



Explore | Expand | Enrich



Explore | Expand | Enrich



Explore | Expand | Enrich

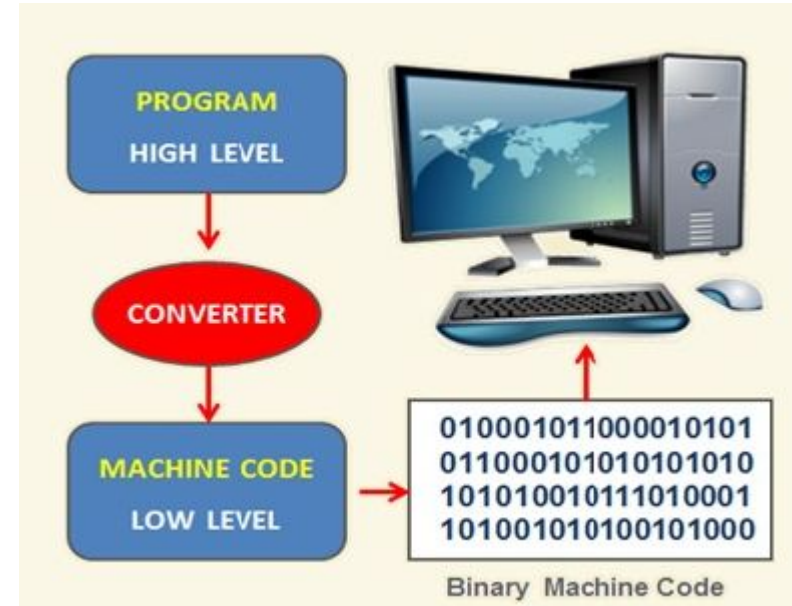
Helping Students
to Realize Their Dreams
for the Last 14 Years

Introduction to Computer Programs

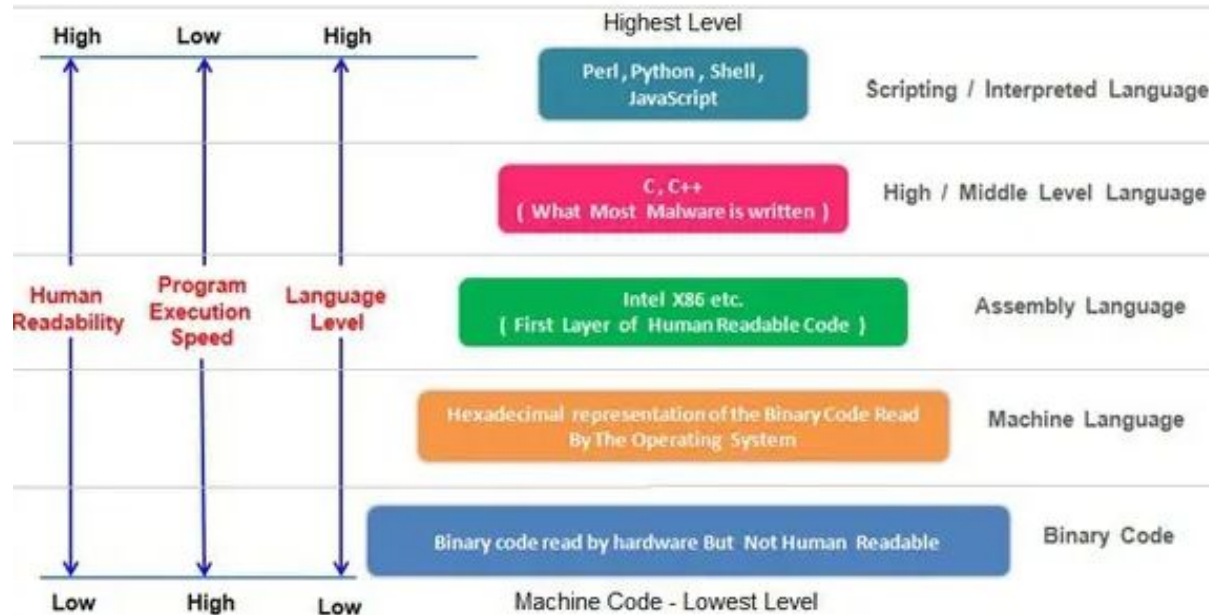
It is a step by step process of designing and developing various sets of computer programs to accomplish a specific computing outcome.

Elements of Computer Programs

- Programming Environment
- Data Types
- Variables
- Keywords
- Logical and Arithmetical Operators
- If else conditions
- Loops
- Numbers, Characters and Arrays
- Functions
- Input and Output Operations



Types of Programming Languages

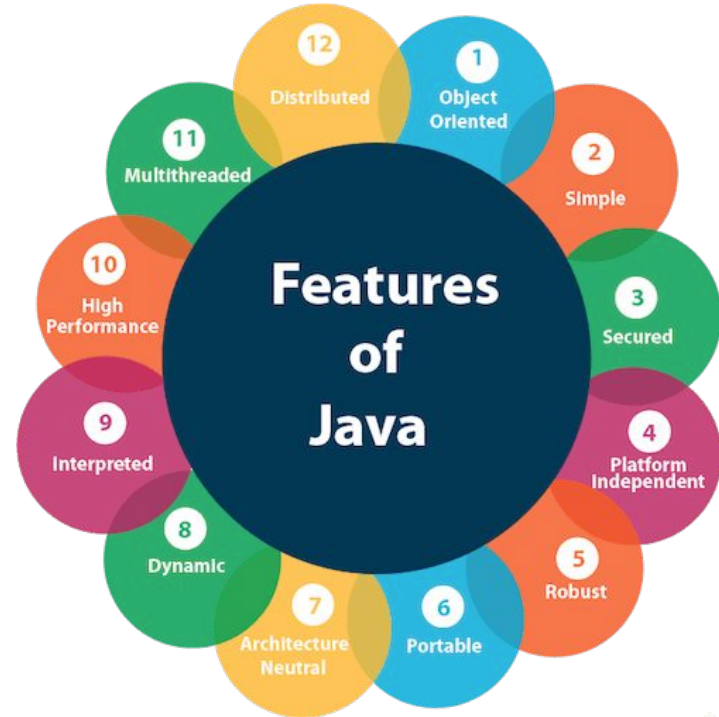


Competitive Programming

- Competitive programming is a mind sport usually held over the Internet or a local network, involving participants trying to program according to provided specifications.
- One of the oldest contests known is ICPC which originated in the 1970s, and has grown to include 88 countries in its 2011 edition.
- The aim of competitive programming is to write source code of computer programs which are able to solve given problems.
- Typical such tasks belong to one of the following categories: combinatorics, number theory, graph theory, algorithmic game theory, computational geometry, string analysis and data structures
- Competitive programming is typically the first or second round of hiring for SDE roles

Introduction to Java

- Java is a **programming language** and a **platform**. It is a high level, secure, robust and an Object Oriented programming language
- The syntax of Java is based on C
- There are no pointers in Java and there is automatic garbage collection
- The features of Java are as given in the diagram



Getting Started with Java



Explore | Expand | Enrich

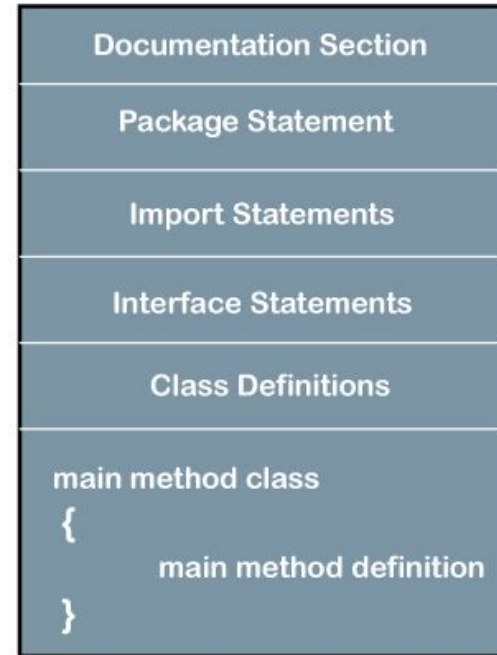
Example and Structure

```
JavaExample.java
1 package com.beginnersbook;
2 import java.util.Scanner;
3 public class JavaExample
4 {
5     public static void main(String args[])
6     {
7         float p, r, t, sinterest;
8         Scanner scan = new Scanner(System.in);
9         System.out.print("Enter the Principal : ");
10        p = scan.nextFloat();
11        System.out.print("Enter the Rate of interest : ");
12        r = scan.nextFloat();
13        System.out.print("Enter the Time period : ");
14        t = scan.nextFloat();
15        scan.close();
16        sinterest = (p * r * t) / 100;
17        System.out.print("Simple Interest is: " + sinterest);
18    }
19 }
```

Problems @ Javadoc Declaration Console Progress Cover

<terminated> JavaExample [Java Application] /Library/Java/JavaVirtualMachines/jdk-9.0.4

