# Algorítimos

## Problema

Um problema computacional é um conjunto de inputs e outputs que devem ser relacionados por um algorítimo de modo que para cada input um ou mais outputs corretos sejam encontrados.

Dizer qual a relação entre input x output um a um seria difícil, então normalmente o problema é especificado por um predicado genérico, verificável e com inputs de tamanho arbitrário que deve ser usado para encontrar o output correto (ex: Encontre quantas pessoas são mais velhas que a idade X em um conjunto).

## Algorítimo

Um algorítimo é basicamente uma função (matemática) que recebe um input e mapeia ele para N inputs baseado em um conjunto de passos. No caso do exemplo “ Encontre quantas pessoas são mais velhas que a idade X em um conjunto” os passos são:

1. Perguntar a idade de todas as pessoas do conjunto
2. Contar quantas pessoas têm idade maior que X
3. Retornar o resultado

### Correctness

Consiste em provar que um algorítimo funciona, quando o problema computacional é simples e possui um input pequeno então basta analisar caso a caso e verificar, quando o input é grande e varia então deve ser usada indução matemática.

Indução matemática é um método usado para provar que uma teoria matemática é verdadeira para qualquer número natural quando: a teoria é verdadeira para X = 0 e para qualquer valor X = X + 1 (qualquer número natural).

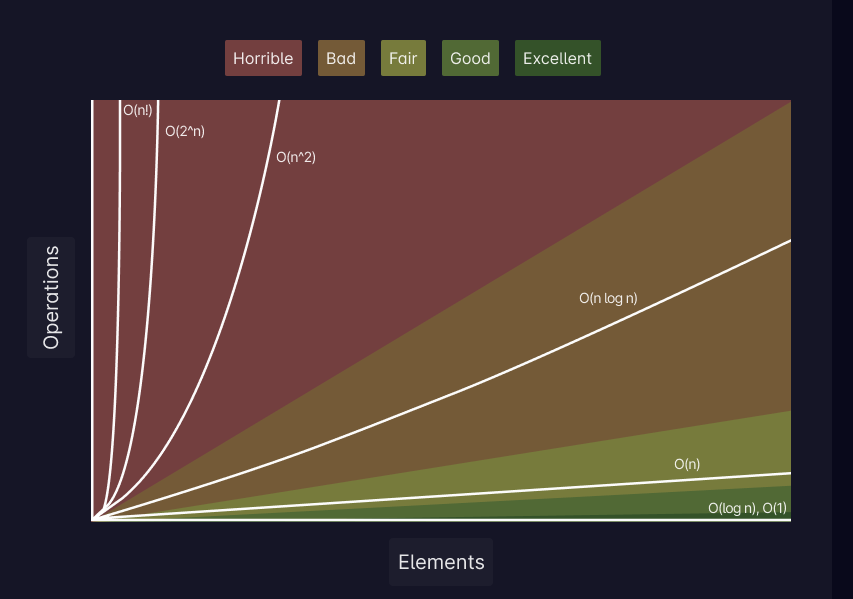
A ideia é usar inputs de teste e verificar se o algorítimo funciona para eles e caso o algorítimo seja funcional para um caso assume-se que ele também será funcional para os outros.

### Eficiência

Usado para medir quão rápido um algorítimo roda, não é medido em tempo de execução (varia com a máquina) e sim em operações fundamentais necessárias para resolver o problema, de modo que a performance (quantidade) varia com o tamanho do input (quanto maior input mais operações necessárias).

A variação que ocorre entre a quantidade de inputs x quantidade de operações necessárias para resolver o problema, do mais eficiente ao menos eficiente, são representadas por big O notation (onde n é o número de inputs):

* O^1: A quantidade de inputs não afeta o número de operações necessárias
* O(log n): Quase tão bom quanto linear, a quantidade de operações sobe um pouco no início, porém conforme N aumenta a variação é quase nenhuma
* O(n): A quantidade de operações aumenta 1:1 com a quantidade de inputs
* O (n log n)
* O(n^2): Complexidade de um for com um for dentro, sem levar em conta o código que é executado nesse for
* O(2^n): Exponencial
* O(n!): Fatorial, uma função exponencial crescerá mais



# Data Structures

## Data structure vs Interface (API/ADT)

* Interface: Define as operações suportadas por uma estrutura de dados
* Estrutura de dados: Uma forma de se guardar dados que implementa uma interface

