We iterate through all number starting from 1. For every number, we recursively try all greater numbers and if we are able to find sum, we increment result

# Python3 program to count number of ways any

# given integer x can be expressed as n-th

# power of unique natural numbers.

# Function to calculate and return the

# power of any given number

**def** power(num, n):

**if**(n **==** 0):

**return** 1

**elif**(n **%** 2 **==** 0):

**return** power(num, n **//** 2) **\*** power(num, n **//** 2)

**else**:

**return** num **\*** power(num, n **//** 2) **\*** power(num, n **//** 2)

# Function to check power representations recursively

**def** checkRecursive(x, n, curr\_num**=**1, curr\_sum**=**0):

    # Initialize number of ways to express

    # x as n-th powers of different natural

    # numbers

    results **=** 0

    # Calling power of 'i' raised to 'n'

    p **=** power(curr\_num, n)

**while**(p **+** curr\_sum < x):

        # Recursively check all greater values of i

        results **+=** checkRecursive(x, n, curr\_num **+** 1, p **+** curr\_sum)

        curr\_num **=** curr\_num **+** 1

        p **=** power(curr\_num, n)

    # If sum of powers is equal to x

    # then increase the value of result.

**if**(p **+** curr\_sum **==** x):

        results **=** results **+** 1

    # Return the final result

**return** results

# Driver Code.

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    x **=** 10

    n **=** 2

    print(checkRecursive(x, n))

# This code is contributed by

# Sanjit\_Prasad

**Output**

1

**Alternate Solution :**

Below is an alternate simpler solution provided by **Shivam Kanodia**.

# Python 3 program to find number of ways to express

# a number as sum of n-th powers of numbers.

**def** checkRecursive(num, rem\_num, next\_int, n, ans**=**0):

**if** (rem\_num **==** 0):

        ans **+=** 1

    r **=** int(num**\*\***(1 **/** n))

**for** i **in** range(next\_int **+** 1, r **+** 1):

        a **=** rem\_num **-** int(i**\*\***n)

**if** a >**=** 0:

            ans **+=** checkRecursive(num, rem\_num **-** int(i**\*\***n), i, n, 0)

**return** ans

# Wrapper over checkRecursive()

**def** check(x, n):

**return** checkRecursive(x, x, 0, n)

# Driver Code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    print(check(10, 2))

# This code is contributed by

# Surendra\_Gangwar

**Output**

1

**Simple Recursive Solution:**

contributed by[**Ram Jondhale**](https://www.linkedin.com/in/ram-jondhale-a98255135/)**.**

# Helper function

**def** getAllWaysHelper(remainingSum, power, base):

    # calculate power

    result **=** pow(base, power)

**if**(remainingSum **==** result):

**return** 1

**if**(remainingSum < result):

**return** 0

    # Two recursive calls one to include

    # current base's power in sum another to exclude

    x **=** getAllWaysHelper(remainingSum **-** result, power, base **+** 1)

    y **=** getAllWaysHelper(remainingSum, power, base**+**1)

**return** x **+** y

**def** getAllWays(sum, power):

**return** getAllWaysHelper(sum, power, 1)

# Driver Code

x,n **=** 10,2

print(getAllWays(x, n))

# This code is contributed by shinjanpatra.

**Output**

1

**N-th root of a number**

Given two numbers N and A, find N-th root of A. In mathematics, Nth root of a number A is a real number that gives A, when we raise it to integer power N. These roots are used in Number Theory and other advanced branches of mathematics.

Refer [Wiki](https://en.wikipedia.org/wiki/Nth_root) page for more information.

**Examples:**

Input : A = 81  
 N = 4  
Output : 3   
3^4 = 81

Find Nth root of M

As this problem involves a real valued function A^(1/N) we can solve this using [Newton’s method](https://www.geeksforgeeks.org/program-for-newton-raphson-method/), which starts with an initial guess and iteratively shift towards the result.

We can derive a relation between two consecutive values of iteration using Newton’s method as follows,

according to newton’s method  
x(K+1) = x(K) – f(x) / f’(x)   
here f(x) = x^(N) – A  
so f’(x) = N\*x^(N - 1)  
and x(K) denoted the value of x at Kth iteration  
putting the values and simplifying we get,  
x(K + 1) = (1 / N) \* ((N - 1) \* x(K) + A / x(K) ^ (N - 1))

Using above relation, we can solve the given problem. In below code we iterate over values of x, until difference between two consecutive values of x become lower than desired accuracy.