Enter the Principal : 2000
Enter the Rate of interest : 6
Enter the Time period : 3
Simple Interest is: 360.0



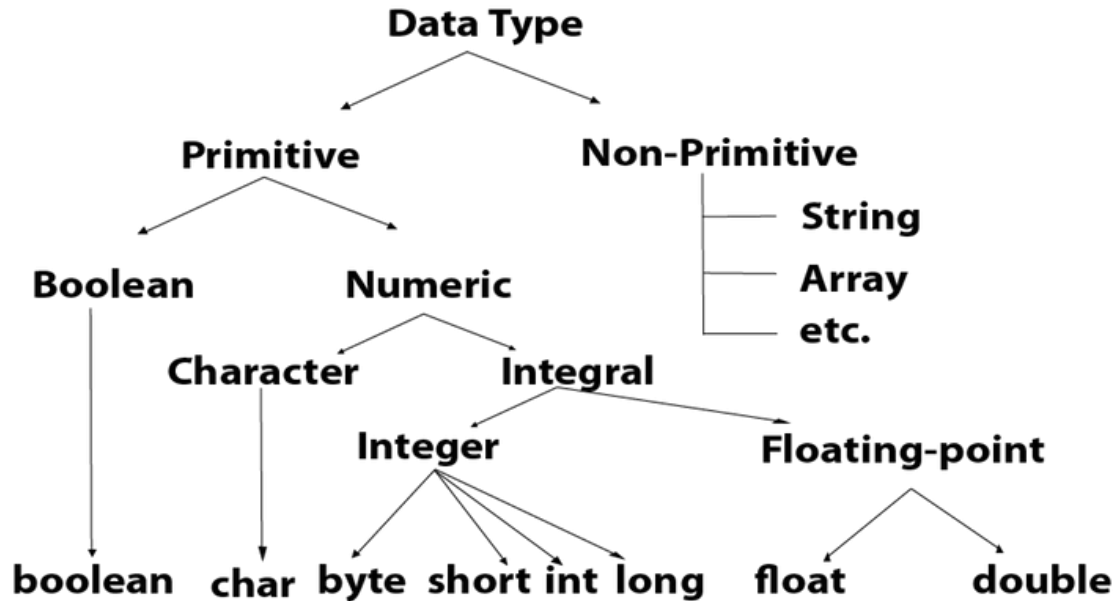
Structure of Java Program

Java in a nutshell

- Data Types: boolean, int, float, char
- I/O: System.out and Scanner
- Operators: Unary, Arithmetic etc.
- Decision Statements: If, else, if-else, switch
- Looping: while, do-while, for, for-each etc

Operator	Category	Precedence
Unary Operator	postfix	expression++ expression--
	prefix	++expression --expression +expression -expression ~!
Arithmetic Operator	multiplication	* / %
	addition	+ -
Shift Operator	shift	<< >> >>>
Relational Operator	comparison	< > <= >= instanceof
	equality	== !=
Bitwise Operator	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	
Logical Operator	logical AND	&&
	logical OR	
Ternary Operator	ternary	? :
Assignment Operator	assignment	= += -= *= /= % = &= ^= = <<= >>= >>>=

Data Types in Java



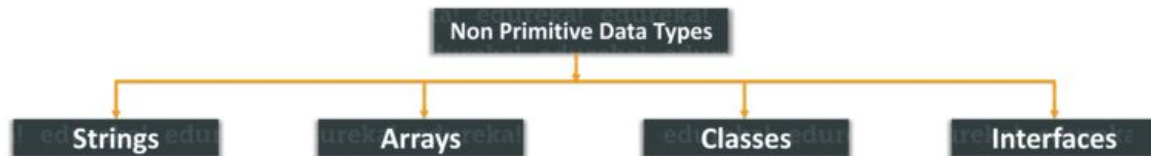
Primitive Data Types

- boolean
- byte
- char
- short
- int
- long
- float
- double

Type	Size in bytes	Range	Default Value
byte	1 byte	-128 to 127	0
short	2 bytes	-32,768 to 32,767	0
int	4 bytes	-2,147,483,648 to 2,147,483, 647	0
long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0
float	4 bytes	approximately $\pm 3.40282347\text{E}+38\text{F}$ (6-7 significant decimal digits) Java implements IEEE 754 standard	0.0f
double	8 bytes	approximately $\pm 1.79769313486231570\text{E}+308$ (15 significant decimal digits)	0.0d
char	2 bytes	0 to 65,536 (unsigned)	'\u0000'
boolean	Not precisely defined*	true or false	false

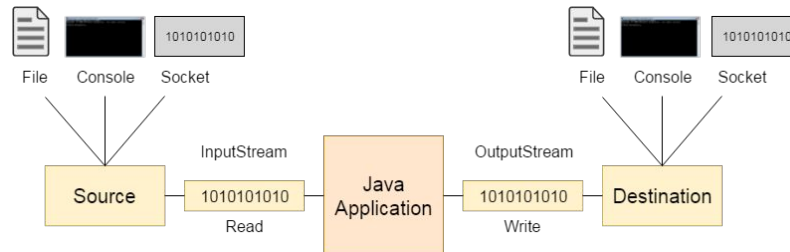
Non-Primitive Data Types

- **Strings:** String is a sequence of characters. But in Java, a string is an object that represents a sequence of characters. The `java.lang.String` class is used to create a string object.
- **Arrays:** Arrays in Java are homogeneous data structures implemented in Java as objects. Arrays store one or more values of a specific data type and provide indexed access to store the same. A specific element in an array is accessed by its index.
- **Classes:** A class in Java is a blueprint which includes all your data. A class contains fields(variables) and methods to describe the behavior of an object.
- **Interface:** Like a class, an interface can have methods and variables, but the methods declared in interface are by default abstract (only method signature, no body).



Java IO

- A stream is a sequence of data. In Java, a stream is composed of bytes.
- There are three default streams in Java:-
 - System.out: standard output stream. Eg: `System.out.println("simple message");`
 - System.in: standard input stream. `int i=System.in.read();`
 - System.err: standard error stream. Eg: `System.err.println("error message");`
- Scanner is a class in java.util package used for obtaining the input of the primitive types like int, double, etc. and strings. It is popularly used for reading from the input stream



Java IO - Example

```
1 import java.util.Scanner;
2
3 class Scan {
4     public static void main(String[] ar) {
5
6         Scanner sc = new Scanner(System.in);
7
8         // String input
9         System.out.println("Enter your Name:");
10        String name = sc.nextLine();
11
12        // Character input
13        System.out.println("Enter your Gender:");
14        char gender = sc.next().charAt(0); // it will take the first character
15
16        // Numerical data input
17        System.out.println("Enter your Age:");
18        int age = sc.nextInt();
19        System.out.print("Enter your Phone Number:+91 ");
20        long phoneNo = sc.nextLong();
21        System.out.println("Enter your CGPA:");
22        double CGPA = sc.nextDouble();
23
24        // Printing all the values
25        System.out.println("Name: " + name);
26        System.out.println("Gender: " + gender);
27        System.out.println("Age: " + age);
28        System.out.println("Mobile Number: +91 " + phoneNo);
29        System.out.println("CGPA: " + CGPA);
30    }
31 }
```

Operators in Java

Precedence	Operator	Operand type	Description
1	++, --	Arithmetic	Increment and decrement
1	+, -	Arithmetic	Unary plus and minus
1	~	Integral	Bitwise complement
1	!	Boolean	Logical complement
1	(type)	Any	Cast
2	*, /, %	Arithmetic	Multiplication, division, remainder
3	+, -	Arithmetic	Addition and subtraction
3	+	String	String concatenation
4	<<	Integral	Left shift
4	>>	Integral	Right shift with sign extension
4	>>>	Integral	Right shift with no extension
5	<, <=, >, >=	Arithmetic	Numeric comparison
5	instanceof	Object	Type comparison
6	==, !=	Primitive	Equality and inequality of value
6	==, !=	Object	Equality and inequality of reference
7	&	Integral	Bitwise AND
7	&	Boolean	Boolean AND
8	^	Integral	Bitwise XOR
8	^	Boolean	Boolean XOR
9		Integral	Bitwise OR
9		Boolean	Boolean OR
10	&&	Boolean	Conditional AND
11		Boolean	Conditional OR
12	?:	N/A	Conditional ternary operator
13	=	Any	Assignment

Associativity and Precedence of Operators

Operator	Precedence
Postfix	Expression++, expression--
Unary	++expression, --expression, +expression, -expression, !
Multiplication	* (multiply), / (divide), % (remainder)
Addition	+ (add), - (subtract)
Relational	<, >, <=, >=
Equality	==, !=
Logical AND	&&
Logical OR	
Assignment	=, +=, -=, *=, /=, %=

Decision Making in Java

The decision making statements in Java include the following keywords:-

- if Statement
 - if: An if statement consists of a boolean expression followed by one or more statements.
 - if-else: An if statement can be followed by an optional else statement, which executes when the boolean expression is false.
 - nested-if: You can use one if or else if statement inside another if or else if statement(s).
- switch statement
- ternary operator

Decision Making in Java

Condition is true

```
int number = 5;  
  
if (number > 0) {  
    // code  
}  
else {  
    // code  
}  
  
// code after if...else
```

Condition is false

```
int number = 5;  
  
if (number < 0) {  
    // code  
}  
else {  
    // code  
}  
  
// code after if...else
```

Ternary operator and Switch Statement

- Ternary operator in Java will be in the form of **condition?true-expression:false-expression**
- Switch statement is executes one statement from multiple conditions.
- The switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long.
- Since Java 7, you can use strings in the switch statement.

```
switch(expression) {  
    case x:  
        // code block  
        break;  
    case y:  
        // code block  
        break;  
    default:  
        // code block  
}
```

Algorithms

An algorithm is a step-by-step method for solving some problem.

