Basic Programming Concepts

Topics

1. What is an Algorithm?
2. Interpreters Vs Compilers
3. Hardware Operations
4. Computing Power
5. Time and Space Limits for Programs
6. Process Memory Layout
7. Bits/Bytes?
8. 32bit vs 64bit architecture
9. One’s Complement and Two’s Complement

What is an algorithm?

- "An algorithm is the step-by-step instructions to solve a problem".
- "An algorithm is a well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transforms the input into the output".

Example: We might want to sort a sequence of numbers in a non-decreasing order.
Input: Sequence of n numbers (a_1, a_2, a_3,…., a_n)
Output: A permutation of the input sequence such that $b_1 \leq b_2 \leq b_3 \leq \ldots. \leq b_n$

Interpreters Vs Compilers

We generally write a computer program using a high-level language. A high-level language is one which is understandable by us humans. It contains words and phrases from English (or other) language. But a computer does not understand high-level language. It only understands program written in 0's and 1's in binary, called the machine code. A program written in high-level language is called a source code. We need to convert the source code into machine code and this is accomplished my compilers and interpreters. Hence, a compiler or an interpreter is a program that converts program written in high-level language into machine code understood by the computer.

The difference between an interpreter and a compiler is given below:

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Interprets the same program statements each time it meets them eg: instructions within a loop.	The generation of machine code happens only once.
Programming language like Python, Ruby uses interpreters.	Programming language like C, C++ uses compilers.

Another Beautiful Answer on Quora:
https://www.quora.com/What-are-the-differences-between-a-compiler-an-interpreter-and-an-assembler

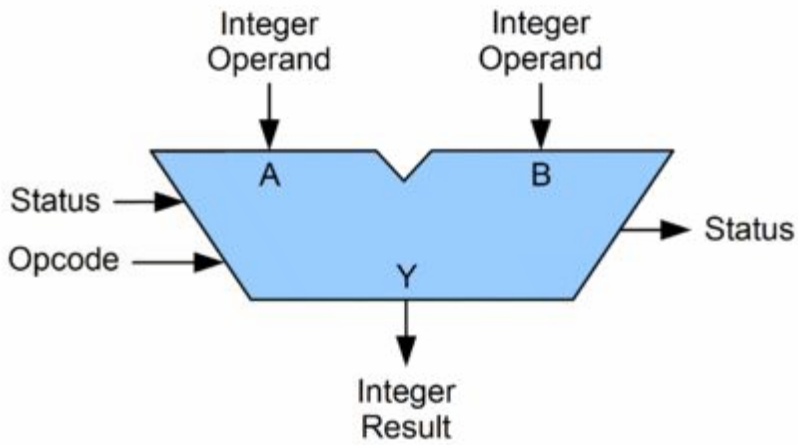
Java's Execution Environment:

1. http://stackoverflow.com/questions/7674839/is-the-jvm-a-compiler-or-an-interpreter
2. https://www.quora.com/Why-does-Java-use-both-compiler-as-well-as-Interpreter

Hardware Operations

An arithmetic-logic unit (ALU) is the part of a computer processor (CPU) that carries out arithmetic and logic operations on the operands in computer instruction words. In some processors, the ALU is divided into two units, an arithmetic unit (AU) and a logic unit (LU). Some processors contain more than one AU - for example, one for fixed-point operations and another for floating-point operations.

An ALU is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPUs, and graphics processing units (GPUs). A single CPU, FPU or GPU may contain multiple ALUs.



Opcode - The opcode input is a parallel bus that conveys to the ALU an operation selection code, which is an enumerated value that specifies the desired arithmetic or logic operation to be performed by the ALU.
Status - The status outputs are various individual signals that convey supplemental information about the result of an ALU operation. Ex: Carry-out, Zero, Negative, Overflow etc.

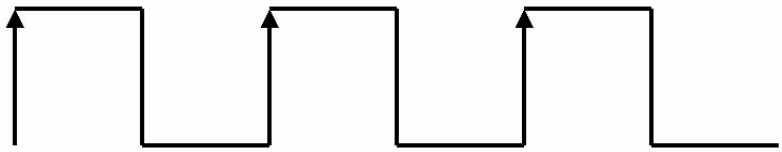
More Details:

1. https://en.wikipedia.org/wiki/Arithmetic_logic_unit
2. http://study.com/academy/lesson/arithmetic-logic-unit-alu-definition-design-function.html

Also, most processors do not have a modulo operator, many even do not have a divide. Now, if you have burned the hardware to make a divide, you might be willing to go that extra mile to add modulo as well. Most processors take your question to the next level, why would you implement a divide in hardware when it can be done in software. The answer to your question is most processor families do not have a modulo, and many do not have a divide because it is not worth the chip real estate, power consumed, etc compared to the software solution. The software solution is less painful/costly/risky.

