School of Computer Science and Applied Mathematics
University of the Witwatersrand, Johannesburg

Advanced Analysis of Algorithms
Assignment - Supplementary Notes

Dr. Hima Vadapalli

Semester II, 2017

2 Empirical analysis of algorithms

2.1 Introduction

Empirical analysis of algorithms is an important idea in its own right. Theoretical analysis does not give much of an idea of how well a given algorithm will perform in a specific situation [2, 3] – empirical analysis would help here. Empirical analysis is also important in comparing two algorithms which may or may not have the same order of complexity – when would one use one and not the other? As an example of this consider Insertion Sort ($O(n^2)$) and Mergesort ($O(n \lg n)$), where for small input instances Insert Sort could be a better algorithm to use and is certainly easier to understand and code (and has a good best case performance). Similar examples can be found in other areas of algorithms.

2.2 Why would we do empirical analysis?

- to check we got the theoretical analysis right
- to get an idea of the performance of an algorithm where the theoretical analysis is really difficult
- to get an idea of the running time of a program where we don't know the detailed working of the program,
- to gain understanding of a particular algorithm (on a particular machine or operating on particular input instances)
- to get exposure to some experimental aspects of computer science
- to compare two or more algorithms
- to choose the best algorithm for some certain circumstances

2.3 Doing Empirical Analysis

In order to do empirical analysis to determine the running time of an algorithm in a specific situation or to verify and expand on the theoretical analysis of an algorithm we need to

- understand the theoretical analysis
- decide on what should be measured and how this can be used to verify the theoretical analysis
- decide on appropriate hardware
- decide on an appropriate implementation language
- decide on appropriate data structures
- implement the algorithms
- implement some form of timing device
- create the input data sets necessary to produce the measurements we need
- measure the performance of the algorithm on the different input data sets created to meet our aim

- interpret the results in terms of our aim
- relate the results to the theoretical analysis – actually verify best, average and worst case analysis (or explain why, if the experiment doesn't verify the theoretical results)

Few of these tasks are trivial. To deal with them adequately knowledge and understanding of a number of theoretical concepts/areas are required – asymptotic notation; probability theory; machine architecture; programming languages; compilers; data structures and machine representation of these; and experimental statistics. In addition, programming and presentation skills are necessary to complete the task successfully.

One does not have to consider all of these aspects in every empirical analysis but it is essential to be aware of them and to be able to determine which require more attention in a particular case. It is also important to note that the decisions made in some regard may have influence on other decisions and that this is not a linear process.

To demonstrate the complexity of some of these tasks let us consider an experiment to verify the theoretical average case analysis of the *Linear Search* algorithm given in Algorithm 1.

Algorithm 1 linearSearch($myList$, n, key)

Input: $myList, n, key$ where n is an array with n entries (indexed $0 \ldots (n-1)$), and key is the item sought.
Output: $index$ the location of key in $myList$ (-1 if key is not found).

```
01  index ← 0
02  While index ≤ (n − 1) and myList[index] ≠ key
03    index ← index + 1
04  If index > n − 1
05    Then index ← −1
06  Return index
```

[1] gives the average case analysis for this algorithm in detail – if the number being searched for has a probability of q of not being in the list then $A(n) = q\frac{(n+1)}{2} + (1-q)n$, i.e. $O(n)$. This is based on the probability of the number being searched for being in each position in the list or not in the list at all. Clearly an understanding of asymptotic notation and some probability theory is required to understand this analysis.

In our experiment we could decide that for each instance size we need to test every possibility and then average the resulting running times for each case. This would be the most exact thing to do but it would probably not be practical for any reasonably sized lists. We thus have to try to approximate this average case performance. This then means making decisions – which must be clear and well motivated – about how many cases to test and how to generate these test cases. This means having an understanding of probability theory and statistics as well as knowing something about pseudo-random number generators. In addition, some understanding of the machine architecture is required to make sure we can handle the test cases we decide on.

Once we have made these decisions we still have lots of other factors to consider and this is in the case of an extremely simple algorithm. More detail is given in the sections below where we discuss the design of an experiment to verify the theoretical analysis of the linear search algorithm.

2.3.1 Understanding the theoretical analysis

The theoretical analysis of the linear search algorithm is given in [1]. From this analysis we know that the best case for the linear search is when the element being searched for is in the first position in the list. The algorithm is O(1) in this case. We also know that the worst case occurs when the element being searched for is in the last position in the list or not in the list at all. In this case the algorithm is in O(n).

The average case analysis arises from the average cost for the element being in *all* other positions in the list. This gives an O(n) running time but with a smaller constant factor than the worst case.