Definition: “In mathematics and computer science, an algorithm is a finite sequence of well-defined, computer-implementable instructions, typically to solve a class of specific problems or to perform a computation.”

We need algorithms because of the following reasons:

- Scalability: It helps us to understand the scalability. When we have a big real-world problem, we need to scale it down into small-small steps to easily analyze the problem.
- Performance: The real-world is not easily broken down into smaller steps. If the problem can be easily broken into smaller steps means that the problem is feasible.

Characteristics of Algorithm

Algorithms generally have the following characteristics:

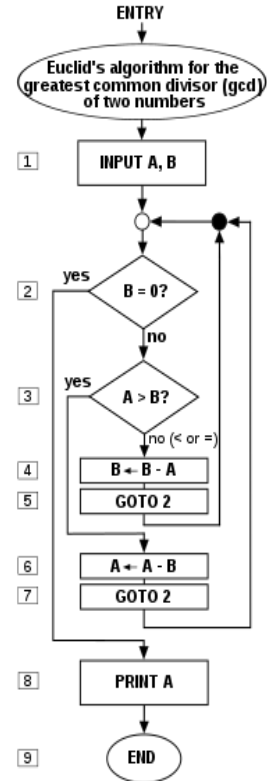
- **Clear and Unambiguous:** Algorithm should be clear and unambiguous. Each of its steps should be clear in all aspects and must lead to only one meaning.
- **Well-Defined Inputs:** If an algorithm says to take inputs, it should be well-defined inputs.
- **Well-Defined Outputs:** The algorithm must clearly define what output will be yielded and it should be well-defined as well.
- **Finiteness:** The algorithm must be finite, i.e. it should not end up in an infinite loops or similar.
- **Feasible:** The algorithm must be simple, generic and practical, such that it can be executed upon will the available resources.
- **Language Independent:** The Algorithm designed must be language-independent, i.e. it must be just plain instructions that can be implemented in any language, and yet the output will be same, as expected.

Introduction to Algorithms

Dataflow of an Algorithm

- **Problem:** A problem can be a real-world problem or any instance from the real-world problem for which we need to create a program or the set of instructions. The set of instructions is known as an algorithm.
- **Algorithm:** An algorithm will be designed for a problem which is a step by step procedure.
- **Input:** After designing an algorithm, the required and the desired inputs are provided to the algorithm.
- **Processing unit:** The input will be given to the processing unit, and the processing unit will produce the desired output.
- **Output:** The output is the outcome or the result of the program.

On the right side, the flow of Euclid's algorithm to calculate GCD of two numbers are given.



Issues while designing Algorithms


How to design algorithms:

As we know that an algorithm is a step-by-step procedure so we must follow some steps to design an algorithm.

Designing algorithm is not a one step process, we need different approaches for different algorithms

How to analyse algorithm efficiency

We cannot determine efficiency of an algorithm by benchmarking its speed. We need a sophisticated method to analyse an efficiency of an algorithm.



Issues while designing Algorithms


How to design algorithms:

As we know that an algorithm is a step-by-step procedure so we must follow some steps to design an algorithm.

Designing algorithm is not a one step process, we need different approaches for different algorithms

How to analyse algorithm efficiency

We cannot determine efficiency of an algorithm by benchmarking its speed. We need a sophisticated method to analyse an efficiency of an algorithm.



Types of approaches to solve while designing an algorithm:-

1. Brute force approach
2. Divide and Conquer
3. Greedy approach
4. Dynamic programming
5. Branch and Bound
- 6. Randomized Algorithm
7. Backtracking




Algorithm Complexity

We need to learn how to compare the performance different algorithms and choose the best one to solve a particular problem.

While analyzing an algorithm, we mostly consider time complexity and space complexity.

Time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input.

Space complexity of an algorithm quantifies the amount of space or memory taken by an algorithm to run as a function of the length of the input.



Time and Space Complexity



Explore | Expand | Enrich

Time and space complexity depends on lots of things like hardware, operating system, processors, etc.

However, we don't consider any of these factors while analyzing the algorithm. We will only consider the execution time of an algorithm.

Lets start with a simple example. Suppose you are given an array and an integer and you have to find if exists in array .

- Simple solution to this problem is traverse the whole array and check if the any element is equal to x

```
for i : 1 to length of A
    if A[i] is equal to x
        return TRUE
return FALSE
```

Time and Space Complexity



Explore | Expand | Enrich

```
for i : 1 to length of A
    if A[i] is equal to x
        return TRUE
return FALSE
```

Consider the above problem.

Each of the operation in computer take approximately constant time.

- Let each operation takes 'c' time. The number of lines of code executed is actually depends on the value of 'x'.

The worst case scenario: The if condition will run 'x' if the array isn't found

Best Case scenario: The if case will be run only once (element will be found in the first pass

The space requirement is constant. It is so because we store only an array and no additional space is required

Big O

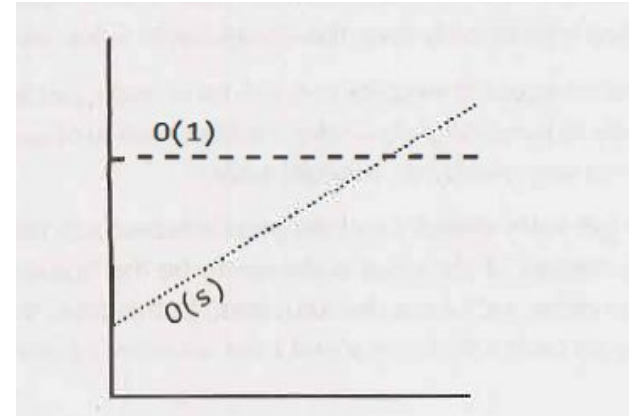
No matter how big the constant is and how slow the linear increase is, linear will at some point surpass constant.

There are many more runtimes than this. Some of the most common ones are $O(\log N)$, $O(N \log N)$, $O(N)$, $O(N^2)$ and $O(2N)$.

There's no fixed list of possible runtimes, though.

You can also have multiple variables in your runtime. For example, the time to paint a fence that's w meters wide and h meters high could be described as $O(wh)$.

If you needed p layers of paint, then you could say that the time is $O(whp)$.



Big O, Big Theta, and Big Omega



Explore | Expand | Enrich

big O, big theta, and big omega are used to describe runtimes.

O (big Oh): Big O describes an upper bound on the time. An algorithm that prints all the values in an array could be described as $O(N)$, but it could also be described as $O(N^2)$, $O(N^3)$, or $O(2N)$ (or many other big O times).

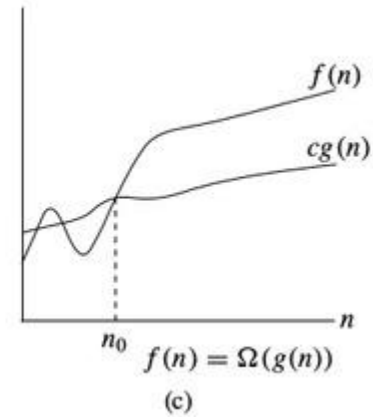
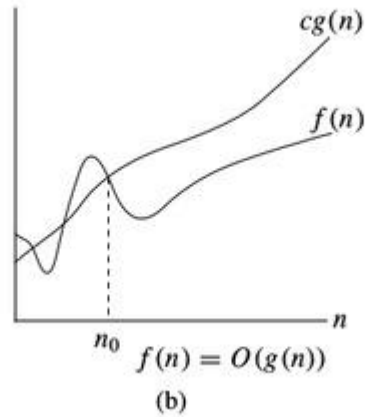
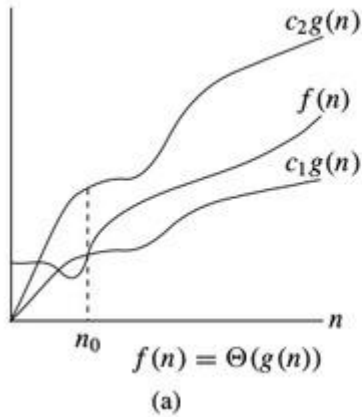
The algorithm is at least as fast as each of these; therefore they are upper bounds on the runtime. This is similar to a less-than-or-equal-to relationship.

