Question 5) Do a bit of research and find out on what factors does the actual time taken by a program depends on. Use these factors as an argument to justify why it is not feasible to do empirical analysis when the goal is to compare two algorithms.

Answer)

Clearly the more quickly a program/function accomplishes its task the better.
The actual running time depends on many factors:

- The speed of the computer: cpu (not just clock speed), I/O, etc.
- The compiler, compiler options .
- The quantity of data - execution search a long list or short.
- The actual data - execution in the sequential search if the name is first or last.

<u>MEASURING TIME</u>
There are three relevant measures of time taken by a process.

- User time measures the amount of time (number of cycles taken by a process in "user mode."
- System time the time taken by the kernel executing on behalf of the process.
- Wallclock time is the total "real" time taken to execute the process. Generally speaking, user time is the most relevant, though it ignores some important operations (I/O, etc.). Wallclock time should be used cautiously/sparingly, but may be necessary for assessment of parallel codes.

<u>TEST SET</u>

- It is crucial to choose your test set well.
- The instances must be chosen carefully in order to allow proper conclusions to be drawn.
- We must pay close attention to their size, inherent difficulty, and other important structural properties.
- This is especially important if we are trying to distinguish among multiple algorithms.

When analyzing for time complexity we can take two approaches:

- Order of magnitude/asymptotic categorization -
    - This uses coarse categories and gives a general idea of performance. If algorithms fall into the same category, if data size is small, or if performance is critical, then the next approach can be looked at more closely.

Estimation of running time - By analysis of the code we can do:

- operation counts - select operation(s) that are executed most frequently and determine how many times each is done.
- step counts - determine the total number of steps, possibly lines of code, executed by the program.
- Empirical analysis is a relative analysis. The time complexity of an algorithm using a emperical analysis differ from system to system.

- Empirical analysis introduces many more factors that need to be controlled for in some way.
- Test platform (hardware, language, compiler)
- Measures of performance (what to compare)
- Benchmark test set (what instances to test on)
- Algorithmic parameters
- Implementational details

How do you compare two algorithms for solving some problem in terms of efficiency?

We could implement both algorithms as computer programs and then run them on a suitable range of inputs, measuring how much of the resources in question each program uses. This approach is often unsatisfactory for four reasons. First, there is the effort involved in programming and testing two algorithms when at best you want to keep only one. Second, when empirically comparing two algorithms there is always the chance that one of the programs was "better written" than the other, and therefore the relative qualities of the underlying algorithms are not truly represented by their implementations. This can easily occur when the programmer has a bias regarding the algorithms. Third, the choice of empirical test cases might unfairly favor one algorithm. Fourth, you could find that even the better of the two algorithms does not fall within your resource budget. In that case you must begin the entire process again with yet another program implementing a new algorithm. Perhaps the problem becomes simply too difficult for any implementation.

These problems can often be avoided by using asymptotic analysis. Asymptotic analysis measures the efficiency of an algorithm, or its implementation as a program, as the input size becomes large. It is actually an estimating technique and does not tell us anything about the relative merits of two programs where one is always "slightly faster" than the other. However, asymptotic analysis has proved useful to computer scientists who must determine if a particular algorithm is worth considering for implementation.

Measuring the physical running time has several disadvantages, both principal (dependence on a particular machine being the most important of them) and technical, not shared by counting the executions of a basic operation. On the other hand, the physical running time provides very specific information about an algorithm’s performance in a particular computing environment, which can be of more importance to the experimenter than, say, the algorithm’s asymptotic efficiency class. In addition, measuring time spent on different segments of a program can pinpoint a bottleneck in the program’s performance that can be missed by an abstract deliberation about the algorithm’s basic operation.

The principal strength of the aymptotic analysis is its independence of specific inputs; its principal weakness is its limited applicability, especially for investigating the average-case efficiency. The principal strength of the empirical analysis lies in its applicability to any algorithm, but its results can depend on the particular sample of instances and the computer used in the experiment.