ACTIVITY NO. 14

ALGORITHM COMPLEXITY	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:**
Section:	**Date Submitted:**
Name:	**Instructor:** Engr. Roman M. Richard

1. Objective(s)

Analyze given algorithms to determine space and time complexity.

2. Intended Learning Outcomes (ILOs)

After this activity, the student should be able to:

- Compute for the space and time complexity of a given algorithm.

3. Discussion

Complexity of an algorithm is a measure of the amount of time and/or space required by an algorithm for an input of a given size (n).

What effects run time of an algorithm?

a) computer used, the hardware platform
b) representation of abstract data types (ADT's)
c) efficiency of compiler
d) competence of implementer (programming skills)
e) complexity of underlying algorithm
f) size of the input

We will show that of those above (e) and (f) are generally the most significant.

Time for an algorithm to run t(n)

A function of input. However, we will attempt to characterise this by the size of the input. We will try and estimate the WORST CASE, and sometimes the BEST CASE, and very rarely the AVERAGE CASE.

What do we measure?

In analyzing an algorithm, rather than a piece of code, we will try and predict the number of times "the principle activity" of that algorithm is performed. For example, if we are analysing a sorting algorithm we might count the number of comparisons performed, and if it is an algorithm to find some optimal solution, the number of times it evaluates a solution. If it is a graph colouring algorithm we might count the number of times we check that a coloured node is compatible with its neighbours.

- **Worst Case:** It is the maximum run time, over all inputs of size n, ignoring effects (a) through (d) above. That is, we only consider the "number of times the principle activity of that algorithm is performed".
- **Best Case:** In this case we look at specific instances of input of size n. For example, we might get best behaviour from a sorting algorithm if the input to it is already sorted.

- **Average Case:** Arguably, average case is the most useful measure. It might be the case that worst case behaviour is pathological and extremely rare, and that we are more concerned about how the algorithm runs in the general case. Unfortunately, this is typically a very difficult thing to measure.
    - Firstly, we must in some way be able to define by what we mean as the "average input of size n". We would need to know a great deal about the distribution of cases throughout all data sets of size n.
    - Alternatively, we might make a possibly dangerous assumption that all data sets of size n are equally likely.
    - Generally, to get a feel for the average case we must resort to an empirical study of the algorithm, and in some way classify the input (and it is only recently with the advent of high-performance, low-cost computation, that we can seriously consider this option).

The Growth rate of t(n)

Suppose the worst-case time for algorithm A is

t(n) = 60*n*n + 5*n + 1

for input of size n.

Assume we have differing machine and compiler combinations, then it is safe to say that

t(n) = n*n + 5*n/60 + 1/60

That is, we ignore the coefficient that is applied to the most significant (dominating) term in t(n). Consequently, this only affects the "units" in which we measure. It does not affect how the worst-case time grows with n (input size) but only the units in which we measure worst case time Under these assumptions we can say ...

"t(n) grows like n*n as n increases"

or

t(n) = O(n*n)

which reads "t(n) is of the order n squared" or as "t(n) is big-oh n squared." In summary, we are interested only in the dominant term, and we ignore coefficients.

Problem Complexity

- Assume we have a problem where we must consider all possible combinations. That is element e can be in or out of the set, and we have n elements in the set. If we had to find the "best" combination we might have to explore all alternatives in the worst case. There are 2**n such alternatives. Such a problem is likely to have an algorithm that is no better than O(2**n).

- Assume we have a problem where we must find the best permutation of n objects, ie given n objects sequence them in such a way that the sequence is "optimal" in some respect. There is n! possible different orderings, and if we had to examine all of these to find the best (in the worst case) the algorithm would be O(n!).
- **Problems of those kind are said to be INTRACTABLE**. Generally a problem is intractable if the best worst case algorithm is NOT polynomial (ie not quadratic, not cubic, not n raised to k). If an algorithm is polynomial it is said to be good, otherwise it is not good. There are a large (and growing) number of problems where there are no good algorithms, and we do not expect ever to find good algorithms for those problems ...but this has not yet been proven.