Computing Power and Time Limit for Programs:

Clock is a signal used to sync things inside the computer. Take a look at Figure 2, where we show a typical clock signal: it is a square wave changing from “0” to “1” at a fixed rate. On this figure you can see three full clock cycles (“ticks”). The beginning of each cycle is when the clock signal goes from “0” to “1”; we marked this with an arrow. The clock signal is measured in a unit called Hertz (Hz), which is the number of clock cycles per second. A clock of 100 MHz means that in one second there is 100 million clock cycles.



Another way to define it is to say that Clock cycle is the amount of time between two pulses of an oscillator. Generally speaking, the higher number of pulses per second, the faster the computer processor will be able to process information.

CPU’s clock speed, or clock rate, is measured in Hertz — generally in gigahertz, or GHz. A CPU’s clock speed rate is a measure of how many clock cycles a CPU can perform per second. For example, a CPU with a clock rate of 1.8 GHz can perform 1,800,000,000 clock cycles per second.

This seems simple on its face. The more clock cycles a CPU can perform the more things it can get done, right? Well, yes and no.

This may not seem obvious at first, but it’s actually for a very simple reason. It’s because the speed of a computer is also influenced by other factors, such as the efficiency of the processor, the bus architecture, the amount of memory available, and the software that is running on the computer. Some processors can perform more instructions per clock cycle than other processors, making them more efficient than other processors with higher clock speeds. All other things being equal, fewer clock cycles mean the CPU requires less power and produces less heat.

In addition, modern processors also have other improvements that allow them to perform faster. This includes additional CPU cores and larger amounts of CPU cache memory that the CPU can work with. For example, if the cache is not sufficient the computer will wait until the instruction or the data will be retrieved from the RAM and the processor speed will be lower.
The Random Access Memory (RAM) modules inside a computer store temporary data for the Central Processing Unit. Data that cannot be stored inside the RAM must be accessed from the hard drive, slowing down the machine. More RAM means that more programs and larger files can be used simultaneously without any impact on performance. RAM is important because it eliminates the need to “swap” programs in and out.

Time and Space Limits for Programs

Different algorithms have different constants. Some operations are slow - modulo (that slows down the program significantly), if-else conditions, standard minimum/maximum etc., while others, like bitwise operations, are really fast and a good way to optimize a program (they can be used for faster min/max functions, for example).

Then, there are small tricks like adding a condition which for most test data makes the code skip some redundant operations, or noticing that it's hard to make test data for which some assumption doesn't hold. But that's another topic!The point is that sometimes, these tricks (they can be in your code even if you don't know about it) can improve your constant factor, or even complexity, noticeably.

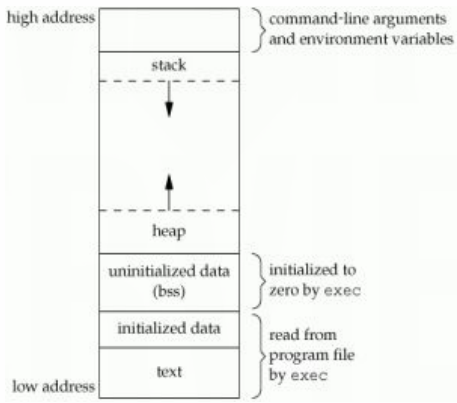
In short, it all depends on experience. When you code more algorithms and note how fast they ran, you'll be able to estimate better what could pass later, even without yet coding it.

- Approximately you can perform about **10^8 operations in one second**. So with that you can estimate whether your algorithm will run in time or not.
- Approximately your program can consume 10^7 bytes of memory (≈ 10MB). From there on it goes on to allocate memory on your virtual memory.

Time Limit Example:
N = 10^9 ≈ 1 second
N*N = 10^{18} ≈ 10^9 Seconds ≈ 10^4 days ≈ 30years
NlogN = 10^9 * 30 ≈ 30 seconds