Below is the implementation of above approach:

# Python3 program to calculate

# Nth root of a number

**import** math

**import** random

# method returns Nth power of A

**def** nthRoot(A,N):

    # initially guessing a random number between

    # 0 and 9

    xPre **=** random.randint(1,101) **%** 10

    #  smaller eps, denotes more accuracy

    eps **=** 0.001

    # initializing difference between two

    # roots by INT\_MAX

    delX **=** 2147483647

    #  xK denotes current value of x

    xK**=**0.0

    #  loop until we reach desired accuracy

**while** (delX > eps):

        # calculating current value from previous

        # value by newton's method

        xK **=** ((N **-** 1.0) **\*** xPre **+**

              A**/**pow(xPre, N**-**1)) **/**N

        delX **=** abs(xK **-** xPre)

        xPre **=** xK;

**return** xK

# Driver code

N **=** 4

A **=** 81

nthRootValue **=** nthRoot(A, N)

print("Nth root is ", nthRootValue)

## Acalc = pow(nthRootValue, N);

## print("Error in difference of powers ",

##             abs(A - Acalc))

# This code is contributed

# by Anant Agarwal.

**Output:**

Nth root is 3

**Fast Fourier Transformation for polynomial multiplication**

Given two polynomial A(x) and B(x), find the product C(x) = A(x)\*B(x). There is already an O(

) naive approach to solve this problem. [here](https://www.geeksforgeeks.org/multiply-two-polynomials-2/). This approach uses the coefficient form of the polynomial to calculate the product.

A coefficient representation of a polynomial

is a = a0, a1, …, an-1.

**Example-**

Coefficient representation of A(x) = (9, -10, 7, 6)

Coefficient representation of B(x) = (-5, 4, 0, -2)

Input :  
 A[] = {9, -10, 7, 6}  
 B[] = {-5, 4, 0, -2}  
Output :

We can do better, if we represent the polynomial in another form.

yes

Idea is to represent polynomial in point-value form and then compute the product. A point-value representation of a polynomial A(x) of degree-bound n is a set of n point-value pairs is{ (x0, y0), (x1, y1), …, (xn-1, yn-1)} such that all of the xi are distinct and yi = A(xi) for i = 0, 1, …, n-1.

Example

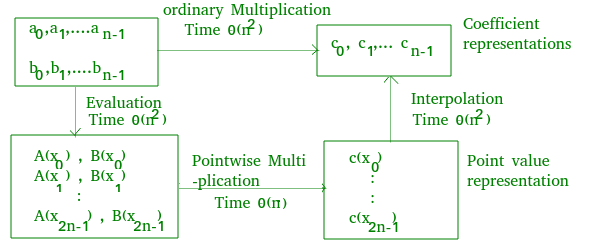
xi -- 0, 1, 2, 3 A(xi) -- 1, 0, 5, 22

Point-value representation of above polynomial is { (0, 1), (1, 0), (2, 5), (3, 22) }. Using Horner’s method, ([discussed here](https://www.geeksforgeeks.org/horners-method-polynomial-evaluation/)), n-point evaluation takes time O(

). It’s just calculation of values of A(x) at some x for n different points, so time complexity is O(

). Now that the polynomial is converted into point value, it can be easily calculated C(x) = A(x)\*B(x) again using horner’s method. This takes O(n) time. An important point here is C(x) has degree bound 2n, then n points will give only n points of C(x), so for that case we need 2n different values of x to calculate 2n different values of y. Now that the product is calculated, the answer can to be converted back into coefficient vector form. To get back to coefficient vector form we use inverse of this evaluation. The inverse of evaluation is called interpolation. Interpolation using Lagrange’s formula gives point value-form to coefficient vector form of the polynomial.Lagrange’s formula is –

So far we discussed,



.

This idea still solves the problem in O(

) time complexity. We can use any points we want as evaluation points, but by choosing the evaluation points carefully, we can convert between representations in only O(n log n) time. If we choose “complex roots of unity” as the evaluation points, we can produce a point-value representation by taking the discrete Fourier transform (DFT) of a coefficient vector. We can perform the inverse operation, interpolation, by taking the “inverse DFT” of point-value pairs, yielding a coefficient vector. Fast Fourier Transform (FFT) can perform DFT and inverse DFT in time O(nlogn).

**DFT**

DFT is evaluating values of polynomial at n complex nth roots of unity

. So, for

k = 0, 1, 2, …, n-1, y = (y0, y1, y2, …, yn-1) is Discrete fourier Transformation (DFT) of given polynomial.

