UNIT I ROLE OF ALGORITHMS IN COMPUTING & COMPLEXITY ANALYSIS

Algorithms – Algorithms as a Technology – Time and Space complexity of algorithms – Asymptotic analysis – Average and worst-case analysis – Asymptotic notation – Importance of efficient algorithms – Program performance measurement – Recurrences: The Substitution Method – The Recursion – Tree Method – Data structures and algorithms.

ALGORITHMS

An algorithm is any well-defined computational procedure that takes some value, or set of values as input and produces some value or set of values, as output in a finite amount of time.

An algorithm is a sequence of computational steps that transform the input into the output. An algorithm as a tool for solving a well-specified computational problem.

Input: A sequence of n numbers $\{a_1,a_2,a_3,.... a_n\}$.

Output: A permutation (reordering) $\{a_1',a_2',a_3',.... a_n'\}$ of the input sequence such that $a_1' \leq a_2' \leq a_3 \leq .... \leq a_n'$

Thus, given the input sequence {31, 41, 59, 26, 41, 58}, a correct sorting algorithm returns as output the sequence {26, 31, 41, 41, 58, 59}. Such an input sequence is called an instance of the sorting problem. In general, an instance of a problem consists of the input needed to compute a solution to the problem.

An algorithm for a computational problem is correct if, for every problem instance provided as input, it finishes its computing in finite time and outputs the correct solution to the problem instance. A correct algorithm solves the given computational problem. An incorrect algorithm might not halt at all on some input instances, or it might halt with an incorrect answer. An algorithm can be specified in English, as a computer program, or even as a hardware design. The only requirement is that the specification must provide a precise description of the computational procedure to be followed.

ALGORITHM AS A TECHNOLOGY

Computers may be fast, but they are not infinitely fast. And memory may be cheap, but it is not free. Computing time and space in memory is therefore a bounded resource. These resources should be used wisely, and algorithms should be efficient in terms of time or space.

Efficiency:

Different algorithms are created to solve the same problem but they differ in their efficiency. These differences can be much more significant than differences due to hardware and software.

Let us see two algorithms for sorting.

- Insertion sort takes time equal to c1n2 to sort n items, where c1 is a constant that does not depend on n. It takes time proportional to n2.
- Merge sort takes time equal to c2n lg n, where lg n stands for log2 n and c2 is another constant that also does not depend on n.

Insertion sort usually has a smaller constant factor than merge sort, so that c1 < c2. The constant factors can be far less significant in the running time than the dependence on the input size *n*. Merge sort has a factor of lg n in its running time, insertion sort has a factor of *n*, which is much larger. For example, when *n* is 1000, lg n is approximately 10, and when *n* is 1,000,000, lg n is approximately only 20.

Although insertion sort is usually faster than merge sort for small input sizes, once the input size n becomes large enough, merge sort log n will be faster than n.

For a concrete example, let us pit a faster computer (computer A) running insertion sort against a slower computer (computer B) running merge sort. They must sort an array of one million numbers. Suppose that computer A executes one billion instructions per second and computer B executes only ten million instructions per second, so that computer A is 100 times faster than computer B in raw computing power. Suppose an efficient programmer codes insertion sort in machine language for computer A, and the resulting code requires $2n^2$ instructions to sort n numbers. (Here, c1 = 2.) On the other hand, an average programmer codes Merge Sort for computer B in high-level language with an inefficient compiler, and the resulting code takes $50n \lg n$ instructions (c2 = 50). To sort one million numbers, computer A takes

$$\frac{2 \cdot (10^6)^2 \text{ instructions}}{10^9 \text{ instructions/second}} = 2000 \text{ seconds},$$

while computer B takes,

$$\frac{50 \cdot 10^6 \lg 10^6 \text{ instructions}}{10^7 \text{ instructions/second}} \approx 100 \text{ seconds}.$$

By using an algorithm whose running time grows more slowly, even with a poor compiler, computer B runs 20 times faster than computer A. The advantage of merge sort is even more pronounced when we sort ten million numbers: where insertion sort takes approximately 2.3 days, merge sort takes under 20 minutes. In general, as the problem size increases, merge sort advantages more.

Algorithms and other technologies:

The example above shows that algorithms are a technology. Total system performance depends on choosing efficient algorithms as much as on choosing fast hardware. Algorithms are truly important on contemporary computers in light of other advanced technologies, such as

- advanced computer architectures and fabrication technologies
- easy-to-use, intuitive, graphical user interfaces (GUIs)
- object-oriented systems
- integrated web technologies
- fast networking, both wired and wireless
- machine learning
- mobile devices

Although there are some applications that do not explicitly require algorithmic content at the application level (e.g., some simple web-based applications), but require a degree of algorithmic content on their own. For example, consider a web-based service that determines how to travel from one location to another. Its implementation would rely on fast hardware, a graphical user interface, wide-area networking, and also possibly on object orientation. However, it also requires algorithms for finding routes (shortest-path algorithm), rendering maps, and interpolating addresses.

An application that does not require algorithmic content at the application level depends heavily upon algorithms. The hardware design of applications uses algorithms. The design of any GUI of applications relies on algorithms. Does the application rely on networking? Routing in networks relies heavily on algorithms. The compiler, interpreter, or assembler used to process the application makes extensive use of algorithms.

Furthermore, with the ever-increasing capacities of computers, we use them to solve larger problems than ever before. As we saw in the above comparison between insertion sort and merge sort, it is at larger problem sizes that the differences in efficiencies between algorithms become particularly prominent.