- A simple algorithm to print the values in an array is $O(N)$

big omega: Omega is the equivalent concept but for lower bound. Printing the values in an array is $O(N)$ as well as $O(\log N)$ and $O(1)$. After all, you know that it won't be faster than those runtimes.

big theta: Theta means both O and Omega. That is, an algorithm is $\Theta(N)$ if it is both $O(N)$ and $\Omega(N)$

Big O, Big Theta, and Big Omega



Time Complexity



Explore | Expand | Enrich

The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input.

Note that the time to run is a function of the length of the input and not the actual execution time of the machine on which the algorithm is running on.

This is what the concept of asymptotic runtime, or big O time, means. We could describe the data transfer "algorithm" runtime as:

Electronic Transfer: $O(s)$, where s is the size of the file. This means that the time to transfer the file increases linearly with the size of the file. (Yes, this is a bit of a simplification, but that's okay for these purposes.)

Airplane Transfer: $O(1)$ with respect to the size of the file. As the size of the file increases, it won't take any longer to get the file to your friend. The time is constant.

Space Complexity

Time is not the only thing that matters in an algorithm. We might also care about the amount of memory or space-required by an algorithm.

Space complexity is a parallel concept to time complexity. If we need to create an array of size n , this will require $O(n)$ space.

If we need a two-dimensional array of size $n \times n$, this will require $O(n^2)$ space.

```
int sum(int n) { /* Ex 1.*/  
    if (n <= 0) {  
        return 0;  
    }  
    return n + sum(n-1);  
}
```

```
int pairSumSequence(int n) { /* Ex 2.*/  
    int sum = 0;  
    for (int i = 0; i < n; i++) {  
        sum += pairSum(i, i + 1);  
    }  
    return sum;  
}  
  
int pairSum(int a, int b) {  
    return a + b;  
}
```

Ex.1: Each call adds to the call stack 'n' times. Hence space complexity is $O(n)$. However, just because you have n calls total doesn't mean it takes $O(n)$ space. Consider Ex 2. There will be roughly $O(n)$ calls to pairSum. However, those calls do not exist simultaneously on the call stack, so you only need $O(1)$ space.

Conditions

It is very possible for $O(N)$ code to run faster than $O(1)$ code for specific inputs. Big O just describes the rate of increase.

For this reason, we drop the constants in runtime. An algorithm that one might have described as $O(2N)$ is actually $O(N)$.

For Eg:

```
for(i=0;i<N;i++){  
    print(x[1])  
    print(y[1])  
}
```

```
for(i=0;i<N;i++){  
    print(x[1])  
}  
for(i=0;i<N;i++){  
    print(y[1])  
}
```

Both can be considered the same. The first is $O(n)$ and second is $O(2n)$ which can be represented as $O(n)$

The second rule is that you can drop the terms which aren't significant. In this case:-

- $O(N^2 + N)$ becomes $O(N^2)$.
- $O(N + \log N)$ becomes $O(N)$.
- $O(5 \cdot 2^N + 1000N^{100})$ becomes $O(2^N)$.

Examples

The Time complexity of this code will be $O(N^2)$ as for $i=0$, it will run 0 times, $i=2$ it will run 2 times and hence $O(N^2)$

$$0 + 1 + 2 + \dots + (N - 1) = \frac{N*(N-1)}{2}$$

```
int count = 0;
for (int i = 0; i < N; i++)
    for (int j = 0; j < i; j++)
        count++;
```

An algorithm like binary search takes $O(\log N)$ as the time complexity as in every iteration array N is divided into $N/2$ times.

When $i=N$, it will do $N/2$ comparisons in the worst case.
When $i=N/2$, it will do $N/4$ comparisons in the worst case

Hence the time complexity of binary search is $O(\log N)$.

```
function binary_search(A, n, T) is
    L := 0
    R := n - 1
    while L ≤ R do
        m := floor((L + R) / 2)
        if A[m] < T then
            L := m + 1
        else if A[m] > T then
            R := m - 1
        else:
            return m
    return unsuccessful
```

Examples

```
int count = 0;
for (int i = N; i > 0; i /= 2)
    for (int j = 0; j < i; j++)
        count++;
```

This is a tricky case. In the first look, it seems like the complexity is $O(N * \log N)$. N for the j 's loop and $\log N$ for i 's loop. But its wrong. Lets see why.

Think about how many times `count++` will run.

When $i = N$, it will run N times.

When $i = N/2$, it will run $N/2$ times.

When $i = N/4$, it will run $N/4$ times and so on.

Total number of times `count++` will run is $N + N/2 + N/4 + \dots + 1 = 2 * N$. So the time complexity will be $O(N)$.

Sieve of Eratosthenes



Explore | Expand | Enrich

Introduction

The sieve of Eratosthenes is an ancient algorithm for finding all prime numbers up to any given limit.

It does so by iteratively marking as composite (i.e., not prime) the multiples of each prime, starting with the first prime number, 2.

The multiples of a given prime are generated as a sequence of numbers starting from that prime, with constant difference between them that is equal to that prime

- This is the sieve's key distinction from using trial division to sequentially test each candidate number for divisibility by each prime.

Once all the multiples of each discovered prime have been marked as composites, the remaining unmarked numbers are primes.

Sieve of Eratosthenes

The Sieve in Action

The steps of the algorithm to find all primes below 121

	2	3	4	5	6	7	8	9	10	Prime numbers
11	12	13	14	15	16	17	18	19	20	
21	22	23	24	25	26	27	28	29	30	
31	32	33	34	35	36	37	38	39	40	
41	42	43	44	45	46	47	48	49	50	
51	52	53	54	55	56	57	58	59	60	
61	62	63	64	65	66	67	68	69	70	
71	72	73	74	75	76	77	78	79	80	
81	82	83	84	85	86	87	88	89	90	
91	92	93	94	95	96	97	98	99	100	
101	102	103	104	105	106	107	108	109	110	
111	112	113	114	115	116	117	118	119	120	

Sieve of Eratosthenes

Algorithm

To find all the prime numbers less than or equal to 30, proceed as follows.

First, generate a list of integers from 2 to 30

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

The first number in the list is 2; cross out every 2nd number in the list after 2 by counting up from 2 in increments of 2 (these will be all the multiples of 2 in the list):

2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~ 17 ~~18~~ 19 ~~20~~ 21 ~~22~~ 23 ~~24~~ 25 ~~26~~ 27 ~~28~~ 29 ~~30~~

The next number in the list after 2 is 3; cross out every 3rd number in the list after 3 by counting up from 3 in increments of 3 (these will be all the multiples of 3 in the list)

2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~ 17 ~~18~~ 19 ~~20~~ 21 ~~22~~ 23 ~~24~~ 25 ~~26~~ 27 ~~28~~ 29 ~~30~~

The next number not yet crossed out in the list after 3 is 5; cross out every 5th number in the list after 5 by counting up from 5 in increments of 5 (i.e. all the multiples of 5):

2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~ 17 ~~18~~ 19 ~~20~~ 21 ~~22~~ 23 ~~24~~ 25 ~~26~~ 27 ~~28~~ 29 ~~30~~

The next number not yet crossed out in the list after 5 is 7; the next step would be to cross out every 7th number in the list after 7, but they are all already crossed out at this point, as these numbers (14, 21, 28) are also multiples of smaller primes because 7×7 is greater than 30. The numbers not crossed out at this point in the list are all the prime numbers below 30:

2 3 5 7 11 13 17 19 23 29

Sieve of Eratosthenes



Explore | Expand | Enrich

Algorithm

algorithm Sieve of Eratosthenes is

input: an integer $n > 1$.

output: all prime numbers from 2 through n .

let A be an array of Boolean values, indexed by integers 2 to n ,
initially all set to true.

for $i = 2, 3, 4, \dots$, not exceeding \sqrt{n} do

- if $A[i]$ is true

- for $j = i^2, i^2+i, i^2+2i, i^2+3i, \dots$, not exceeding n do

- $A[j] := \text{false}$

return all i such that $A[i]$ is true.

Time Complexity: $O(n \log(\log n))$

Sieve of Eratosthenes



Explore | Expand | Enrich

Program: Sieve of Eratosthanes in Java

Sample Input/Output

Input : n =10

Output : 2 3 5 7

Input : n = 20

Output: 2 3 5 7 11 13 17 19



Sieve of Eratosthenes



Explore | Expand | Enrich

Other types of Sieves

The simple sieve algorithm produces all primes not greater than n .

It includes a common optimization, which is to start enumerating the multiples of each prime i from i^2 .

The time complexity of this algorithm is $O(n \log \log n)$, provided the array update is an $O(1)$ operation, as is usually the case.

