The Analysis Framework

Analyzing the Algorithm

In analyzing an algorithm, one of the qualities that should be considered is **efficiency**.

- ***Space Efficiency or Complexity*** – It is the amount of computer space or memory required by an algorithm (including the input values of the algorithm) to complete its execution and produce the result. The memory space we consider is the space of primary memory.

  Types of Space Complexity
  - Constant Space Complexity
  - Linear Space Complexity

  Components of Space/Memory Use:

	Description
Instruction space	It is the amount of memory used to store a compiled version of instructions.
Data space	It is the amount of memory used to store all the variables and constants.
Run-time Stack Space/Environmental Stack	It is the amount of memory used to store information of partially executed functions at the time of function call.

  Memory Required to Store Different Data Type Values:

	Data Types	Storage size	Value range
Integer Types	char	1 byte	-128 to 127 or 0 to 255
	unsigned char	1 byte	0 to 255
	signed char	1 byte	-128 to 127
	int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
	unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
	short	2 bytes	-32,768 to 32,767
	unsigned short	2 bytes	0 to 65,535
	long	4 bytes	-2,147,483,648 to 2,147,483,647
	unsigned long	4 bytes	0 to 4,294,967,295
Floating-Point Types	float	4 byte	1.2E-38 to 3.4E+38
	double	8 byte	2.3E-308 to 1.7E+308
	long double	10 byte	3.4E-4932 to 1.1E+4932

- ***Time Efficiency or Complexity*.**

  - T(n) – It is the amount of computer time required by each operation to execute.

    The actual running time of an algorithm depends on many factors:

1. **Computer's speed** (Whether it is running on a single processor machine or multi-processor machine, whether it is a 32bit machine or 64bit machine, read and write speed of the machine.)

   An algorithm may run faster in one machine but not to another machine because it depends on the computer's speed. Limited computer memory affects the efficiency of the algorithm. If you want your computer to run faster in nearly all cases, consider upgrading the CPU.

2. **Quality of the program implementing the algorithm** (Input data)

   Running complex programs is difficult as compared to running simple programs. Usually, it depends on the input size like sorting algorithm that does better on small lists but slower on longer lists.

3. **Compiler used in generating the machine code** (The amount of time required by an algorithm to perform Arithmetic operations, logical operations, return value and assignment operations, etc.)

   The use of compiler in generating the machine code affects the efficiency of the algorithm because once in the process of compilation, the compiler effectively controls the final configuration of the hardware and how it operates.

- C_{op} – It is the amount of computer time required for a single operation in each line.
  - Comments = 0 step
  - Assignment statement which does not involve any calls to other algorithms = 1 step
  - Condition statement = 1 step

- C(n) – It is the amount of computer time required by each operation for all its repetitions.
  - Loop condition for n times = n + 1 steps
  - Body of loop = n steps

Types of Time Complexity

- Constant Time Complexity
- Linear Time Complexity

Asymptotic Notations and Basic Efficiency Classes

Algorithm's Growth Rate (aka Asymptotic Notations)

- These are a set of languages that allow us to analyze an algorithm's running time for asymptotic analysis by identifying its behavior as the input size for the algorithm increases.

- Classification of Increase (Growth)
  - *Increase with the same rate* – Does it mostly maintain its quick run time as the input size increases?
  - *Increase with a slower rate* – Does the algorithm suddenly become incredibly slow when the input size increases?
  - *Increase with a faster rate*

Three (3) Types of Asymptotic Notations

1. *Big-0 Notation (0)* (with a capital letter O, not a zero) (aka Landau's symbol)

- It comes from the name of the German number theoretician Edmund Landau who invented the notation.
- Unlike in Big Ω (omega) and Big θ (theta), the 'O' in Big O is not Greek. It stands for the rate of growth of a function is also called its order.
- This notation is used to describe the asymptotic upper-bound on growth rate or the worst case of an algorithm in terms of time complexity by taking the highest order of a polynomial function and ignoring all the constants value since they are not that influential for sufficiently large input.
- In general, a function t(n) or f(n) is O(g(n)) if there exist positive constants c and n_0 such that t(n) or f(n) $\leq c * g(n)$ for all $n \geq n_0$.
- Formally, O(g(n)) = {f(n) : $\exists$ positive constants c and n_0, such that $\forall n \geq n_0$, we have $0 \leq f(n) \leq cg(n)$}.
- Intuitively, a set of all functions whose rate of growth is the same as or lower than that of g(n).

  Where
  - "f is big-O of g"
  - "g asymptotically dominates f"

Problem 1:	***Problem 2:***	***Problem 3:***
• _____ _____ _____	• _____ _____ _____	• _____ _____ _____
Solution:	***Solution:***	***Solution:***

- The Big-O notation can be derived from t(n) or f(n) using the following steps:
  1. In each term, set the coefficient of the term to 1. In short, if the term has a constant, that constant will be changed to a value of 1.
  2. Keep the term in the function with the highest exponent and discard the least significant terms
  3. Terms are ranked from lowest to highest as shown:

Efficiency	Big-O	Iterations	Estimated time	Description	Example
Constant	O(1)			The operation takes the same amount of time, no matter how much data we are dealing with.	5, 25, 6000

Logarithmic	O(log n)			The time taken by the operation increases with the size of the data set. In short, the algorithm takes a longer time per item on smaller datasets relative to larger ones. A perfect example of an algorithm that utilizes this approach is the binary search or the merge sort algorithm.	$\log_5 n$, log n
Linear	O(n)			The operation is performed n amount of times. The most common and simplest example is the classic "for loop," in which particular operation is repeated n amount of times.	5n, 25n, 6000n
Linear logarithmic	$O(n(\log_2 n))$				$n \log_5 n$, n log n
Quadratic	$O(n^2)$			The operation is performed n * n times. To execute a program n * n times, we need to nest a "for loop." Examples of algorithms taking quadratic time include the following algorithms: • Bubble sort • Selection sort • Insertion sort	
Polynomial	$O(n^k)$				$5n^2$, $25n^4$, $6000n^{12}$
Exponential	$O(c^n)$			The time taken by the operation gets n times bigger with every additional input.	5^n, 25^{6000n}
Factorial	O(n!)				