The product of two polynomials of degree-bound n is a polynomial of degree-bound 2n. Before evaluating the input polynomials A and B, therefore, we first double their degree-bounds to 2n by adding n high-order coefficients of 0. Because the vectors have 2n elements, we use “complex 2nth roots of unity, ” which are denoted by the W2n (omega 2n). We assume that n is a power of 2; we can always meet this requirement by adding high-order zero coefficients.

**FFT**

Here is the Divide-and-conquer strategy to solve this problem.

Define two new polynomials of degree-bound n/2, using even-index and odd-index coefficients of A(x) separately

The problem of evaluating A(x) at

reduces to evaluating the degree-bound n/2 polynomials A0(x) and A1(x) at the points

Now combining the results by

The list

does not contain n distinct values, but n/2 complex n/2th roots of unity. Polynomials A0 and A1 are recursively evaluated at the n complex nth roots of unity. Subproblems have exactly the same form as the original problem, but are half the size. So recurrence formed is T(n) = 2T(n/2) + O(n), i.e complexity O(nlogn).

Algorithm  
1. Add n higher-order zero coefficients to A(x) and B(x)  
2. Evaluate A(x) and B(x) using FFT for 2n points  
3. Pointwise multiplication of point-value forms  
4. Interpolate C(x) using FFT to compute inverse DFT

Pseudo code of recursive FFT

Recursive\_FFT(a){  
n = length(a) // a is the input coefficient vector  
if n = 1  
 then return a

// wn is principle complex nth root of unity.  
wn = e^(2\*pi\*i/n)  
w = 1

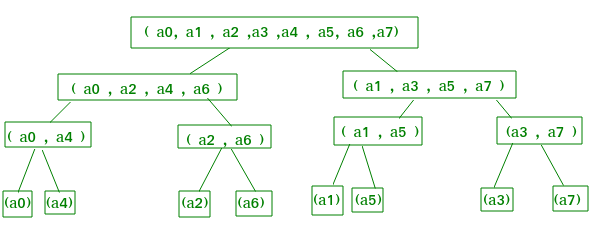
// even indexed coefficients  
A0 = (a0, a2, ..., an-2 )

// odd indexed coefficients  
A1 = (a1, a3, ..., an-1 )

y0 = Recursive\_FFT(A0) // local array  
y1 = Recursive-FFT(A1) // local array

for k = 0 to n/2 - 1

// y array stores values of the DFT   
 // of given polynomial.   
 do y[k] = y0[k] + w\*y1[k]   
 y[k+(n/2)] = y0[k] - w\*y1[k]  
 w = w\*wn  
return y  
}  
Recursion Tree of Above Execution-



Why does this work?

since,

Thus, the vector y returned by Recursive-FFT is indeed the DFT of the input

vector a.

**from** math **import** sin,cos,pi

# Recursive function of FFT

**def** fft(a):

    n **=** len(a)

    # if input contains just one element

**if** n **==** 1:

**return** [a[0]]

    # For storing n complex nth roots of unity

    theta **= -**2**\***pi**/**n

    w **=** list( complex(cos(theta**\***i), sin(theta**\***i)) **for** i **in** range(n) )

    # Separe coefficients

    Aeven **=** a[0::2]

    Aodd  **=** a[1::2]

    # Recursive call for even indexed coefficients

    Yeven **=** fft(Aeven)

    # Recursive call for odd indexed coefficients

    Yodd **=** fft(Aodd)

    # for storing values of y0, y1, y2, ..., yn-1.

    Y **=** [0]**\***n

    middle **=** n**//**2

**for** k **in** range(n**//**2):

        w\_yodd\_k  **=** w[k] **\*** Yodd[k]

        yeven\_k   **=**  Yeven[k]

        Y[k]          **=**  yeven\_k  **+**  w\_yodd\_k

        Y[k **+** middle] **=**  yeven\_k  **-**  w\_yodd\_k

**return** Y

# Driver code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    a **=** [1, 2, 3, 4]

    b **=** fft(a)

**for** B **in** b:

        print(B)

Input: 1 2 3 4  
Output:  
(10, 0)  
(-2, 2)  
(-2, 0)  
(-2,-2)

**Interpolation**

Switch the roles of a and y.

Replace wn by wn^-1.

Divide each element of the result by n.

Time Complexity: O(nlogn).

**Find Harmonic mean using Arithmetic mean and Geometric mean**

Given two numbers, first calculate arithmetic mean and geometric mean of these two numbers. Using the arithmetic mean and geometric mean so calculated, find the harmonic mean between the two numbers.