- The problem with the sieve of Eratosthenes is not the number of operations it performs but rather its memory requirements.

For large n , the range of primes may not fit in memory; worse, even for moderate n , its cache use is highly suboptimal.

The simple sieve algorithm walks through the entire array A , exhibiting almost no locality of reference

Locality of Reference: In computer science, locality of reference, also known as the principle of locality, is the tendency of a processor to access the same set of memory locations repetitively over a short period of time

Segmented Sieves



Explore | Expand | Enrich

Introduction

A solution to these problems is offered by segmented sieves, where only portions of the range are sieved at a time.

The idea of a segmented sieve is to divide the range $[0..n-1]$ in different segments and compute primes in all segments one by one.

This algorithm first uses Simple Sieve to find primes smaller than or equal to \sqrt{n} .

- In Simple Sieve, we needed $O(n)$ space which may not be feasible for large n .

Here we need $O(\sqrt{n})$ space and we process smaller ranges at a time (locality of reference)

Algorithm: Segmented Sieves

- Use Simple Sieve to find all primes up to the square root of 'n' and store these primes in an array "prime[]". Store the found primes in an array 'prime[]'.
- We need all primes in the range [0..n-1]. We divide this range into different segments such that the size of every segment is at-most \sqrt{n}
- Do following for every segment [low..high]
 - Create an array mark[high-low+1]. Here we need only $O(x)$ space where x is a number of elements in a given range.
 - Iterate through all primes found in step 1. For every prime, mark its multiples in the given range [low..high].

Segmented Sieves



Explore | Expand | Enrich

Program: Segmented Sieves

Sample I/O

Input

n=100

Output

2 3 5 7 11 13 17 19 23 29 31 37 41
43 47 53 59 61 67 71 73 79 83 89 97

Note

The time complexity (or a number of operations) by Segmented Sieve is the same as Simple Sieve.

It has advantages for large 'n' as it has better locality of reference thus allowing better caching by the CPU and also requires less memory space.

Introduction

The standard Sieve of Eratosthenes requires that you specify an upper bound before you start

But often, programs that use prime numbers don't know the upper bound in advance, or don't want to pre-compute and store all of the primes up to their bound.

In such cases, Incremental Sieve algorithm can be used

An incremental formulation of the sieve generates primes indefinitely (i.e. without an upper bound) by interleaving the generation of primes with the generation of their multiples (so that primes can be found in gaps between the multiples), where the multiples of each prime p are generated directly, by counting up from the square of the prime in increments of p (or $2p$ for odd primes)

The generation must be initiated only when the prime's square is reached, to avoid adverse effects on efficiency. It can be expressed symbolically under the dataflow paradigm as

$\text{primes} = [2, 3, \dots] \setminus [[p^2, p^2+p, \dots] \text{ for } p \text{ in primes}]$

Where \setminus denotes the set subtraction of arithmetic progressions of numbers.

Euler's Phi Algorithm



Explore | Expand | Enrich

Introduction

Euler's totient function, also known as phi-function $\phi(n)$, counts the number of integers between 1 and n inclusive, which are coprime to n

Two numbers are coprime if their greatest common divisor equals 1 (1 is considered to be coprime to any number).

Here are values of $\phi(n)$ for the first few positive integers:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
$\phi(n)$	1	1	2	2	4	2	6	4	6	4	10	4	12	6	8	8	16	6	18	8	12

Examples:

$$\Phi(4) = 2$$

$\gcd(1, 4)$ is 1 and $\gcd(3, 4)$ is 1

$$\Phi(5) = 4$$

$\gcd(1, 5)$ is 1, $\gcd(2, 5)$ is 1,

$\gcd(3, 5)$ is 1 and $\gcd(4, 5)$ is 1

$$\Phi(6) = 2$$

$\gcd(1, 6)$ is 1 and $\gcd(5, 6)$ is 1,

Euler's Phi Algorithm



Explore | Expand | Enrich

Basic Algorithm

A simple solution is to iterate through all numbers from 1 to $n-1$ and count numbers with gcd with n

EulerPhi1.java

Time Complexity: $O(N \log N)$



Better Solution

The idea is based on Euler's product formula which states that the value of totient functions is below the product overall prime factors p of n .

$$\phi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right)$$

- The formula basically says that the value of $\Phi(n)$ is equal to n multiplied by-product of $(1 - 1/p)$ for all prime factors p of n . For example value of $\Phi(6) = 6 * (1-1/2) * (1 - 1/3) = 2$.

Euler's Phi Algorithm



Explore | Expand | Enrich

Better Solution - 1

Program: EulerPhi2.java

Algorithm

- 1) Initialize : result = n
- 2) Run a loop from 'p' = 2 to \sqrt{n} , do following for every 'p'.
 - a) If p divides n, then
 - Set: $\text{result} = \text{result} * (1.0 - (1.0 / (\text{float}) p));$
 - Divide all occurrences of p in n.
- 3) Return result

Euler's Phi Algorithm



Explore | Expand | Enrich

Better Solution - 2

Program: EulerPhi3.java

We can avoid floating-point calculations in the above method. The idea is to count all prime factors and their multiples and subtract this count from n to get the totient function value (Prime factors and multiples of prime factors won't have GCD as 1)

Algorithm

- 1) Initialize result as n
- 2) Consider every number 'p' (where 'p' varies from 2 to Φn).
 - If p divides n , then do following
 - a) Subtract all multiples of p from 1 to n [all multiples of p will have gcd more than 1 (at least p) with n]
 - b) Update n by repeatedly dividing it by p.
- 3) If the reduced n is more than 1, then remove all multiples of n from result.

Introduction

Strobogrammatic Number is a number whose numeral is rotationally symmetric so that it appears the same when rotated 180 degrees.

In other words, Strobogrammatic Number appears the same right-side up and upside down.

When written using standard characters (ASCII), the numbers, 0, 1, 8 are symmetrical around the horizontal axis, and 6 and 9 are the same as each other when rotated 180 degrees.

- In such a system, the first few strobogrammatic numbers are:

0, 1, 8, 11, 69, 88, 96, 101, 111, 181, 609, 619, 689, 808, 818, 888.....

The years 1881 and 1961 were the most recent strobogrammatic years; the next strobogrammatic year will be 6009.

Examples

0 after 180° rotation : $(0 \rightarrow 0)$

1 after 180° rotation : $(1 \rightarrow 1)$

8 after 180° rotation : $(8 \rightarrow 8)$

• 6 after 180° rotation : $(6 \rightarrow 9)$

9 after 180° rotation : $(9 \rightarrow 6)$

Strobogrammatic Number



Explore | Expand | Enrich

Program

StrobogrammaticNumber.java

For the given length n , find all n -length Strobogrammatic numbers.



Strobogrammatic Number



Explore | Expand | Enrich

Program

StrobogrammaticNumber2.java

Given number n , check if it is a Strobogrammatic Number.

Sample IO

Input: "69"

- Output: true

Input: "88"

Output: true

Input: "962"

Output: false

Introduction

Also known as the Chinese Remainder Theorem

The Chinese remainder theorem states that if one knows the remainders of the Euclidean division of an integer x by several integers, then one can determine uniquely the remainder of the division of x by the product of these integers, under the condition that the divisors are pairwise coprime.

We are given two arrays $\text{num}[0..k-1]$ and $\text{rem}[0..k-1]$.

- In $\text{num}[0..k-1]$, every pair is coprime (gcd for every pair is 1). We need to find minimum positive number x such that:

$$x \% \text{num}[0] = \text{rem}[0],$$

$$x \% \text{num}[1] = \text{rem}[1],$$

.....

.....

.....

$$x \% \text{num}[k-1] = \text{rem}[k-1]$$

Note that the integers in the num array would be pairwise coprime.

Remainder Theorem

Explanation

Basically, we are given k numbers which are pairwise coprime, and given remainders of these numbers when an unknown number x is divided by them.

We need to find the minimum possible value of x that produces given remainders.

Chinese Remainder Theorem states that there always exists an x that satisfies given congruences

Let $\text{nums} = [n_1, n_2, n_3 \dots n_k]$ and $\text{rems} = [a_1, a_2, a_3 \dots a_k]$. Then, for any given sequence of integers $\text{rem}[0]$, $\text{rem}[1]$, ... $\text{rem}[k-1]$, there exists an integer x solving the following system of simultaneous congruences.

THEOREM

Chinese Remainder Theorem

Given pairwise coprime positive integers n_1, n_2, \dots, n_k and arbitrary integers a_1, a_2, \dots, a_k , the system of simultaneous congruences

$$x \equiv a_1 \pmod{n_1}$$

$$x \equiv a_2 \pmod{n_2}$$

$$\vdots$$

$$x \equiv a_k \pmod{n_k}$$

has a solution, and the solution is unique modulo $N = n_1 n_2 \dots n_k$.

