Big o Notation

Assembly Line O(n)

Matrix Multiplication O(n3)

1. There are *two aspects* of algorithmic performance

Time

Space

1. ***Analysis of Algorithms*** is the area of computer science that provides tools to analyze the efficiency of different methods of solutions.
2. How do we compare the time efficiency of two algorithms that solve the same problem?
3. ***Naïve Approach***: implement these algorithms in a programming language (C++), and run them to compare their time requirements. Comparing the programs (instead of algorithms) has difficulties.
4. while (i <= n) n+1
5. An algorithm’s proportional time requirement is known as ***growth rate***.
6. We can compare the efficiency of two algorithms by comparing their growth rates.

|  |  |
| --- | --- |
| 1. Function | 1. Growth Rate Name |
| 1. *c* | 1. Constant |
| 1. *log N* | 1. Logarithmic |
| 1. *log2N* | 1. Log-squared |
| 1. *N* | 1. Linear |
| 1. *N log N* |  |
| 1. *N2* | 1. Quadratic |
| 1. *N3* | 1. Cubic |
| 1. *2N* | 1. Exponential |

1. O notation :Big-O is the formal method of expressing the upper bound of an algorithm's running time.
2. ***Big-Omega Notation*** *Ω*
3. This is almost the same definition as Big Oh, except that "*f(n)* ≥ *cg(n)*”
4. This makes *g(n)* a lower bound function, instead of an upper bound function.
5. It describes the **best that can happen** for a given data size.
6. **Theta Notation** *Θ*
7. Theta Notation For non-negative functions, *f(n)* and *g(n)*, *f(n)* is theta of *g(n)* if and only if *f(n)* = *O(g(n))* and *f(n)* = *Ω(g(n))*. This is denoted as "*f(n)* = *Θ(g(n))*".
8. This is basically saying that the function, *f(n)* is bounded both from the top and bottom by the same function, *g(n)*.
9. The **function f(n)** is called the algorithm’s **growth-rate function**.
10. ***Worst-Case Analysis*** –The maximum amount of time that an algorithm require to solve a problem of size n.
11. This gives an upper bound for the time complexity of an algorithm.
12. Normally, we try to find worst-case behavior of an algorithm.
13. ***Best-Case Analysis*** –The minimum amount of time that an algorithm require to solve a problem of size n.
14. The best case behavior of an algorithm is NOT so useful.
15. ***Average-Case Analysis*** –The average amount of time that an algorithm require to solve a problem of size n.
16. Sometimes, it is difficult to find the average-case behavior of an algorithm.
17. We have to look at all possible data organizations of a given size n, and their distribution probabilities of these organizations.
18. ***Worst-case analysis is more common than average-case analysis.***

Backtracking:

* Backtracking is a methodical way of trying out various sequences of decisions, until you find one that “works”
* Backtracking is a modified depth-first search of a tree.
* Backtracking involves only a tree search.
* We call a node nonpromising if when visiting the node we determine that it cannot possibly lead to a solution. Otherwise, we call it promising.
* This is called pruning the state space tree, and the subtree consisting of the visited nodes is called the pruned state space tree.
* Brute force algorithms are slow
  + backtracking is a form of a brute force algorithm

Dynamic Programming:

* Divide and conquer – Partition the problem into independent subproblems – Solve the subproblems recursively – Combine the solutions to solve the original problemDynamic Programming • Used for optimization problemsz
* Dynamic Programming • Used for optimization problems

Greedy Algorithm:

* Optimization Problem

• Problem with an objective function to either:

• Maximize some profit • Minimize some cost

Optimization problems appear in so many applications

• Maximize the number of jobs using a resource [Activity-Selection Problem]

• Encode the data in a file to minimize its size [Huffman Encoding Problem]

• Collect the maximum value of goods that fit in a given bucket [knapsack Problem]

• Select the smallest-weight of edges to connect all nodes in a graph [Minimum Spanning Tree]

• Two techniques for solving optimization problems:

• Greedy Algorithms (“Greedy Strategy”)

• Dynamic Programming

• Dynamic programming can be overkill (slow)

; greedy algorithms tend to be easier to code

• Activity-Selection Problem • Huffman Codesf

If the items are already sorted into decreasing order of vi / wi, then the while-loop takes a time in O(n); Therefore, the total time including the sort is in O(n log n).