## Arrays e Sequences

### Static Array

É uma estrutura de dados que representa uma lista de dados, os dados inseridos mantém a ordem de inserção e suas operações para tamanho(length), get e set são O^1 uma vez que a criação do array precisa alocar N espaços de memória sua criação é O(n). A memória alocada é sempre fixa. A eficiência das operações é:

* Get(index) e Set(index): O^1. Set sempre sobrescreve um elemento
* Criação de array: O(n)
* Size/Length/len: O^1

### Dynamic Array

É o mesmo de static array, mas quando o array atinge sua capacidade ocorre um resize, que consiste em criar um array e copiar os itens automaticamente. Resize sempre é O(n), o tamanho do array normalmente é multiplicado por progressão geométrica de 2 (1, 2, 4, 8, 16…).

Remover um elemento no final de um dynamic array é O(1) já que basta reduzir o tamanho do array em 1, sem a necessidade de percorre-lo.

### Linked List

* Nó: estrutura com referência para o próximo dado e para seu próprio dado

Estrutura que usa um array de nós para formar uma sequência ordenada de elementos. Possui uma referência para o primeiro elemento (head) e para o último (tail).

* Ler, remover e inserir um elemento inicial ou final será O^1, enquanto operações nos elementos intermediários serão O(n).

### Armortização

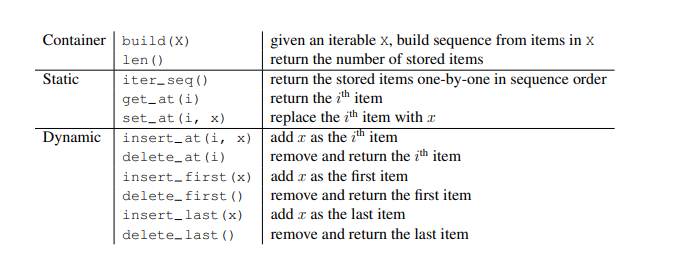
Técnica de análise avaliar a eficiência média de um algorítimo que possui operações lentas que são usadas ocasionalmente (resize e operações O(n) de array) e operações rápidas que são usadas com mais frequência (set, get, etc em um array).

A ideia é analisar a performance de um algorítimo em múltiplas iterações e se obter uma media em vez de se usar o pior caso possível.

Um exemplo disso é a inserção em um array dinâmico, inserir em um array é O(n) já que no pior caso será necessário aumentar o tamanho do array, mas considerando que nem sempre isso será necessário se pode dizer que a operação leva O(1) tempo amortizado.

### Sequence

Interface que representa uma sequência de dados, ordenados de maneira extrínseca (normalmente por inserção). Pode ser static quando possui um tamanho fixo, definido durante a criação, ou dynamic quando possui um tamanho variável. Toda sequence tem operações de:



#### Static Array Sequence

Estrutura de dados que usa um static array para armazenar os elementos da sequence. Operações static (getAt e setAt) são sempre O(1), porém operações dynamic são sempre O(n), especialmente operações de insert/remove já que é necessário realocar/recriar o array toda vez.

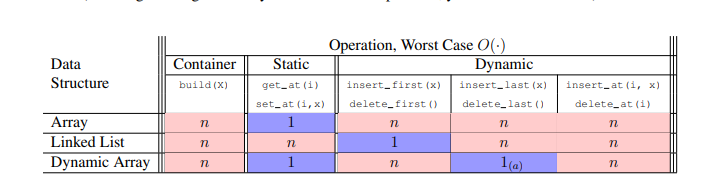
#### Linked List Sequence

Usa uma linkedlist para representar a sequence, de modo que fazer operações em uma posição basta alterar seu apontamento do node.

Operações com tail/head são sempre O^1, já que existe uma referência para eles. Qualquer outra operação é O(n), de modo que é uma estrutura recomendada quando se quer utilizar os elementos da ponta.

#### Dynamic Array sequence

É o mesmo caso de static array sequence, porém seu diferencial é que é possível remover elementos no final com complexidade O(1).



### Sets e Sorting

Um set é similar um sequence, porém os dados são organizados de maneira intrínseca, de modo que são ordenados por uma lógica específica, de acordo com sua classificação, chamada de key.