Memory Usage Example:
10^6 ≈ 1 MB
10^9 ≈ 1 GB
10^{10} ≈ 10 GB

Process Memory Layout



A typical memory layout of a running process

1. **Text Segment:**
   - sections of a program in memory which contains executable instructions.
   - As a memory region, a text segment may be placed below the heap or stack in order to prevent heaps and stack overflows from overwriting it.
2. **Initialized Data Segment:**
   - contains the global variables and static variables that are initialized by the programmer.
   - This segment can be further classified into initialized read-only area and initialized read-write area.
3. **Uninitialized Data Segment:**
   - often called the "bss" segment, named after an ancient assembler operator that stood for "block started by symbol."
   - Data in this segment is initialized by the kernel to arithmetic 0 before the program starts executing
   - contains all global variables and static variables that are initialized to zero or do not have explicit initialization in source code.
4. **Stack:**
   - The stack area contains the program stack, a LIFO structure, typically located in the higher parts of memory
   - A "stack pointer" register tracks the top of the stack; it is adjusted each time a value is "pushed" onto the stack.
   - The set of values pushed for one **function call** is termed a "stack frame". A stack frame consists at minimum of a return address.
   - Each time a function is called, the address of where to return to and certain information about the caller's environment, such as some of the machine registers, are saved on the stack. The newly called function then allocates room on the stack for its automatic and temporary variables. This is how recursive functions in C can work.
5. **Heap:**
   - Heap is the segment where dynamic memory allocation usually takes place.
   - The heap area begins at the end of the BSS segment and grows to larger addresses.
   - The Heap area is managed by malloc, realloc, and free, ex:
   ptr=(int*)malloc(100*sizeof(int));
   - The Heap area is shared by all shared libraries and dynamically loaded modules in a process.

32bit vs 64bit architecture

- The key difference: 32-bit processors are perfectly capable of handling a limited amount of RAM, and 64-bit processors are capable of utilizing much more.

- As a general rule, if you have less than 4 GB of RAM in your computer, you don't need a 64-bit CPU, but if you have 4 GB or more, you do.
- More bits means our system can point to or address a larger number of locations in physical memory

Bits/Bytes

- A bit (short for BInarydigiT) is the smallest unit of data in a computer. A bit can have a binary value of either 0 or 1. Binary means that there are only two logical (i.e. on/off, true/false) choices. Until the value is actually determined, it can have two states.
- A byte is composed of 8 bits. Half a byte (4 bits) is called a nibble.
  8 bits = 1 Byte
  1,024 Bytes = 1 Kilobyte
  1,024KB = 1 Megabyte
  1,024MB = 1 Gigabyte
  1,024GB = 1 Terabyte
  1,024TB = 1 Petabyte
  1,024PB = 1 Exabyte
  1,024EB = 1 Zetabyte
  1,024ZB = 1 Yottabyte

One's Complement and Two's Complement

- Two's complement is especially important because it allows us to represent signed numbers in binary, and one's complement is the interim step to finding the two's complement.
- If all bits in a byte are inverted by changing each 1 to 0 and each 0 to 1, we have formed the one's complement of the number.
- Twos Complement is the negative representation of the number
  Example:
  65 -> 01000001
  -65 -> 10111111 [2's complement of 65]
- With a system like two's complement, the circuitry for addition and subtraction can be unified, whereas otherwise they would have to be treated as separate operations.

Tips:

1. Powers of 2 should be at the tip of your tongue – 1,2,4,8,16,32,64,128,256,512,1024
   $2^{10} \approx 10^3$
   $2^{20} \approx 10^6$
   $2^{30} \approx 10^9$
   $2^{60} \approx 10^{18}$

Important Links:

1. http://www.c-jump.com/CIS77/CPU/VonNeumann/lecture.html
2. http://www3.ntu.edu.sg/home/ehchua/programming/java/datarepresentation.html
3. https://en.wikipedia.org/wiki/Booting
4. http://electronics.stackexchange.com/questions/123760/how-can-a-cpu-deliver-more-than-one-instruction-per-cycle
5. http://www.geeksforgeeks.org/memory-layout-of-c-program/
6. http://www.tenouk.com/ModuleZ.html
7. https://www.cs.cornell.edu/~tomf/notes/cps104/twoscomp.html
8. http://stackoverflow.com/questions/21434682/what-do-x86-64-i386-ia64-and-other-such-jargons-stand-for
9. https://superuser.com/questions/179919/x86-vs-x64-why-is-32-bit-called-x86
10. http://www.pcadvisor.co.uk/feature/pc-components/32-bit-vs-64-bit-3584953/
11. https://en.wikipedia.org/wiki/X86-64
12. https://www.quora.com/Processor-Architecture-What-is-the-difference-between-32-bit-and-64-bit-CPUs-How-does-the-performance-increase-for-64-bit-CPUs
13. http://stackoverflow.com/questions/3072444/long-long-int-on-32-bit-machines
14. https://en.wikipedia.org/wiki/Physical_Address_Extension
15. https://en.wikipedia.org/wiki/Superscalar_processor
16. https://en.wikipedia.org/wiki/Instruction_pipelining
17. https://en.wikipedia.org/wiki/Hyper-threading
18. https://www.quora.com/What-is-the-difference-between-multiprogramming-multitasking-multiprocessing-and-multiexecution
19. https://www.quora.com/What-is-the-difference-between-a-multi-processor-system-and-a-multicore-processor-system