Algorithms and Data Structures

Algorithm

- sequence of computational steps which transforms a set of values (input) to another set of values (output)

Data Structure

- a way to store and organize data in order to facilitate acces and modification

1. Insertion sort - SORTING ALGORITHM

Insertion sort animation

```
Insertion-Sort (A,n)
  for j=2 to n
    key = A[j]
    i=j-1
    while i>0 and A[i] > key
        A[i+1]=A[i]
        i=i-1
    A[i+1]=key
```

Intuitive explanation: Algorithm goes through every element of the aray from the position 2 (j). If the current element is smaller/bigger, the algorithm inserts it on the "left" such that the elements in range (1,j) are sorted.

Correctness

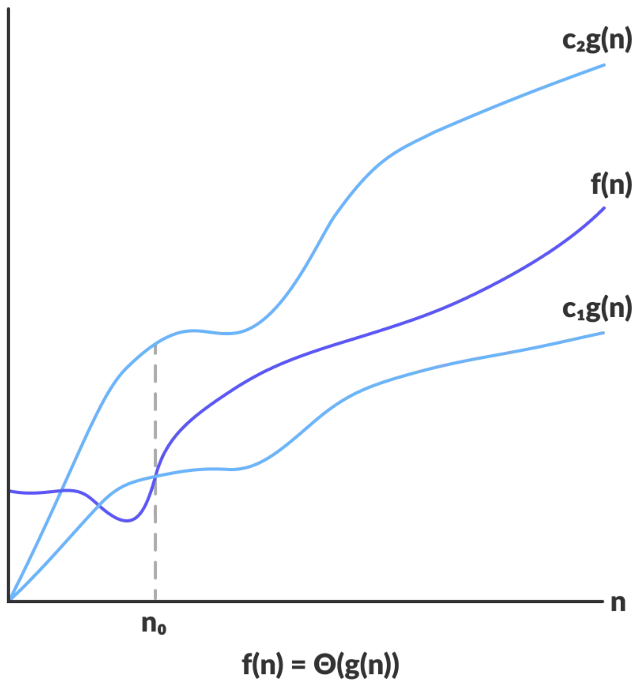
- prove it with the help of loop invariants
- loop invariant = a property of a program loop that is true before and after each iteration
- if you want to prove the correctness:
  - find loop invariant in the program
  - show that when the program ends the loop invariant will still hold true

Time of the execution - Asymptotic Analysis

- **Intuitive Idea**: look at growth of T(n) as n approaches infinity

  Asymptotically Tight Bound: Θ-Notation

  $\Theta(g(n)) = f(n) | \exists c_1, c_2 > 0\ and\ n_0\ such\ that\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0 \quad \theta$ - represents the zone in which our function will "perform" -> includes the worst and best case

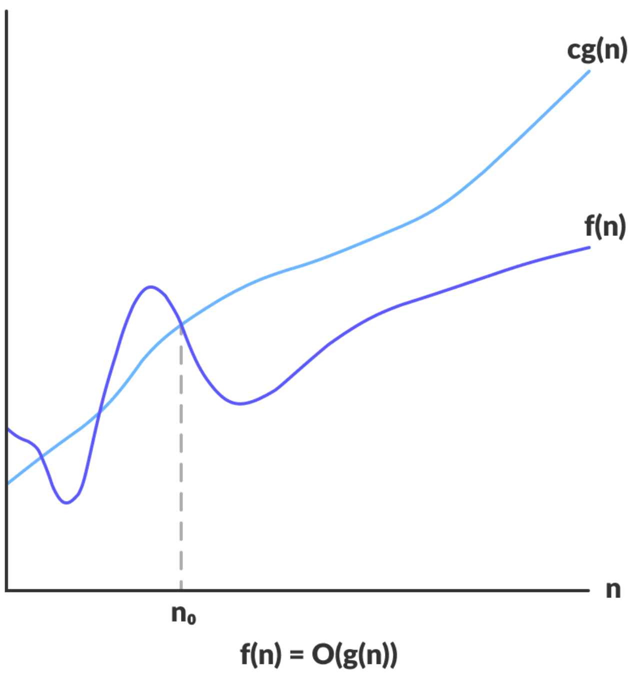


Informal: to compute the θ take the term to the highest power and ignor the coefficients

- Ex: $3n^3 + 90n^2 - 5n + 6064 = \theta(n^3)$

Asymptotically Upper Bond: O-Notation

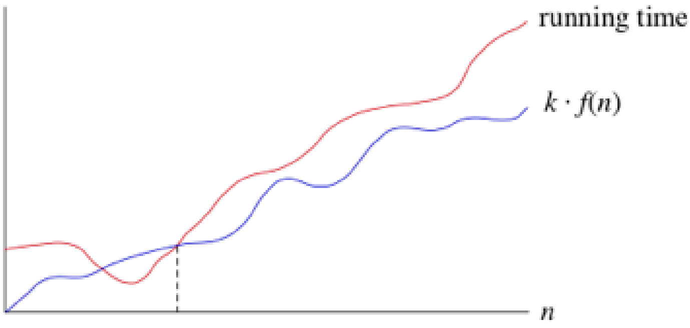
$O(g(n)) = f(n) | \exists c \; and \; n > 0, such \; that \; 0 \leq f(n) \leq cg(n), \forall n \geq n_0$ O - represents the upper bound => the worst case scenario O have to always be the **tightest** function we can get



Asymptotically Lower Bond: Ω-Notation

$\Omega(g(n)) = f(n) | \exists c > 0 \; and \; n_0 \; such \; that \; 0 \leq cg(n) \leq f(n), \forall n \geq n_0$ Ω - represents the lower bond => tells you can't have a better algorithm than Ω (best case scenario) For tight bonds, we get:

$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$



Non-tight Upper Bound: o -Notation

$o(g(n)) = f(n) | \exists c > 0 \; and \; n > 0, such \; that \; 0 \leq f(n) < cg(n), \forall n \geq n_0$ o represents a bound that is non-tight (the function never hits the upper bound)

Non-tight Lower Bound: ω - Notation

For a given asymptotically non-negative functon g(n), we define $f(n) \in \omega(g(n)) \; iff \; g(n) \in o(f(n))$

Asymptotic Analysis: Computation with Limits:

$f \in o(g)$	$lim_{n->\infty} \frac{f(n)}{g(n)} = 0$
$f \in O(g)$	$lim_{n->\infty} \frac{f(n)}{g(n)} < \infty$
$f \in \omega(g)$	$lim_{n->\infty} \frac{f(n)}{g(n)} = \infty$
$f \in \Omega(g)$	$lim_{n->\infty} \frac{f(n)}{g(n)} > 0$
$f \in \Theta(g)$	$0 < lim_{n->\infty} \frac{f(n)}{g(n)} < \infty$

Heap as a Data Structure

Max Priority Queues

- **Priority Queue**: data structure for maintaining a set S of elements, each with an associated value called a key
- **Max-priority Queue**: priority queue with the following operations
  - *Maximum(S)*: return element from S with largest key
  - *Extract-Max(S)*: remove and return element from S with largest key
  - *Increase-Key*(s,x,k): increase the value of the key of element x to k, where k is assumed to be larger or equal than the current key.
  - *Insert*: add element x to set S