**DATA STRUCTURES AND ALGORITHMS**

Why is DSA important?

A: Make you a better Software Developer, help you in getting a job, winning the sport of competitive coding.

**Analysis of Algorithm**:

Algorithm analysis is an important part of computational complexity theory, which provides theoretical estimation for the required resources of an algorithm to solve a specific computational problem. Analysis of algorithms is the determination of the amount of time and space resources required to execute it.

Types of Algorithm Analysis:

1. Best Case

2. Worst Case  
3. Average Case

Asymptotic Analysis:

* The idea is to measure the order of growth.
* Does not depend upon machines, programming language etc.
* No need to implement the code, we can directly analyze it directly.

[*Asymptotic Analysis*](http://en.wikipedia.org/wiki/Asymptotic_analysis)*is the big idea that handles the above issues in analyzing algorithms. In Asymptotic Analysis, we****evaluate the performance of an algorithm in terms of input size****(we don’t measure the actual running time). We calculate, how the time (or space) taken by an algorithm increases with the input size.*

**For example**, let us consider the search problem (searching a given item) in a sorted array.

The solution to above search problem includes:

* **Linear Search** (order of growth is linear)
* **Binary Search** (order of growth is logarithmic).

To understand how Asymptotic Analysis solves the problems mentioned above in analyzing algorithms,

* let us say:
  + we run the Linear Search on a fast computer A and
  + Binary Search on a slow computer B and
  + pick the constant values for the two computers so that it tells us exactly how long it takes for the given machine to perform the search in seconds.
* Let’s say the constant for A is 0.2 and the constant for B is 1000 which means that A is 5000 times more powerful than B.
* For small values of input array size n, the fast computer may take less time.
* **But, after a certain value of input array size, the Binary Search will definitely start taking less time compared to the Linear Search even though the Binary Search is being run on a slow machine**.

| **Input Size** | **Running time on A** | **Running time on B** |
| --- | --- | --- |
| **10** | 2 sec | ~ 1 h |
| **100** | 20 sec | ~ 1.8 h |
| **10^6** | ~ 55.5 h | ~ 5.5 h |
| **10^9** | ~ 6.3 years | ~ 8.3 h |

* The reason is the order of growth of Binary Search with respect to input size is logarithmic while the order of growth of Linear Search is linear.
* **So the machine-dependent constants can always be ignored after a certain value of input size.**

Running times for this example:

* Linear Search running time in seconds on A: 0.2 \* n
* Binary Search running time in seconds on B: 1000\*log(n)

**Does Asymptotic Analysis always work?**

Asymptotic Analysis is not perfect, but that’s the best way available for analyzing algorithms. For example, say there are two sorting algorithms that take 1000nLogn and 2nLogn time respectively on a machine. Both of these algorithms are asymptotically the same (order of growth is nLogn). So, With Asymptotic Analysis, we can’t judge which one is better as we ignore constants in Asymptotic Analysis.

Also, in Asymptotic analysis, we always talk about input sizes larger than a constant value. It might be possible that those large inputs are never given to your software and an asymptotically slower algorithm always performs better for your particular situation. So, you may end up choosing an algorithm that is Asymptotically slower but faster for your software.

Order of Growth:

A function f(n) is said to be growing faster that g(n) if

lim g(n)/f(n) = 0 or lim f(n)/g(n) = infinity

* + F(n), g(n) represent time taken and n,f(n),g(n) >=0

Direct way to find and compare growths:

1. Ignore lower-order terms
2. Ignore leading terms constant

Constant(C) < loglogn < logn < n^1/3 < n^1/2 < n < n^2 < n^3 < n^4< 2^n < n^n

Worst, Average and Best Case Time Complexities:

**Worst Case Analysis** **(Usually Done):** In the worst case analysis, we calculate upper bound on running time of an algorithm. We must know the case that causes the maximum number of operations to be executed. For Linear Search, the worst case happens when the element to be searched (x in the above code) is not present in the array. When x is not present, the search() functions compares it with all the elements of arr[] one by one. Therefore, the worst case time complexity of linear search would be  O(N), where N is the number of elements in the array.

**Average Case Analysis** **(Sometimes done):**  In average case analysis, we take all possible inputs and calculate computing time for all of the inputs. Sum all the calculated values and divide the sum by total number of inputs. We must know (or predict) distribution of cases. For the linear search problem, let us assume that all cases are uniformly distributed (including the case of x not being present in array). So we sum all the cases and divide the sum by (N+1). Following is the value of average case time complexity.

A math equation with black text

Description automatically generated

**Best Case Analysis (Bogus):** In the best case analysis, we calculate lower bound on running time of an algorithm. We must know the case that causes minimum number of operations to be executed. In the linear search problem, the best case occurs when x is present at the first location. The number of operations in the best case is constant (not dependent on N). So time complexity in the best case would be O(1).

**Time Complexity Analysis: (In Big-O notation)**

* **Best Case:**O(1),This will take place if the element to be searched is on the first index of the given list. So, the number of comparisons, in this case, is 1.
* **Average Case:**O(n), This will take place if the element to be searched is on the middle index of the given list.
* **Worst Case:**O(n), This will take place if:
  + The element to be searched is on the last index
  + The element to be searched is not present on the list

**Important Points:**

* Most of the times, we do the worst case analysis to analyze algorithms. In the worst analysis, we guarantee an upper bound on the running time of an algorithm which is a good piece of information.
* The average case analysis is not easy to do in most of the practical cases and it is rarely done. In the average case analysis, we must know (or predict) the mathematical distribution of all possible inputs.
* The Best Case analysis is bogus. Guaranteeing a lower bound on an algorithm doesn't provide any information as in the worst case, an algorithm may take years to run.