The theoretical analysis gives us some idea of what we should expect if we actually measure the running time of the algorithm on appropriately chosen data, plot these running times on a graph relating running time to input size and then fit curves to these data points. The best case data should result in a constant function. The average and worse case should behave like functions of the form $f(n) = An + B$ with the growth in the average case being slower than that for the worst case.

In the section below we discuss how appropriate data can be selected in this case.

2.3.2 Deciding on what we need to measure

Clearly we want to measure the running time of an (appropriate) implementation of the algorithm on data sets which have been created to reflect the best, worst and average case performance of the algorithm. In order to study the asymptotic behaviour of the algorithm we must test lists of different sizes where n is large.

The best case is easy because we just have to make sure that the element we are looking for is at the beginning of our list – it is essentially unimportant what is in the rest of the list. We do, however, still have to make sure that we test with increasing sizes of list to make sure that the performance of the algorithm does not depend in some way on the size of the list.

The worst case is also relatively easy as we just have to make sure that the element we are looking for is at the end of our list and does not appear anywhere else in the list or is not in the list at all. We could test both of these cases but it is probably sufficient to pick one and test that case only.

For the average case performance the theoretical analysis tells us that we should measure the running time for the algorithm to find the first (or only) occurrence of the search element in each position of the list and then average these measurements. The question arises as to whether it would be feasible to do this when the list is long. Even for the very simple linear search algorithm the cost of doing this could be prohibitive – it would require a thousand runs of the implementation of the algorithm to calculate the average cost of searching a list of one thousand elements and this would give one data point on our plot of running time versus size of input. This means that we need to find some way of approximating the average case performance of the algorithm without compromising the integrity of our experiment.

Essentially what we want to do is to generate a sufficient number of test cases in the hope of getting a "representative sample" for each data point.

Some questions arise – What is a sufficient number of cases? How could we decide if the sample is representative? Clearly here, 3 different input keys for each size of list would not give us a good idea, whereas 30 or 300 might well do so.

Once we have decided on how to generate the test data for each case we still need to decide on how many data points are necessary in order for us to do the "curve fitting" to determine the performance of the algorithm. We need to answer the question "How many different list sizes should be measured?". This can be answered by having an understanding of the curve fitting approach/software that you will be using.

Another thing which we must worry about is what are appropriate sizes for the input instances. We are only concerned about asymptotic performance so we really only need to measure the running times of representative samples where n is big enough. We need to answer the question "What values of n should we be testing?".

In our example of verifying the theoretical analysis of the linear search we may decide that we will test lists of size $n = 10000, 15000, 20000, 25000, 30000, 35000, 40000, 45000, 50000$. This would give us 9 data points to use in the curve fitting and these instances would seem to be big enough for us to not have to worry about overheads in the measuring of the running times. If we then use 10% of the list size as the number of random positions of the number that we are looking for then we would be getting a reasonably representative sample of our data.

Note that there are situations where we might want to do an empirical analysis but that timing the running of an implementation is not sufficient for us. We might actually not have a very good idea of how often the "basic operation" in our theoretical analysis gets executed. We might be able to come up with an upper bound but not really know how tight this bound is. In this case we might actually count these operations in our experiment.

There are also cases when counting the basic operations is not really warranted. We sometimes do empirical work to verify our theory. In the theory we develop a model of the algorithm's performance by picking a basic operation and counting how many of these are done. Our assumption is that that basic operation gives an idea of the performance of the algorithm. We might be wrong about this. So why would we now want to make the same "mistake" in our experiment. It would be better to look at some other measure so that we really can check the theory. Also if we aren't wrong and in this case (linear search) we probably can be pretty sure we aren't then why do an experiment which involves counting when we can just do the counting.

Decisions on what to measure when must be made with careful consideration of what you are trying to learn.

2.3.3 Deciding on appropriate hardware and programming language

To some extent we can argue that deciding on appropriate hardware and programming language is less important than some of the other decisions that we have to make. This is so because we are more interested in the rate of growth of the performance of the algorithm than the actual running times. There could be instances when we are actually interested in the running times on a particular machine but then some of the decisions have been made for us.

So what criteria do we using for deciding on hardware and programming language. Some suggestions are given below.

- Availability – clearly doing the experiment on a machine that is readily available using a language that is already available on that machine has advantages unless there is some reason (related to one of the other decisions we need to make) not to use those resources.
- Familiarity – a similar comment to the one above applies.
- Ease of use – again this is a pretty obvious comment
- Measurements are unlikely to be affected by outside influences – if you are running your test program on a networked or multitasking machine then your running times might be affected by the performance of other users' job.
- Specifics of the programming language will not affect your measurements – Java does automatic garbage collection, could this affect your measurements?
- Other factors?