A problem that can be solved with polynomial worst-case complexity is called **tractable**. Problems of higher complexity are called **intractable**.

Problem 1: • _____ _____ _____	***Problem 2:*** • _____ _____ _____	***Problem 3:*** • _____ _____ _____
Solution: • Step 1: • Step 2: • Step 3:	***Solution:*** • Step 1: • Step 2: • Step 3:	***Solution:*** • Step 1: • Step 2: • Step 3:

2. *Big-0mega Notation (Ω)*
   - This notation is used to describe the asymptotic lower-bound on growth rate or the best case of an algorithm in terms of time complexity. Its only difference from the Big O definition is the omega symbol and the greater than or equal to a symbol (which was less than or equal to in Big O).
   - In general, a function t(n) or f(n) is $\Omega(g(n))$ if there exists positive constants c and n_0 such that t(n) or f(n) $\geq$ c * g(n) for some n $\geq n_0$.
   - Formally, $\Omega(g(n))$ = {f(n) : $\exists$ positive constants c and n_0, such that $\forall n \geq n_0$, we have $0 \leq cg(n) \leq f(n)$}.
   - Intuitively, a set of all functions whose rate of growth is the same as or higher than that of g(n).

     Where
     - o "f is big-Omega of g"

Problem 1: • _____ _____ _____	***Problem 2:*** • _____ _____ _____	***Problem 3:*** • _____ _____ _____
Solution:	***Solution:***	***Solution:***

3. *Big-Theta Notation (Θ)*
   - This notation is used to describe asymptotic tight bound for it bounds a function from above and below.
   - In general, a function t(n) or f(n) is Θ(g(n)) if there exists a constant c_1, c_2, $n_0 > 0$; such that for every integer $n \geq n_0$ we have $c_1 * g(n) \leq t(n)$ or $f(n) \leq c_2 * g(n)$.
   - Formally, Θ(g(n)) = {f(n) : ∃ positive constants c_1, c_2, and n_0, such that $\forall n \geq n_0$, $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$}.
   - Intuitively, set of all functions that have the same rate of growth as g(n).

     Where
     - "f is big-Theta of g"
     - "f is of order g"
     - "f and g are of the same order"

Problem 1: • _____ _____ _____ ***Solution:***	***Problem 2:*** • _____ _____ _____ ***Solution:***	***Problem 3:*** • _____ _____ _____ ***Solution:***

References

Algorithms I : Searching and Sorting algorithms. (2018). Retrieved from: https://codeburst.io/algorithms-i-searching-and-sorting-algorithms-56497dbaef20
Algorithm Efficiency. (n. d.). Retrieved from: http://www.cs.kent.edu/~durand/CS2/Notes/03_Algs/ds_alg_efficiency.html
Algorithm Efficiency. (n. d.). Retrieved from: http://www2.cs.uidaho.edu/~rinker/cs113/BigO.pdf
Asymptotic Notations. (2019). Retrieved from: https://learnxinyminutes.com/docs/asymptotic-notation/
Big-O Notation Explained with Examples. (2019). Retrieved from: https://developerinsider.co/big-o-notation-explained-with-examples/
Big O Notation Simplified to the Max. (2018). Retrieved from: https://www.thecodingdelight.com/big-o-time-complexity-simplified/
C - Data Types. (n. d.). Retrieved from: https://www.tutorialspoint.com/cprogramming/c_data_types.htm
Data Structures - Algorithms Basics. (2019). Retrieved from: https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
Efficiency Definition. (2019). Retrieved from: https://www.investopedia.com/terms/e/efficiency.asp
Efficiency and Complexity. (2005). Retrieved from: https://users.cs.jmu.edu/bernstdh/Web/common/lectures/algorithms_efficiency.php
Extension: Intro to Asymptotic Notation. (2019). Retrieved from: https://www.expii.com/t/extension-intro-to-asymptotic-notation-112
Space Complexity. (2019). Retrieved from: http://btechsmartclass.com/data_structures/space-complexity.html
Space Complexity of Algorithms. (2019). Retrieved from: https://www.studytonight.com/data-structures/space-complexity-of-algorithms
Theoretical Review. (n. d.). Retrieved from: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=73929
Time and Space Analysis of Algorithm. (n. d.). Retrieved from: https://www.includehelp.com/data-structure-tutorial/time-and-space-analysis-of-algorithm.aspx
Time and Space Complexity. (2019). Retrieved from: https://www.hackerearth.com/practice/basic-programming/complexity-analysis/time-and-space-complexity/practice-problems/algorithm/vowel-game-f1a1047c/
What is 'Space Complexity'?. (2019). Retrieved from: https://www.tutorialspoint.com/What-is-Space-Complexity