Examples:

Input : a = 2  
 b = 4  
Output : 2.666

Input : a = 5  
 b = 15  
Output : 7.500

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

[Arithmetic Mean:](https://en.wikipedia.org/wiki/Arithmetic_mean) Arithmetic Mean ‘AM’ between two numbers a and b is such a number that AM-a = b-AM. Thus, if we are given these two numbers, the arithmetic mean AM = 1/2(a+b)

[Geometric Mean:](https://en.wikipedia.org/wiki/Geometric_mean) Geometric Mean ‘GM’ between two numbers a and b is such a number that GM/a = b/GM. Thus, if we are given these two numbers, the geometric mean GM = sqrt(a\*b)

[Harmonic Mean:](https://en.wikipedia.org/wiki/Harmonic_mean) Harmonic Mean ‘HM’ between two numbers a and b is such a number that 1/HM – 1/a = 1/b – 1/HM. Thus, if we are given these two numbers, the harmonic mean HM = 2ab/a+b

Now, we also know that

# Python 3 implementation of computation

# of arithmetic mean, geometric mean

# and harmonic mean

**import** math

# Function to calculate arithmetic

# mean, geometric mean and harmonic mean

**def** compute( a, b) :

    AM **=** (a **+** b) **/** 2

    GM **=** math.sqrt(a **\*** b)

    HM **=** (GM **\*** GM) **/** AM

**return** HM

# Driver function

a **=** 5

b **=** 15

HM **=** compute(a, b)

**print**("Harmonic Mean between " , a,

      " and ", b , " is " , HM )

# This code is contributed by Nikita Tiwari.

**Output:**

Harmonic Mean between 5 and 15 is 7.500

***Time Complexity:*** O(log(a\*b)), for using sqrt function where a and b represents the given integers.

***Auxiliary Space:***O(1), no extra space is required, so it is a constant.

**Double Base Palindrome**

Double base Palindrome as the name suggest is a number which is Palindrome in 2 bases. One of the base is 10 i.e. decimal and another base is k.(which can be 2 or others).

**Note :** The palindromic number, in either base, may not include leading zeros.

**Example :** The decimal number, 585 = 10010010012 (binary), is palindromic in both bases.

A **Palindrome** is a word, phrase, number, or other sequence of characters which reads the same backward as forward, such as madam or 12321.

Find the sum of all numbers less than n which are palindromic in base 10 and base k.

**Examples:**

Input : 10 2  
Output : 25  
Explanation : (here n = 10 and k = 2)  
 1 3 5 7 9 (they are all palindrome   
 in base 10 and 2) so sum is :  
 1 + 3 + 5 + 7 + 9 = 25

Input : 100 2  
Output : 157  
Explanation : 1 + 3 + 5 + 7 + 9 + 33 + 99 = 157

**Method 1 :**This method is simple. For every number less than n :

* Check if it is a palindrome in base 10
* If yes, then convert it into base k
* Check if it is palindrome in base k
* If yes, then add it in sum.

This method is quite lengthy as it checks for every number whether it is a palindrome or not. So, for number as large as 1000000, it checks for every number.

If k = 2, then a palindrome in base 2 can only be odd number, which might reduce the comparisons to 1000000 / 2 = 500000 (which is still large).

Below is the implementation of the above approach :

# Python3 Program for Checking

# double base Palindrome.

# converts number to base

# k by changing it into string.

**def** integer\_to\_string(n, base):

    str **=** "";

**while** (n > 0):

        digit **=** n **%** base;

        n **=** int(n **/** base);

        str **=** chr(digit **+** ord('0')) **+** str;

**return** str;

# function to check for palindrome

**def** isPalindrome(i, k):

    temp **=** i;

    # m stores reverse of a number

    m **=** 0;

**while** (temp > 0):

        m **=** (temp **%** 10) **+** (m **\*** 10);

        temp **=** int(temp **/** 10);

    # if reverse is equal to number

**if** (m **==** i):

        # converting to base k

        str **=** integer\_to\_string(m, k);

        str1 **=** str;

        # reversing number in base k

        # str=str[::-1];

        # checking palindrome

        # in base k

**if** (str[::**-**1] **==** str1):

**return** i;

**return** 0;

# function to find sum of palindromes

**def** sumPalindrome(n, k):

    sum **=** 0;

**for** i **in** range(n):

        sum **+=** isPalindrome(i, k);

**print**("Total sum is", sum);

# Driver code

n **=** 100;

k **=** 2;

sumPalindrome(n, k);

# This code is contributed

# by mits

**Output:**

Total sum is 157

**Method 2 :**This method is a little complex to understand but more advance than method 1. Rather than checking palindrome for two bases. This method generates palindrome in given range.