Remainder Theorem



Explore | Expand | Enrich

Program

ChineseRemainderThm.java

Sample IO

Input

num= [5, 7]

rem = [1, 3]

Output

31

Explanation

31 is the smallest number such that:

- (1) When we divide it by 5, we get remainder 1.
- (2) When we divide it by 7, we get remainder 3.

Input

num= [3, 4, 5]

rem = [2, 3, 1]

Output

11

Explanation

11 is the smallest number such that:

- (1) When we divide it by 3, we get remainder 2.
- (2) When we divide it by 4, we get remainder 3.
- (3) When we divide it by 5, we get remainder 1.

Introduction

You are given an integer 'X', you need to convert the integer to binary format and check if the binary format is palindrome or not

For Example, 5 i.e. 101, 27 i.e. 11011 are numbers whose binary representations are palindromes. Whereas 10 i.e. 1011 and 20 i.e. 10100 are not palindromes

- The problem is very similar to checking whether a string is palindrome or not. Hence the binary representation must be strings.

We start from leftmost and rightmost bits and compare bits one by one. If we find a mismatch, then return false.

We can use the regular palindrome program. Converting integer 'val' to binary is as easy as `Integer.toBinaryString(val)`

Binary Palindrome



Explore | Expand | Enrich

Program

Check whether a string is palindrome or not

palindrome1.java

Sample IO

Input: madam

Output: The string is palindrome

Input: deed

Output: The string is palindrome

Input: algorithm

Output: The string is not palindrome



Binary Palindrome




Explore | Expand | Enrich

Find the nth number whose binary representation is a palindrome

Find the nth number whose binary representation is a palindrome.

A special caution that you should not consider the leading zeros, while considering the binary representation.

Approach: Traverse through all the integers from 1 to $2^{31} - 1$ and increment palindrome count, if the number is a palindrome. When the palindrome count reaches the required n, break the loop and return the current integer.



Programs

palindrome2.java

Sample IO

Input : 1

Output : 1

1st Number whose binary representation
is palindrome is 1 (1)

Input : 9

Output : 27

9th Number whose binary representation
is palindrome is 27 (11011)

Introduction

The booth algorithm is a multiplication algorithm that allows us to multiply the two signed binary integers in 2's complement, respectively.

It is also used to speed up the performance of the multiplication process. It is very efficient too.

- It works on the string bits 0's in the multiplier that requires no additional bit only shift the right-most string bits and a string of 1's in a multiplier bit weight 2^k to weight 2^m that can be considered as $2^{k+1} - 2^m$.

Flowchart

As in all multiplication schemes, booth algorithm requires examination of the multiplier bits and shifting of the partial product.

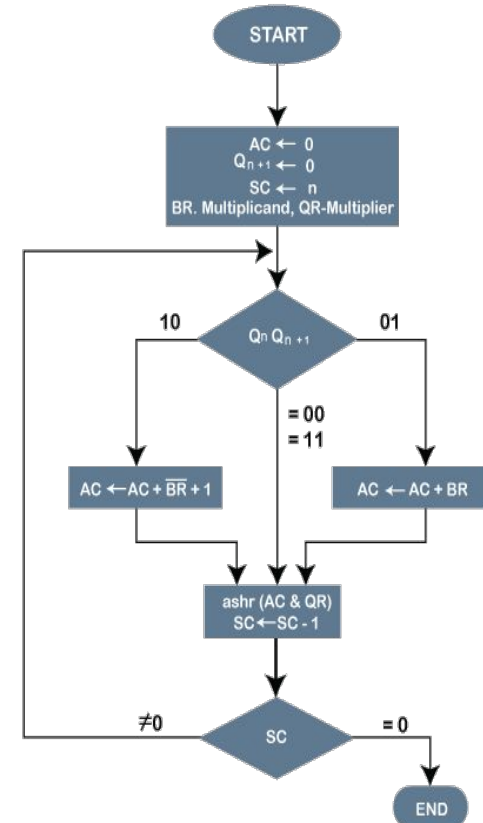
Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial product, or left unchanged according to following rules:

1. The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier
2. The multiplicand is added to the partial product upon encountering the first 0 (provided that there was a previous '1') in a string of 0's in the multiplier.
3. The partial product does not change when the multiplier bit is identical to the previous multiplier bit.

Booth's Algorithm

Flowchart

1. Initially, AC and $Q_n + 1$ bits are set to 0
2. The SC is a sequence counter that represents the total bits set n , which is equal to the number of bits in the multiplier.
3. There are BR that represent the multiplicand bits, and QR represents the multiplier bits.
4. After that, we encountered two bits of the multiplier as Q_n and $Q_n + 1$, where Q_n represents the last bit of QR, and $Q_n + 1$ represents the incremented bit of Q_n by 1.
5. Suppose two bits of the multiplier is equal to 10; it means that we have to subtract the multiplier from the partial product in the accumulator AC and then perform the arithmetic shift operation (ashr).
6. If the two of the multipliers equal to 01, it means we need to perform the addition of the multiplicand to the partial product in accumulator AC and then perform the arithmetic shift operation (ashr), including $Q_n + 1$.
7. The arithmetic shift operation is used in Booth's algorithm to shift AC and QR bits to the right by one and remains the sign bit in AC unchanged.
8. And the sequence counter is continuously decremented till the computational loop is repeated, equal to the number of bits (n).



Algorithm

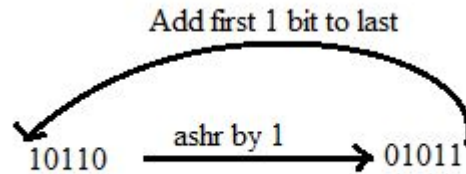
1. Set the Multiplicand and Multiplier binary bits as M and Q, respectively.
2. Initially, we set the AC and $Q_n + 1$ registers value to 0.
3. SC represents the number of Multiplier bits (Q), and it is a sequence counter that is continuously decremented till equal to the number of bits (n) or reached to 0.
4. A Q_n represents the last bit of the Q, and the Q_{n+1} shows the incremented bit of Q_n by 1.
5. On each cycle of the booth algorithm, Q_n and $Q_n + 1$ bits will be checked on the following parameters as follows:
 1. When two bits Q_n and $Q_n + 1$ are 00 or 11, we simply perform the arithmetic shift right operation (ashr) to the partial product AC. And the bits of Q_n and $Q_n + 1$ is incremented by 1 bit.
 2. If the bits of Q_n and $Q_n + 1$ is shows to 01, the multiplicand bits (M) will be added to the AC (Accumulator register). After that, we perform the right shift operation to the AC and QR bits by 1.
 3. If the bits of Q_n and $Q_n + 1$ is shows to 10, the multiplicand bits (M) will be subtracted from the AC (Accumulator register). After that, we perform the right shift operation to the AC and QR bits by 1.
6. The operation continuously works till we reached $n - 1$ bit in the booth algorithm.
7. Results of the Multiplication binary bits will be stored in the AC and QR registers.

Booth's Algorithm

Methods used in Booth's algorithm

RSC (Right Shift Circular)

It shifts the right-most bit of the binary number, and then it is added to the beginning of the binary bits.



RSA (Right Shift Arithmetic)

It adds the two binary bits and then shift the result to the right by 1-bit position.

Example: $0100 + 0110 \Rightarrow 1010$, after adding the binary number shift each bit by 1 to the right and put the first bit of resultant to the beginning of the new bit.

Booth's Algorithm

Example

A	Q	Q ₋₁	M	Initial values	
0000	0011	0	0111		
1001	0011	0	0111	A ← A - M Shift	First cycle
1100	1001	1	0111		
1110	0100	1	0111	Shift	Second cycle
0101	0100	1	0111	A ← A + M Shift	
0010	1010	0	0111		
0001	0101	0	0111	Shift	Fourth cycle

Booth's Algorithm



Explore | Expand | Enrich

Program

Booth1.java

Sample IO

Input

Enter two integer numbers

7 -7

Output

A : 0111 0000 0

S : 1001 0000 0

P : 0000 1001 0

P : 1100 1100 1

P : 0001 1110 0

P : 0000 1111 0

P : 1100 1111 1

Result : $7 * -7 = -49$

Introduction

Euclidean algorithm or Euclid's algorithm, is an efficient method for computing the greatest common divisor (GCD) of two integers (numbers), the largest number that divides them both without a remainder.

First the two numbers are subjected to prime factorizations, and the common factors of the two prime factorizations are multiplied to get the GCD

$$36 = 2 \times 2 \times 3 \times 3$$

$$60 = 2 \times 2 \times 3 \times 5$$

$$\begin{aligned}\text{GCD} &= \text{Multiplication of common factors} \\ &= 2 \times 2 \times 3 \\ &= 12\end{aligned}$$

Idea

The algorithm is based on the below facts.

If we subtract a smaller number from a larger (we reduce a larger number), GCD doesn't change. So if we keep subtracting repeatedly the larger of two, we end up with GCD.

- Now instead of subtraction, if we divide the smaller number, the algorithm stops when we find remainder 0.