2.3.4 Deciding on appropriate data structures

In order to decide on what data structures to use we need to have a good understanding of the algorithm and how the choice of data structure could affect the performance of the algorithm. In some instances the difference might not really be important. If we are measuring the performance of sorting algorithms then the difference in sorting lists represented as arrays of integers, arrays of reals and arrays of records would not affect the growth rate of the algorithm but would certainly affect the running time and the constant factors in the curve fitting. This may or not be acceptable depending on what you are trying to achieve – the fastest implementation or an understanding of the rate of growth of the running time of the algorithm.

2.3.5 Implement the algorithms

Once the decisions about hardware and programming have been made we would then implement the algorithm in the chosen language on the chosen machine. Even at this level we still have decisions to make. Clearly these decisions are at the detail level but they could still have impacts on our measurements and we should be aware of them.

2.3.6 Implement some form of timing device

In doing this experiment we want to measure the running time of the algorithm that we are concerned about so we must be sure we are measuring what we want to measure – the running time of the search and not the set up, output, waiting time, etc. We should also worry about what exactly we are timing – is it the cpu usage time or the elapsed time? Does it (or could it) make a difference?

We also need to be as sure we can that the timing will not be affected by operating system or network issues – multitasking, paging, garbage collection, caching, etc. On shared machines or networked machines this is often hard to achieve. Certainly you could disconnect the machine from the network and kill all other processes. This would help. You could also repeat measurements and hope that you will get an accurate reflection of the running time in that way.

2.3.7 Create the data

Creating the data for your experiment will essentially mean writing code to generate the input data to test the various situations which you have already identified.

In this example this would be relatively simple.

1. best case – simple, write a program which ensures that the number you are looking for is in the first position in the list and then fills the rest of the list with random numbers.

2. worst case – similar (but slightly different)
3. average case – harder, can again use a process of random generation, could check where (if) x occurs, various options are available,

In other cases this might be much more difficult to accomplish. Imagine for instance generating data to test the average case of the quicksort.

Of course, if you are using a pseudo-random number generator to generate (hopefully) random numbers for your input, then you do need to understand something about the pseudo-random number generator – periodicity, etc. There is a lot of literature in this regard.

2.3.8 Measure the performance on different data sets

This is possibly the most straightforward part of the experiment – run the program on the chosen data sets and measure the running times.

2.3.9 Interpret the results

In order to be able to make sense of the results/measurements/data that you have collected, you first need to decide on an appropriate way to present the data. You could use tables and graphs etc. The aim should be to have a form which helps you rather than obscures the data. In this case, graphs or tables of input size versus running time for the different cases would seem to be a sensible approach. Do not try to put too much information on any graph or in any table.

The next step would be to use some statistical techniques to start to make some sense of your data. The most obvious point to start in our example would be to do curve fitting to see if we can fit a curve to the growth rate of our algorithm. There is a lot of literature in this regard. The curves which we fit could then be shown as overlays on our raw data.

It is important to understand the statistical results of the curve fitting (or other techniques) which you have used. What does the regression data – constants for curve fitting, accuracy of regression, etc. – mean to your experiment?

You then need to work out what this data means in terms of analysis of algorithms

2.3.10 Verify best, average and worst case analysis

Once you have made sense of your raw data then you need to start relating this understanding of your empirical results to the theoretical results. In our example we expect the best case of the linear search to be O(1). Is the curve that we fitted for the measurements for the input data which we expected to give best case performance actually a constant? How much certainty can we say this with?

Similar approaches apply for the other cases.

2.3.11 Any other ideas

We could extend our experiment by looking at other aspects as well.

- looking at different types of data to verify that the growth rate is unaffected.
- looking at the space utilization of the algorithm
- looking at special cases of input data
- other things...

2.4 Document format – Lab Report

1. Aims
   Give details of what you are trying to do.
2. Summary of Theory
   Give *necessary* details and *refer* to the literature. Cite the literature that you refer to.
3. Experimental Methodology
   As detailed above. Motivate and explain why you plan to do what you plan to do.
4. Presentation of results
   Graphs (and tables), statistics.
5. Interpretation of results
   Discuss results of the experiment as is. What do the results tell you about the algorithm?
6. Relate results to theory
   Do the results confirm the theory? Explain (or explain why not).
7. Conclusion
   What did your experiment show? – based on discussion of results as above
8. References
9. Acknowledgments

Acknowledgments: This material was prepared by Prof. Ian Sanders.

References

[1] Sara Baase. *Computer Algorithms: Introduction to Design and Analysis (2Nd Ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988.

[2] Robert Sedgewick. *Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[3] Robert Sedgewick and Philippe Flajolet. *An Introduction to the Analysis of Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.