Suppose we have a palindrome of the form **123321** in base k, then the first 3 digits define the palindrome. However, the 3 digits **123** also define the palindrome **12321**. So the 3-digit number **123** defines a 5-digit palindrome and a 6 digit palindrome. From which follows that every positive number less than kn generates two palindromes less than k2n . This holds for every base k. Example : let’s say k = 10 that is decimal. Then for n = 1, all numbers less than 10n have 2 palindrome, 1 even length and 1 odd length in 102n. These are 1, 11 or 2, 22 or 3, 33 and so on. So, for 1000000 we generate around 2000 and for 108 we generate around 20000 palindromes.

* Start from i=1 and generate odd palindrome of it.
* Check if this generated odd palindrome is also palindrome in base k
* If yes, then add this number to sum.
* Repeat the above three steps by changing i=i+1 until the last generated odd palindrome has crossed limit.
* Now, again start from i=1 and generate even palindrome of it.
* Check if this generated even palindrome is also palindrome in base k
* If yes, then add this number to sum.
* Repeat the above three steps by changing i=i+1 until the last generated even palindrome has crossed limit.

Below is the implementation of the above approach :

# Python3 Program for Checking double

# base Palindrome.

# Function generates even and

# odd palindromes

**def** makePalindrome(n, odd):

    res **=** n;

**if** (odd):

        n **=** int(n **/** 10);

**while** (n > 0):

        res **=** 10 **\*** res **+** n **%** 10;

        n **=** int(n **/** 10);

**return** res;

# Check if a number is palindrome

# in base k

**def** isPalindrome(n, base):

    reversed **=** 0;

    temp **=** n;

**while** (temp > 0):

        reversed **=** reversed **\*** base **+** temp **%** base;

        temp **=** int(temp **/** base);

**return** reversed **==** n;

# function to print sum of Palindromes

**def** sumPalindrome(n, k):

    sum **=** 0;

    i **=** 1;

    p **=** makePalindrome(i, True);

    # loop for odd generation of

    # odd palindromes

**while** (p < n):

**if** (isPalindrome(p, k)):

            sum **+=** p;

        i **+=** 1;

        p **=** makePalindrome(i, True);

    i **=** 1;

    # loop for generation of

    # even palindromes

    p **=** makePalindrome(i, False);

**while** (p < n):

**if** (isPalindrome(p, k)):

            sum **+=** p;

        i **+=** 1;

        p **=** makePalindrome(i, False);

    # result of all palindromes in

    # both bases.

**print**("Total sum is", sum);

# Driver code

n **=** 1000000;

k **=** 2;

sumPalindrome(n, k);

# This code is contributed by mits

**Output:**

Total sum is 872187

**Program for Derivative of a Polynomial**

Given a polynomial as a string and a value. Evaluate polynomial’s derivative for the given value.

**Note:**The input format is such that there is a white space between a term and the ‘+’ symbol

*The derivative of p(x) = ax^n is p'(x) = a\*n\*x^(n-1)*

*Also, if p(x) = p1(x) + p2(x)*

*Here p1 and p2 are polynomials too*

*p'(x) = p1′(x) + p2′(x)*

Input : 3x^3 + 4x^2 + 6x^1 + 89x^0  
 2   
Output :58   
Explanation : Derivative of given  
polynomial is : 9x^2 + 8x^1 + 6  
Now put x = 2  
9\*4 + 8\*2 + 6 = 36 + 16 + 6 = 58   
   
Input : 1x^3  
 3  
Output : 27

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

We split the input string into tokens and for each term calculate the derivative separately for each term and add them to get the result.

# Python3 program to find

# value of derivative of

# a polynomial.

**def** derivativeTerm(pTerm, val):

    # Get coefficient

    coeffStr **=** ""

    i **=** 0

**while** (i < len(pTerm) **and**

           pTerm[i] !**=** 'x'):

        coeffStr **+=** (pTerm[i])

        i **+=** 1

    coeff **=** int(coeffStr)

    # Get Power (Skip 2 characters

    # for x and ^)

    powStr **=** ""

    j **=** i **+** 2

**while** j < len(pTerm):

        powStr **+=** (pTerm[j])

        j **+=** 1

    power **=** int(powStr)

    # For ax^n, we return

    # a(n)x^(n-1)

**return** (coeff **\*** power **\***

            pow(val, power **-** 1))

**def** derivativeVal(poly, val):

    ans **=** 0

    i **=** 0

    stSplit **=** poly.split("+")

**while** (i < len(stSplit)):

        ans **=** (ans **+**

               derivativeTerm(stSplit[i],

                              val))

        i **+=** 1

**return** ans

# Driver code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    st **=** "4x^3 + 3x^1 + 2x^2"

    val **=** 2

    print(derivativeVal(st, val))

# This code is contributed by Chitranayal

**Output:**

59

**Sgn value of a polynomial**

Given a polynomial function f(x) = 1+ a1\*x + a2\*(x^2) + … an(x^n). Find the [Sgn](https://en.wikipedia.org/wiki/Sign_function) value of these function, when x is given and all the coefficients also.

