Number Theory

Contents

Prime numbers

A prime number is a natural number greater than **1**, which is only divisible by 1 and itself. First few prime numbers are : 2 3 5 7 11 13 17 19 23 …..

Some interesting fact about Prime numbers

1. Two is the only even prime number.
2. Every prime number can be represented in form of 6n+1 or 6n-1 except the prime number 2 and 3, where n is a natural number.
3. Two & Three are only two consecutive natural numbers that are prime.

Applications

Application 1 -

How to check if a number is prime or not?

Naive approach

A naive solution is to iterate through all numbers from 2 to √n and for every number check if it divides n. If we find any number that divides, we return false.

Implementation

Visit the link - https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Number%20Theory/Codes/Prime%20Numbers/checkIfPrimeOrNot_naiveApproach.cpp

Time Complexity - O(√n)

Recursive approach

Implementation

Visit the link - https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Number%20Theory/Codes/Prime%20Numbers/checkIfPrimeOrNot_recursion.cpp

Time Complexity - O(n)

Space Complexity - O(n)

Application 2 -

Given a number n, print all primes smaller than or equal to n. It is also given that n is a small number.

Introduction

The sieve of Eratosthenes is one of the most efficient ways to find all primes smaller than n when n is smaller than 10 million or s

Explanation with Example

1. Let us take an example when n = 50. So we need to print all prime numbers smaller than or equal to 50.
2. We create a list of all numbers from 2 to 50. According to the algorithm we will mark all the numbers which are divisible by 2 and are greater than or equal to the square of it.
3. Now we move to our next unmarked number 3 and mark all the numbers which are multiples of 3 and are greater than or equal to the square of it.
4. We move to our next unmarked number 5 and mark all multiples of 5 and are greater than or equal to the square of it.

5. So the prime numbers are the unmarked ones: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47.
6. In the implementation, a boolean array arr[] of size n is used to mark multiples of prime numbers.

Implementation

Visit the link - https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Number%20Theory/Codes/Prime%20Numbers/findAllPrimesLessOrEqualToN.cpp

Time Complexity - O(n*log(log(n)))

Space Complexity - O(n)

Square root of a number

Given an integer x, find it's square root. If x is not a perfect square, then return floor(√x).

Approach - 1

To find the floor of the square root, try with all-natural numbers starting from 1. Continue incrementing the number until the square of that number is greater than the given number.

Algorithm

1. Create a variable (counter) i and take care of some base cases, i.e when the given number is 0 or 1.
2. Run a loop until i*i <= n , where n is the given number. Increment i by 1.
3. The floor of the square root of the number is i – 1.

Implementation

Visit the link - https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Number%20Theory/Codes/Square%20Root%20of%20a%20Number/squareRoot_simpleApproach.cpp

Time Complexity - O(√n)

Space Complexity - O(1)

Approach -2

Binary Search

The idea is to find the largest integer i whose square is less than or equal to the given number. The idea is to use binary search to solve the problem. The values of i * i is monotonically increasing, so the problem can be solved using binary search.

Algorithm

1. Take care of some base cases, i.e when the given number is 0 or 1.
2. Create some variables, lowerbound l = 0, upperbound r = n, where n is the given number, mid and ans to store the answer.
3. Run a loop until l <= r , the search space vanishes
4. Check if the square of mid (mid = (l + r)/2) is less than or equal to n, If yes then search for a larger value in second half of search space, i.e l = mid + 1, update ans = mid
5. Else if the square of mid is more than n then search for a smaller value in first half of search space, i.e r = mid – 1
6. Print the value of answer (ans)

Implementation

Visit the link - https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Number%20Theory/Codes/Square%20Root%20of%20a%20Number/squareRoot_BinarySearch.cpp

Time complexity - O(log(n))

Space Complexity - O(1)

Square root of a number with precision

Find square root of number upto given precision.

Approach - 1

Using Binary Search

Approach

1. As the square root of number lies in range 0 <= squareRoot <= number, therefore, initialize start and end as : start = 0, end = number.
2. Compare the square of the mid integer with the given number. If it is equal to the number, the square root is found. Else look for the same in the left or right side depending upon the scenario.
3. Once we are done with finding an integral part, start computing the fractional part.
4. Initialize the increment variable by 0.1 and iteratively compute the fractional part up to P places. For each iteration, the increment changes to 1/10th of its previous value.
5. Finally return the answer computed.

Implementation

Visit the link - https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Number%20Theory/Codes/Square%20Root%20of%20a%20Number/squareRootWithPrecision_BinarySearch.cpp

Time Complexity - The overall time complexity is O(log(n) + precision) which is approximately equal to O(log(n)).

Space Complexity - O(1)

Approach - 2

Newton–Raphson method OR Babylonian method

Algorithm

1. Start with an arbitrary positive start value x (the closer to the root, the better).

2. Initialize y = 1.
3. Do the following until desired approximation is achieved.
4. Get the next approximation for the root using the average of x and y.
5. Set y = n/x

Implementation

Visit the link - https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Number%20Theory/Codes/Square%20Root%20of%20a%20Number/squareRootWithPrecision_NewtonRaphson.cpp

Time Complexity - O(√n)

Space Complexity - O(1)