4. Materials and Equipment

Personal Computer with C++ IDE
Recommended IDE:

- CLion (must use TIP email to download)
- DevC++ (use the embarcadero fork or configure to C++17)

5. Procedure

Tabulated below, are a number of functions against n (from 1 to 10)

```
A = (log2 n) {log to base 2 of n}
B = n {linear in n}
C = (* n (log2 n)) {n log n}
D = (* n n) {quadratic in n}
E = (* n n n) {cubic in n}
F = (expt 2 n) {exponential in n}
G = (expt 3 n) {exponential in n}
H = (fact n) {factorial in n}
```

n	A	B	C	D	E	F	G	H
1	0.0	1	0.0	1	1	2	3	1
2	1.0	2	2.0	4	8	4	9	2
3	1.0	3	4.0	9	27	8	27	6
4	2.0	4	8.0	16	64	16	81	24
5	2.0	5	11.0	25	125	32	243	120
6	2.0	6	15.0	36	216	64	729	720
7	2.0	7	19.0	49	343	128	2187	5040
8	3.0	8	24.0	64	512	256	6561	40320
9	3.0	9	28.0	81	729	512	19683	362880
10	3.0	10	33.0	100	1000	1024	59049	3628800

Think of this as algorithms A through H with complexities as defined above, showing growth rate versus input size n. Tabulated below are functions F, G and H from above (ie 2 to power n, 3 to power n, and n factorial). Problem size n varies from 10 to 100 in steps of 10. I have assumed that we have a machine that can perform "the principle activity of the algorithm" in a micro second (ie. if we are considering a graph colouring algorithm, it can compare the colour of two nodes in a millionth of a second). The columns give the number of years this machine would take to execute those algorithms on problems of size n (note: YEARS). This is expressed as 10^x, "10 raised to the power x".

```
  n       F          G          H
 10     10^-10     10^-8      10^-6
 20     10^-7      10^-3      10^4
 30     10^-4      10^0       10^18
 40     10^-1      10^5       10^34
 50     10^1       10^10      10^50
 60     10^4       10^15      10^68
 70     10^7       10^19      10^86
 80     10^10      10^24      10^105
 90     10^13      10^29      10^124
100     10^16      10^34      10^144
```

Therefore, if we have a problem of size (lets say 40) and the machine specified above, if the best algorithm is O(2**n) it will take 1 year, if the best algorithm is O(3**n) it will take 100,000 years, and if the best algorithm is O(n!) it will take

10,000,000,000,000,000,000,000,000,000,000,000 years

approximately. Out of interest, the age of the universe is estimated to be between 15 and 20 billion years old, ie 20,000,000,000 years. That is, even at modest values of n we are presented with problems that will never be solved. It is almost tempting to say, that from a computational perspective, the universe is a small thing.

Contrast and compare two algorithms used in sorting. For the two algorithms, identify the best and worst case. Additionally, provide an analysis of the algorithms if they were to sort data with the following values of n: 100, 1,000, and 10,000 (use experimental and theoretical analysis tools used in previous activities).

6. Output

	Best Case	Worst Case	Analysis
Algorithm 1			
Algorithm 2			

Table 14-1. Best- and Worst-Case Analysis using Theoretical Tools

	Algorithm 1	Algorithm 2
Best Case		
Worst Case		

Table 14-2. Best- and Worst-Case Analysis using Experimental Tools

7. Supplementary Activity

ILO A: Contrast and compare the two searching techniques used in class. The required deliverable for this activity would be the theoretical and empirical analysis.

For the theoretical analysis, the deliverable must be:

- Pseudocode Analysis
- Time and Space Complexities

For the empirical analysis, the deliverable must be the filled-out table and your analysis.

Input Size (N)	Elapsed Time for Binary Search (Worst-Case)	Elapsed Time for Linear Search (Worst-Case)
1,000		
10,000		
100,000		

Provide your analysis.

8. Conclusion

Provide the following:

- Summary of lessons learned
- Analysis of the procedure
- Analysis of the supplementary activity
- Concluding statement / Feedback: How well did you think you did in this activity? What are your areas for improvement?

9. Assessment Rubric

References

http://www.dcs.gla.ac.uk/~pat/52233/complexity.html
https://viterbi-web.usc.edu/~adamchik/15-121/lectures/Algorithmic%20Complexity/complexity.html#:~:text=Algorithmic%20complexity%20is%20concerned%20about,depending%20on%20the%20implementation%20details