If value of polynomial greater than 0  
 Sign = 1  
Else If value of polynomial less than 0  
 Sign = -1  
Else if value of polynomial is 0  
 Sign = 0

**Examples:**

Input: poly[] = [1, 2, 3]   
 x = 1   
Output: 1   
Explanation: f(1) = 6 which is > 0   
hence 1.

Input: poly[] = [1, -1, 2, 3]   
 x = -2   
Output: -1   
Explanation: f(-2)=-11 which is less   
than 0, hence -1.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

A **naive** approach will be to calculate every power of x and then add it to the answer by multiplying it with its coefficient. Calculating power of x will take O(n) time and for n coefficients. Hence, taking the total complexity to O(n \* n), as we will need nested loops for traversing n\*n times.

An **efficient** approach is to use [Horner’s method](https://www.geeksforgeeks.org/horners-method-polynomial-evaluation/). We evaluate value of polynomial using Horner’s method. Then we return value according to sign of the value.

Below is the implementation of the above approach

 # Python3 program to find

# sign value of a

# polynomial

# returns value of poly[0]x(n-1) +

# poly[1]x(n-2) + .. + poly[n-1]

**def** horner( poly, n, x):

    # Initialize result

    result **=** poly[0];

    # Evaluate value of

    # polynomial using

    # Horner's method

**for** i **in** range(1,n):

        result **=** (result **\*** x **+**

                     poly[i]);

**return** result;

# Returns sign value

# of polynomial

**def** findSign(poly, n, x):

    result **=** horner(poly, n, x);

**if** (result > 0):

**return** 1;

**elif** (result < 0):

**return -**1;

**return** 0;

# Driver Code

# Let us evaluate value

# of 2x3 - 6x2

# + 2x - 1 for x = 3

poly **=** [2, **-**6, 2, **-**1];

x **=** 3;

n **=** len(poly);

print("Sign of polynomial is ",

         findSign(poly, n, x));

# This code is contributed by mits

**Output:**

Sign of polynomial is 1

**Time Complexity**: O(N), as we are using a loop to traverse N times.

**Auxiliary Space**: O(1), as we are not using any extra space.

**Check if a number is a power of another number**

Given two positive numbers x and y, check if y is a power of x or not.

**Examples :**

***Input:****x = 10, y = 1*

***Output:****True*

*x^0 = 1*

***Input:****x = 10, y = 1000*

***Output:****True*

*x^3 = 1*

***Input:****x = 10, y = 1001*

***Output:****False*

Recommended Problem

Check if a number is power of another number

A simple solution is to repeatedly compute the powers of x. If a power becomes equal to y, then y is a power, else not.

# python program to check

# if a number is power of

# another number

# Returns true if y is a

# power of x

**def** isPower (x, y):

    # The only power of 1

    # is 1 itself

**if** (x **==** 1):

**return** (y **==** 1)

    # Repeatedly compute

    # power of x

    pow **=** 1

**while** (pow < y):

        pow **=** pow **\*** x

    # Check if power of x

    # becomes y

**return** (pow **==** y)

# Driver Code

# check the result for

# true/false and print.

**if**(isPower(10, 1)):

    print(1)

**else**:

**print**(0)

**if**(isPower(1, 20)):

**print**(1)

**else**:

    print(0)

**if**(isPower(2, 128)):

**print**(1)

**else**:

    print(0)

**if**(isPower(2, 30)):

    print(1)

**else**:

    print(0)

# This code is contributed

# by Sam007.

**Output**

1  
0  
1  
0

**Time complexity:** O(Logxy)

**Auxiliary space:** O(1)

**Optimization:**

We can optimize above solution to work in O(Log Log y). The idea is to do squaring of power instead of multiplying it with x, i.e., compare y with x^2, x^4, x^8, …etc. If x becomes equal to y, return true. If x becomes more than y, then we do binary search for power of x between previous power and current power, i.e., between x^i and x^(i/2).

Following are detailed step.

