CONTENTS

Chapters and Topics **page**

CHAPTER – 19

ADVANCED SORTING OPTIONS

EFFICIENCY CONSIDERATIONS

There are three most important factors to be considered for sorting:

- the length of time that must be spent by the programmer in coding a particular program
- the amount of machine time necessary for running the program
- the amount of space necessary for the program.

1. For small files sophisticated sorting techniques designed to minimize space and time requirements are usually worse or only marginally better in achieving efficiencies than simpler, less efficient methods.

2. Sorting programs, which will run only once and there is sufficient machine time and space in which to run it, it is unwise to investigate the best methods to obtain efficiency.

3. The time required to sort a file of size n records is not measured in actual time units taken by a machine. The number of critical operations performed measures the time efficiency. The critical operations are key comparisons, movement of records or pointers to records, or interchanges of records.

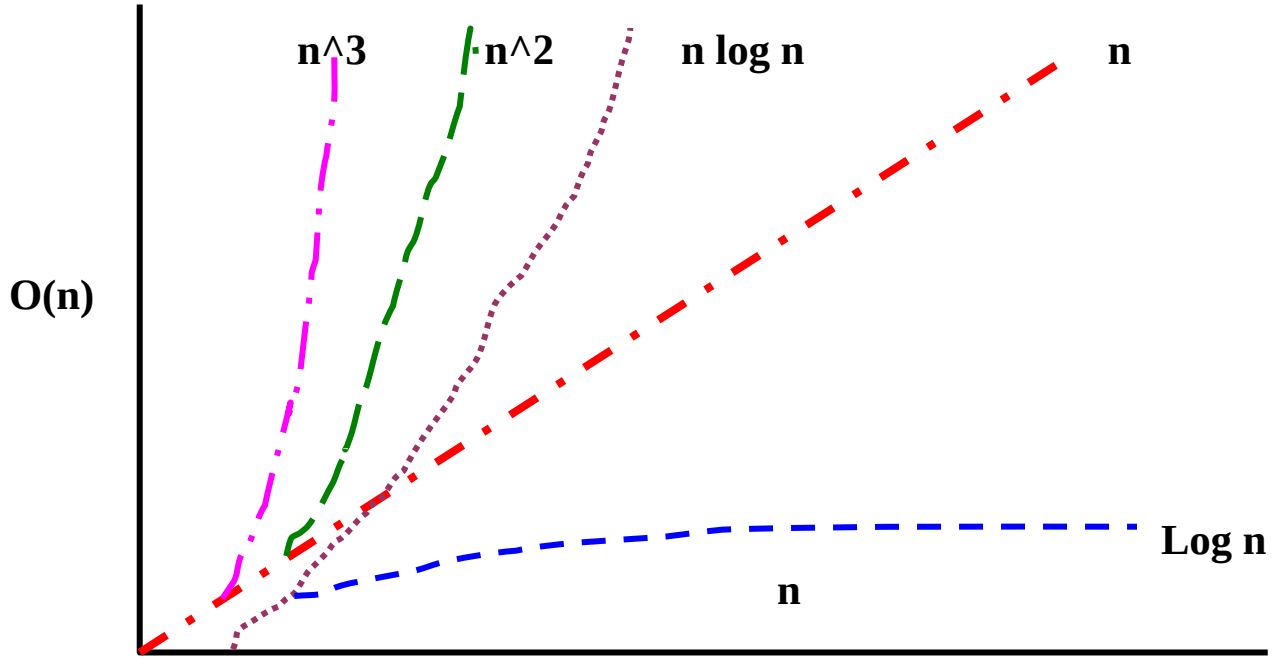
4. Mathematical analysis of sorting often results in a formula in terms of exponents of the file size n giving the average time required for a particular sort function.

5. As the file size n grows larger the effect of the associated constants in the formula becomes less dominant compared to n. Finally the highest power of n becomes dominant as n grows larger and the complexity is assumed approximately proportional to the highest exponent of n.

6. Algorithms are compared with their complexities. The big *O* notation is the usual way to address the complexity of an algorithm.

With the speed of computers today, we are not concerned with an exact measurement of an algorithms's efficiency as much as we are with its general magnitude. The number of statements executed in a function for n elements of data is a function of the number of elements, expressed as f(n). Complete measure of the efficiency is not necessary, only the factor that determines the magnitude is important. This factor is the big-O, as of-the-order-of , and expressed as O(n).

Efficiency	Big-O	Iterations	Est. Time
Logarithmic	O(log n)	14	microseconds
Linear	O(n)	10,000	.1 seconds
Linear Logarithmic	O(n(log n))	140,000	2 seconds
Quadratic	O(n^2)	10,000^2	15 – 20 min.
Ploynomial	O(n^k)	10,000^k	hours
Exponential	O(C^n)	2^10,000	intractable
Factorial	O(n!)	10,000!	intractable



EFFICIENCY OF SORTING

Using this concept of the order of a sort, various sorting techniques can be compared and classified them as being "**good**" or "**bad**" in general terms. One might hope to discover the "**optimal**" sort that is *O*(*n*) regardless of the contents or order of the input. Unfortunately, however, it can be shown that no such generally useful sort exists. Most of the classical sorts considered have time requirements that range from *O*(*n***log n*) to *O*(*n****2*). A sort should not be selected simply because it is *O*(*n***log n*). The relation of the file size n and the other terms constituting the actual sorting time must be known. Some of the issues that play an insignificant role for large values of n may play a very dominant role for small values of n. All issues must be considered before an intelligent sort selection can be made. A second method of determining time requirements of a sorting technique is to actually run the program and measure its efficiency.

In most cases the time needed by a sort depends on the original sequence of the data. For some sorts, input data which is almost in sorted order can be completely sorted in time *O*(*n*), where as input data that is in reverse order needs time that is *O*(*n****2*). For other sorts the time required is *O(n*log n*) regardless of the original order of the data. The choice of a sort must, of necessity, depend on the specific circumstances. Once a particular sorting technique has been selected, the program should then be made as efficient as possible.

Space constraints are usually less important than time considerations. One reason for this is that, for most sorting programs; the amount of space needed is closer to *O*(n) than to *O*(n**2). A second reason is that if more space is required it can almost always be found in auxiliary storage. An ideal sort is an in-place sort whose additional space requirements are *O(*1).