Binary Palindrome



Explore | Expand | Enrich

Program

euclid1.java

Sample IO

Input : 10 15

Output : 5

Input : 35 10

Output : 5

Input : 31 2

Output : 1

Idea

Extended Euclidean algorithm also finds integer coefficients x and y such that:

$$ax + by = \gcd(a, b)$$

- The extended Euclidean algorithm updates results of $\gcd(a, b)$ using the results calculated by recursive call $\gcd(b\%a, a)$.

Let values of x and y calculated by the recursive call be x_1 and y_1 .

x and y are updated using the below expressions.

$$\begin{aligned}x &= y_1 - \lfloor b/a \rfloor * x_1 \\ y &= x_1\end{aligned}$$

Program

euclid2.java

Sample IO

Same as above, but you can also print the integer coefficients

The extended Euclidean algorithm is particularly useful when a and b are coprime.

- With that provision, x is the modular multiplicative inverse of a modulo b , and y is the modular multiplicative inverse of b modulo a .

Similarly, the polynomial extended Euclidean algorithm allows one to compute the multiplicative inverse in algebraic field extensions and, in particular in finite fields of non prime order.

It follows that both extended Euclidean algorithms are widely used in cryptography.

In particular, the computation of the modular multiplicative inverse is an essential step in the derivation of key-pairs in the RSA public-key encryption method.

Introduction

The Karatsuba algorithm is a fast multiplication algorithm. It was discovered by Anatoly Karatsuba in 1960 and published in 1962.

It is a fast multiplication algorithm that uses a divide and conquer approach to multiply two numbers.

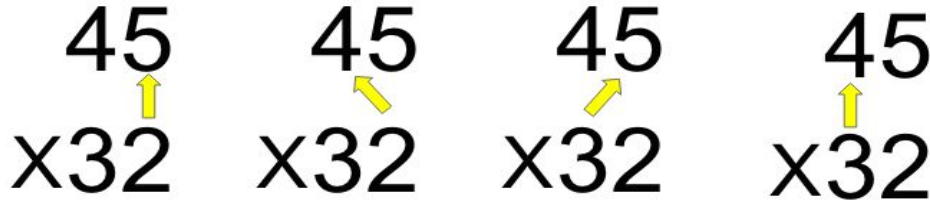
The naive algorithm for multiplying 2 numbers has a running time of $O(n^2)$, whereas the Karatsuba algorithm has a runtime of

$$\Theta(n^{\log_2 3}) \approx \Theta(n^{1.585})$$

Being able to multiply numbers quickly is very important. Computer scientists often consider multiplication to be a constant time $\sim O(1)$ operation which is reasonable for small numbers.

Whereas for larger numbers, the actual running times need to be factored in, which is $O(n^2)$

Naive Algorithm

The image shows four identical multiplication problems arranged horizontally: 45 x 32. Each problem has a yellow arrow pointing from the '5' in the top number to the '2' in the bottom number, representing the four sub-problems in the naive algorithm.
$$\begin{array}{r} 45 \\ \times 32 \\ \hline \end{array}$$

The key idea is to reduce the four sub-problems in multiplication to three unique problems.

Thus, on calculating the three unique sub-problems, the original four sub-problems are solved using addition or subtraction operation.

Karatsuba Algorithm



Explore | Expand | Enrich

Explanation

Karatsuba stated that if we have to multiply two n-digit numbers x and y

The two numbers x and y has to be decomposed into 4 numbers x1, x2 and y1,y2

$$x = x1 * B^m + x2$$

$$y = y1 * B^m + y2$$

Eg: 45 * 32

45 can be represented as $4*10 + 5$

Where $x1 = 4$, $x2=5$ with $m=1$

Note: B is the base (base-10 in this case)

Explanation

The product of x and y after the decomposition can be represented by:-

$$xy = (x1 * B^m + x2)(y1 * B^m + y2)$$

$$\Rightarrow xy = x1 * y1 * B^{(2m)} + x1 * y2 * B^m + x2 * y1 * B^m + x2 * y2$$

Observe that there are 4 sub-problems:

- $X1 * Y1$
- $X1 * Y2$
- $X2 * Y1$
- $X2 * Y2$

We are going to reduce the 4 sub-problems to 3 sub-problems

Procedure

Consider the following

- $a = X1 * Y1$
- $b = X1 * Y2 + X2 * Y1$
- $c = X2 * Y2$

Then the following equation

$$xy = x1 * y1 * B^{(2m)} + x1 * y2 * B^m + x2 * y1 * B^m + x2 * y2$$

Becomes $xy = a * B^{(2m)} + b * B^m + c$

Karatsuba came up with a brilliant idea to calculate b with the following formula

$$b = (x1 + x2)(y1 + y2) - a - c$$

Example

Consider the following multiplication: 47×78

$$x = 47$$

$$x = 4 * 10 + 7$$

$$x_1 = 4$$

$$x_2 = 7$$

$$y = 78$$

$$y = 7 * 10 + 8$$

$$y_1 = 7$$

$$y_2 = 8$$

The Three subproblems:

$$a = x_1 * y_1 = 4 * 7 = 28$$

$$c = x_2 * y_2 = 7 * 8 = 56$$

$$b = (x_1 + x_2)(y_1 + y_2) - a - c = 11 * 15 - 28 - 56$$

Note: $11 * 15$ can in turn be multiplied using Karatsuba Algorithm

Time Complexity

Assuming that we replace two of the multiplications with only one makes the program faster.

Karatsuba improves the multiplication process by replacing the initial complexity from quadratic to $\Theta(n^{\log_2 3}) \approx \Theta(n^{1.585})$

Where n is the number of digits of the numbers multiplying.

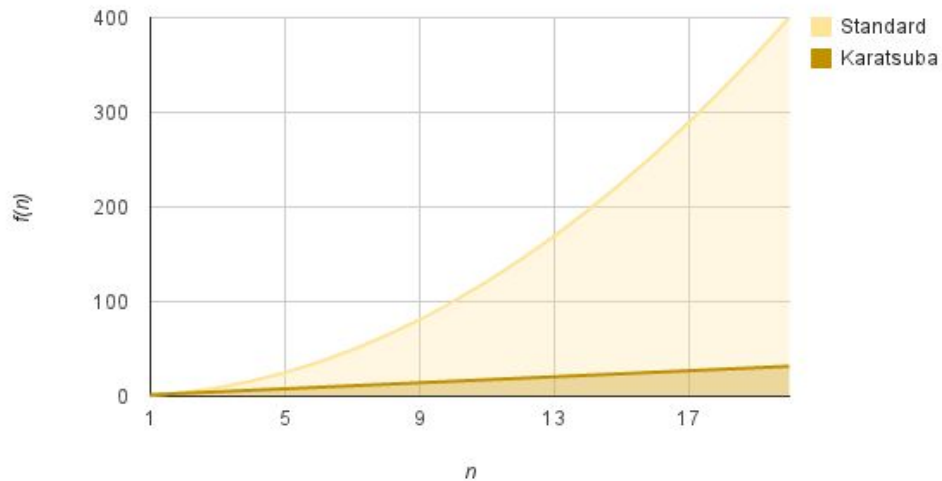
- The Time Complexity of the algorithm can be represented as follows

$$T(n) = 3 * T(n/2) + O(n)$$

Karatsuba Algorithm

Time Complexity

$$T(n) = 3 * T(n/2) + O(n)$$



Implementation

We shall look at implementing Karatsuba algorithm for four digits.

Algorithm

1. Compute starting set ($a*c$)
2. Compute set after starting set may it be ending set ($b*d$)
3. Compute starting set with ending sets
4. Subtract values of Step 3 from Step 2 from Step 1
5. Add all the values with the following modifications:-
 1. Pad up 4 zeros to the number obtained from Step 1
 2. Step 2 value unchanged
 3. Pad up two zeros to value obtained from Step 4.

Karatsuba Algorithm



Explore | Expand | Enrich

Program

karatsuba1.java

Sample IO

Karatsuba Multiplication Algorithm Test

Enter two integer numbers:
24061994 28563

Product : 687282734622

Introduction

In this program you are assumed to have a sequence of bits 0s and 1s

The sequence can be in the form of a binary string ("110011") or as a binary array([1,0,1,1])

- Given that you can flip any one bits in the sequence, you have to do it such that you get the longest sequence of 1s.



Longest Sequence of 1s after flip



Explore | Expand | Enrich

Example

Consider the sequence 11011101111

To get the longest sequence of bits, out of all the possible flips, flipping the highlighted digit in the sequence 110111**0**1111.

- This will yield the following sequence: 110111**1**1111

The output of the program would be 8 since there are 8 consecutive bits in the sequence

Additionally you may be given with a decimal which you would have to convert to a bit string to apply the algorithm too

Longest Sequence of 1s after flip



Explore | Expand | Enrich

Solution

Given a number, provide the longest sequence of 1s after a flip of its bit sequence.