1) Initialize pow = x, i = 1  
2) while (pow < y)  
 {  
 pow = pow\*pow   
 i \*= 2  
 }   
3) If pow == y  
 return true;  
4) Else construct an array of powers  
 from x^i to x^(i/2)  
5) Binary Search for y in array constructed  
 in step 4. If not found, return false.   
 Else return true.

**Alternate Solution :**

The idea is to take log of y in base x. If it turns out to be an integer, we return true. Else false.

# Python program to check if given number y

# is power of x

**import** math

**def** is\_power(x, y):

    # logarithm function to calculate value

    res1 **=** math.log(y) **/** math.log(x)

    res2 **=** math.log(y) **/** math.log(x) # Note: this is float

    # compare to the result1 or result2 both are equal

**return** res1 **==** res2

# Driven program

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    print(is\_power(2, 128))

**Output**

1

**Time complexity**: O(1)

**Auxiliary space**: O(1)

**Program to evaluate simple expressions**

You are given a string that represent an expression of digits and operands. E.g. 1+2\*3, 1-2+4. You need to evaluate the string or the expression. NO BODMAS is followed. If the expression is of incorrect syntax return -1.

Test cases:

a) 1+2\*3 will be evaluated to 9.

b) 4-2+6\*3 will be evaluated to 24.

c) 1++2 will be evaluated to -1(INVALID).

Also, in the string spaces can occur. For that case we need to ignore the spaces. Like :- 1\*2 -1 is equals to 1.

Source: [Amazon Interview Question](https://www.geeksforgeeks.org/amazon-interview-set-98-campus/)

**It is strongly recommend to minimize the browser and try this yourself first.**

The idea is simple start from the first character and traverse from left to right and check for errors like two consecutive operators and operands. We also keep track of result and update the result while traversing the expression.

Following is the program to evaluate the given expression.

 # Python3 program to evaluate a

# given expression

# A utility function to check if

# a given character is operand

**def** isOperand(c):

**return** (c >**=** '0' **and** c <**=** '9');

# utility function to find

# value of and operand

**def** value(c):

**return** ord(c) **-** ord('0');

# This function evaluates simple

# expressions. It returns -1 if the

# given expression is invalid.

**def** evaluate(exp):

    len1 **=** len(exp);

    # Base Case: Given expression is empty

**if** (len1 **==** 0):

**return -**1;

    # The first character must be

    # an operand, find its value

    res **=** value(exp[0]);

    # Traverse the remaining

    # characters in pairs

**for** i **in** range(1,len1,2):

        # The next character must be

        # an operator, and next to

        # next an operand

        opr **=** exp[i];

        opd **=** exp[i **+** 1];

        # If next to next character

        # is not an operand

**if** (isOperand(opd)**==**False):

**return -**1;

        # Update result according

        # to the operator

**if** (opr **==** '+'):

            res **+=** value(opd);

**elif** (opr **==** '-'):

            res **-=** int(value(opd));

**elif** (opr **==** '\*'):

            res **\*=** int(value(opd));

**elif** (opr **==** '/'):

            res **/=** int(value(opd));

        # If not a valid operator

**else**:

**return -**1;

**return** res;

# Driver Code

expr1 **=** "1+2\*5+3";

res **=** evaluate(expr1);

print(expr1,"is Invalid") **if** (res **== -**1) **else print**("Value of",expr1,"is",res);

expr2 **=** "1+2\*3";

res **=** evaluate(expr2);

**print**(expr2,"is Invalid") **if** (res **== -**1) **else** print("Value of",expr2,"is",res);

expr3 **=** "4-2+6\*3";

res **=** evaluate(expr3);

print(expr3,"is Invalid") **if** (res **== -**1) **else print**("Value of",expr3,"is",res);

expr4 **=** "1++2";

res **=** evaluate(expr4);

print(expr4,"is Invalid") **if** (res **== -**1) **else** print("Value of",expr4,"is",res);

# This code is contributed by mits

**Output:**

Value of 1+2\*5+3 is 18  
Value of 1+2\*3 is 9  
Value of 4-2+6\*3 is 24  
1++2 is Invalid

Time Complexity: O(|exp|)

Auxiliary Space: O(1)

**Make a fair coin from a biased coin**

You are given a function foo() that represents a biased coin. When foo() is called, it returns 0 with 60% probability, and 1 with 40% probability. Write a new function that returns 0 and 1 with a 50% probability each. Your function should use only foo(), no other library method.

**Solution:**

We know foo() returns 0 with 60% probability. How can we ensure that 0 and 1 are returned with a 50% probability?

The solution is similar to [this](https://www.geeksforgeeks.org/generate-integer-from-1-to-7-with-equal-probability/)post. If we can somehow get two cases with equal probability, then we are done. We call foo() two times. Both calls will return 0 with a 60% probability. So the two pairs (0, 1) and (1, 0) will be generated with equal probability from two calls of foo(). Let us see how.

**(0, 1):** The probability to get 0 followed by 1 from two calls of foo() = 0.6 \* 0.4 = 0.24

**(1, 0):** The probability to get 1 followed by 0 from two calls of foo() = 0.4 \* 0.6 = 0.24

*So the two cases appear with equal probability. The idea is to return consider only the above two cases, return 0 in one case, return 1 in other case. For other cases [(0, 0) and (1, 1)], recur until you end up in any of the above two cases.*

The below program depicts how we can use foo() to return 0 and 1 with equal probability.

# Python3 program for the

# above approach

**def** foo():

    # Some code here

**pass**

# Returns both 0 and 1

# with 50% probability

**def** my\_fun():

    val1, val2 **=** foo(), foo()

**if** val1 ^ val2:

        # Will reach here with

        # (0.24 + 0.24) probability

**return** val1

    # Will reach here with

    # (1 - 0.24 - 0.24) probability

**return** my\_fun()

# Driver Code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    print(my\_fun())

# This code is contributed by sgshah2

***Time Complexity:****O(1)*

***Auxiliary Space:****O(1)*

**Implement \*, – and / operations using only + arithmetic operator**

Given two numbers, perform multiplication, subtraction, and division operations on them, using ‘+’ arithmetic operator only.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Operations can be performed as follows:**

**Subtraction :-**  a - b = a + (-1)\*b.  
**Multiplication :-** a \* b = a + a + a ... b times.  
**Division :-** a / b = continuously subtract b from a and   
 count how many times we can do that.

The above steps look simple, but it is slightly challenging as we can’t even use – to subtract.

# Python3 code to illustrate \*, -, / using

# only  '+' arithmetic operator

# Function to flip the sign using only "+"

# operator (It is simple with '\*' allowed.

# We need to do a = (-1)\*a

**def** flipSign(a):

    neg **=** 0;

    # If sign is + ve turn it -ve

    # and vice-versa

    tmp **=** 1 **if** a < 0 **else -**1;

**while** (a !**=** 0):

        neg **+=** tmp;

        a **+=** tmp;

**return** neg;

# Check if a and b are of different signs

**def** areDifferentSign(a, b):

**return** ((a < 0 **and** b > 0) **or**

            (a > 0 **and** b < 0));

# Function to subtract two numbers

# by negating b and adding them

**def** sub(a, b):

    # Negating b

**return** a **+** flipSign(b);

# Function to multiply a by b by

# adding a to itself b times

**def** mul(a, b):

    # because algo is faster if b<a

**if** (a < b):

**return** mul(b, a);

    # Adding a to itself b times

    sum **=** 0;

**for** i **in** range(abs(b), 0, **-**1):

        sum **+=** a;

    # Check if final sign must

    # be -ve or + ve

**if** (b < 0):

        sum **=** flipSign(sum);

**return** sum;

# Function to divide a by b by counting

# how many times 'b' can be subtracted

# from 'a' before getting 0

**def** division(a, b):

    quotient **=** 0;

    # Negating b to subtract from a

    divisor **=** flipSign(abs(b));

    # Subtracting divisor from dividend

**for** dividend **in** range(abs(a),

                          abs(divisor) **+** divisor,

                                         divisor):

        quotient **+=** 1;

    # Check if a and b are of similar

    # symbols or not

**if** (areDifferentSign(a, b)):

        quotient **=** flipSign(quotient);

**return** quotient;

# Driver code

**print**("Subtraction is", sub(4, **-**2));

**print**("Product is", mul(**-**9, 6));

a, b **=** 8, 2;

**if**(b):

**print**("Division is", division(a, b));

**else**:

**print**("Exception :- Divide by 0");

# This code is contributed by mits

**Output:**

Subtraction is 6  
Product is -54  
Division is 4

**Related Articles :**

* [Add two numbers without using arithmetic operators](https://www.geeksforgeeks.org/add-two-numbers-without-using-arithmetic-operators/)
* [Subtract two numbers without using arithmetic operators](https://www.geeksforgeeks.org/subtract-two-numbers-without-using-arithmetic-operators/)
* [Multiply two integers without using multiplication, division and bitwise operators, and no loops](https://www.geeksforgeeks.org/multiply-two-numbers-without-using-multiply-division-bitwise-operators-and-no-loops/)