A simple solution is to store the binary representation of a given number in a binary array.

An efficient solution is to walk through the bits in the binary representation of the given number.

We keep track of the current 1's sequence length and the previous 1's sequence length. When we see a zero, update the previous Length:

- If the next bit is a 1, the previous Length should be set to the current Length.
- If the next bit is a 0, then we can't merge these sequences together. So, set the previous Length to 0.

We update max length by comparing the following two:

- The current value of max-length
- Current-Length + Previous-Length .

Result = return max-length+1 (// add 1 for flip bit count)

Longest Sequence of 1s after flip



Explore | Expand | Enrich

Program

bitflip1.java

Sample IO

Enter a number: 13

Answer is: 4

- Enter a number: 1775

Answer is: 8

Enter a number: 15

Answer is: 5

Time Complexity: $O(n)$

Where n is the number of bits

Longest Sequence of 1s after flip



Explore | Expand | Enrich

Program

bitflip2.java, bitflip3.java

Given a binary array `nums` and an integer `k`, return the maximum number of consecutive 1's in the array if you can flip at most `k` 0's.

Sample IO

Input: `nums = [1,1,1,0,0,0,1,1,1,1,0]`, `k = 2`

Output: 6

Explanation: `[1,1,1,0,0,1,1,1,1,1,1]`

Bolded numbers were flipped from 0 to 1. The longest subarray is underlined.

Input: `nums = [0,0,1,1,0,0,1,1,1,0,1,1,0,0,0,1,1,1,1]`, `k = 3`

Output: 10

Explanation: `[0,0,1,1,1,1,1,1,1,1,1,0,0,0,1,1,1,1]`

Bolded numbers were flipped from 0 to 1. The longest subarray is underlined.

Time Complexity: $O(n)$

Where `n` is the number of bits

Swap two nibbles in a byte



Explore | Expand | Enrich

Introduction

A nibble consists of four bits. It is a four-bit aggregation, or half an octet.

There are two nibbles in a byte.

For example, 64 is to be represented as 01000000 in a byte (or 8 bits). The two nibbles are (0100) and (0000).

• Nibble Swap

Suppose you have a number say 100.

100 is to be represented as 01100100 in a byte (or 8 bits)

The two nibbles are 0110 and 0100.

After swapping the nibbles, we get 01000110 which is 70 in decimal.

To swap the nibbles, We will use bitwise operators `&`, `|`, `<<` and `>>` to swap the nibbles in a byte.

Swap two nibbles in a byte



Explore | Expand | Enrich

Procedure

The snippet of code for nibble swap is as follows:

```
(x & 0x0F) << 4 | (x & 0xF0) >> 4
```

Explanation

As we know binary of 100 is 01100100. To swap the nibble we split the operation in two parts, in first part we get last 4 bits and in second part we get first 4 bit of a byte.

First operation: The expression “data & 0x0F” gives us the last 4 bits of data and result would be 00000100.

Using bitwise left shift operator ‘<<’, we shift the last four bits to the left 4 times and make the new last four bits as 0. The result after the shift is 01000000.

Second operation: The expression “data & 0xF0” gives us first four bits of data and result would be 01100000.

Using bitwise right shift operator ‘>>’, we shift the digit to the right 4 times and make the first four bits as 0. The result after the shift is 00000110.

After completing the two operation we use the bitwise OR ‘|’ operation on them. After OR operation you will find that first nibble to the place of last nibble and last nibble to the place of first nibble

Swap two nibbles in a byte



Explore | Expand | Enrich

Program

nibble1.java

Sample IO

Input: 100

- Output: 70

Input: 200

Output: 140

Input: 67

Output: 52

Introduction

The block swap algorithm for array rotation is an efficient algorithm that is used for array rotation.

It can do the work in $O(n)$ time complexity

- We would be given an array `arr` of size `n`, and an integer `d`

One would have to rotate the array `arr` by `d` elements

We have both recursive and iterative versions of the algorithms



Block Swap Algorithm

Demo

Write an algorithm to rotate the following array by 2

1	2	3	4	5	6	7
---	---	---	---	---	---	---

3	4	5	6	7	1	2
---	---	---	---	---	---	---

Algorithm

Initialize $A = \text{arr}[0..d-1]$ and $B = \text{arr}[d..n-1]$

- 1) Do following until size of A is equal to size of B
 - a) If A is shorter, divide B into B_l and B_r such that B_r is of same length as A. Swap A and B_r to change AB_lB_r into B_rB_lA . Now A is at its final place, so recur on pieces of B.
 - b) If A is longer, divide A into A_l and A_r such that A_l is of same length as B. Swap A_l and B to change A_lA_rB into BA_rA_l . Now B is at its final place, so recur on pieces of A.
- 2) Finally when A and B are of equal size, block swap them.

Block Swap Algorithm



Explore | Expand | Enrich

Programs

blockswap1.java

blockswap2.java



Subarray vs Subsequence

Subsequence: A subsequence is a sequence that can be derived from another sequence by zero or more elements, without changing the order of the remaining elements.

Subarray: A subarray is a contiguous part of array. An array that is inside another array.

A Subsequence cannot be a subarray

For instance, let $arr = [1, 2, 3, 4, 5]$

Examples for Subarrays: $[1]$, $[1, 2]$, $[2, 3, 4]$ etc

Examples for Subsequences: $[1]$, $[1, 3]$, $[1, 2, 3]$, $[2, 3, 4]$


Introduction

You would be given an array of integers and you have to find a subarray which has the largest product

The array can contain both positive and negative integers

- As discussed earlier -- A subarray is a contiguous subsequence of the array.

The solution must preferably have $O(n)$ time complexity and constant space complexity



Examples

Example-1

Input: nums = [2,3,-2,4]

Output: 6

Explanation: [2,3] has the largest product 6.

Example-2

Input: nums = [-2,0,-1]

Output: 0

Explanation: The result cannot be 2, because [-2,-1] is not a subarray.

Example-3

Input: nums = [6, -3, -10, 0, 2]

Output: 180

Explanation: The subarray is [6, -3, -10] with largest product as 180

Maximum Product Subarray



Explore | Expand | Enrich

Program

mps1.java

This solution is a naive solution. The idea is to traverse over every contiguous subarrays, find the product of each of these subarrays and return the maximum product from these results.

Constraints:

- Time Complexity: $O(N^2)$

- Auxiliary Space: $O(1)$

Maximum Product Subarray



Explore | Expand | Enrich

Program

mps2.java

This solution is an improvement to the previous naive solution.
It achieves the goal of linear time complexity constraint

Constraints:

- Time Complexity: $O(N)$
- Auxiliary Space: $O(1)$

Introduction

In this problem, we are given a 2D array (matrix) of any order.

Our task is to create a program that finds the maximum sum of the hourglass.

Out of all the possible hourglass sums of a given $m \times n$ ordered matrix, we have to print the maximum of all the sums

In our problem, Hourglass is a 7 element shape made in the matrix in the following form:

```
X X X
  X
X X X
```



Maximum sum of hourglass in matrix



Example

Consider the matrix:-

```
2 4 0 0
0 1 1 0
4 2 1 0
0 3 0 1
```

The hourglasses are :

```
2  4  0    0  1  1
   1          2
4  2  1    0  3  0
```

```
4  0  0    1  1  0
   1          1
2  1  0    3  0  1
```

So, an hourglass can be created using the following indexes:-

$\text{matrix}[i][j]$

$\text{matrix}[i][j+1]$

$\text{matrix}[i][j+2]$

$\text{matrix}[i+1][j+1]$

$\text{matrix}[i+2][j]$

$\text{matrix}[i+2][j+1]$

$\text{matrix}[i+2][j+2]$

We will find the sum of all these elements of the array from $[0][0]$ to $[R-2][C-2]$ starting points. And then find the maxSum for all these hourglasses created from array elements.

Maximum sum of hourglass in matrix



Explore | Expand | Enrich

Program

hourglass1.java

The first line contains m and n which are the number of rows and columns of the matrix.
The next m lines contain n space separated integers which are the elements of the matrix

Test Cases

Input

```
5 5
1 2 4 5 6
7 5 2 3 6
0 0 0 0 0
7 5 1 3 5
0 0 0 0 0
```

Output

27

Explanation

All the possible hourglasses are given. The bold hourglass is the one with the maximum sum.

```
1 2 4 2 4 5 4 5 6
  5 2 3
0 0 0 0 0 0 0 0 0

7 5 2 5 2 3 2 3 6
  0 0 0
7 5 1 5 1 3 1 3 5

0 0 0 0 0 0 0 0 0
  5 1 3
0 0 0 0 0 0 0 0 0
```



/ethnuscodemithra



Ethnus Codemithra



/ethnus



/code_mithra

THANK YOU

<https://www.codemithra.com/>



Explore | Expand | Enrich



codemithra@ethnus.com



+91 7815 095 095



+91 9019 921 340