# Relatório 1º Projeto ASA 2024/2025

**Grupo:** AL063

**Alunos:** Madalena Mota (110355) e Ricardo Fonseca (109834)

---

## Descrição do Problema e da Solução

A solução proposta adota uma abordagem dinâmica eficiente. O algoritmo consiste em dividir a sequência em segmentos e testar o resultado da operação entre o segmento esquerdo e o direito de forma dinâmica. Para reconstruir a expressão, foi implementado um algoritmo recursivo em que retornamos à posição em que dividimos a sequência e procuramos os valores pretendidos no segmento esquerdo e direito. Por fim, reaplicamos o algoritmo a ambos os segmentos.

## Preenchimento da Tabela

Cada célula da tabela contém um vetor de estruturas com todos os valores que é possível obter à esquerda (l) e direita (r) dos parêntesis aplicados na posição onde dividimos a sequência (k), e o resultado da operação entre l e r (res). Um exemplo de preenchimento da tabela seria (n = 3, m = 3, X = 3 3 1):

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 3 | (left: 3 right: 3 k: 1 res: 1) | (left: 1 right: 1 k: 2 res: 3) |
| 1 | | 3 | (left: 3 right: 1 k: 2 res: 1) |
| 2 | | | 1 |

## Análise Teórica

- Leitura do input: $O(m^2n)$ - recebemos uma sequência de comprimento **n** e uma tabela **m×m** com o resultado da operação entre cada um dos números;
- Preenchimento da tabela: $O(m^3n^2)$ - a tabela tem dimensões **m×m** e o custo de preencher cada célula é $O(mn^2)$: temos **m-1** posições possíveis no pior caso para escolher k e depois testamos cada combinação de valores da esquerda e da direita (no máximo **n** valores em ambos);
- Reconstrução da string: $O(nm)$ - temos **m-1** possibilidades para dividir a sequência, pelo que fazemos **m-1** chamadas recursivas. Cada chamada tem custo $O(n)$, pois percorremos todos os valores da posição da tabela correspondente (no máximo **n** valores).

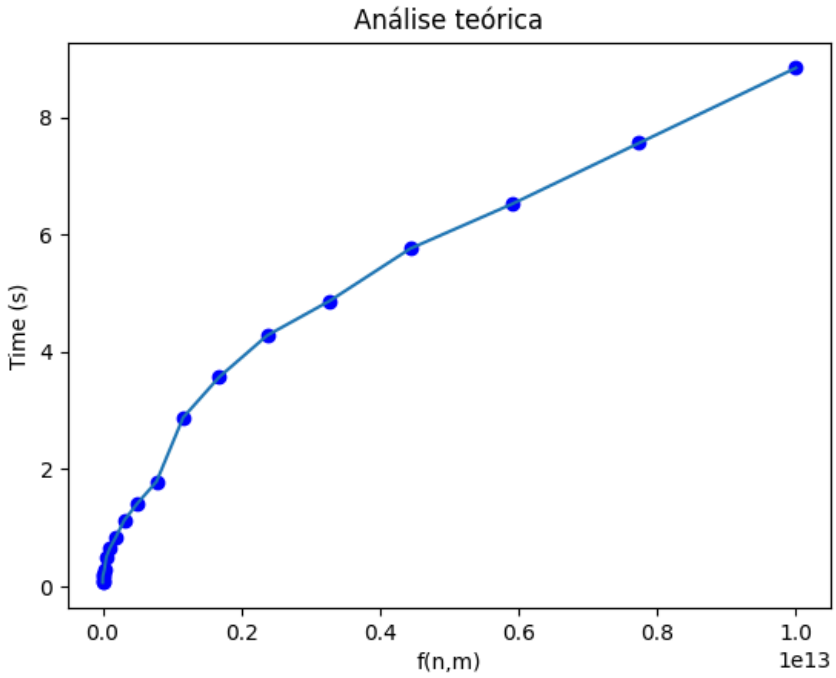
Complexidade global da solução: $O(m^3n^2)$.

```
BDyn(T, X, n, m, c)
  let table be a new vector of size m×m
  for i = 0 to m do
  |  store node res = X[i] in table[i][i]
  for i = 1 to m do
  |  for j = 0 to m-i do
  |  |  let row = j, col = i+j, registered[n] = {0}
  |  |  for k = col to row do
  |  |  |  let left = table[row][k-1]
  |  |  |  let right = table[k][col]
  |  |  |  for each l in left do
  |  |  |  |  for each r in right do
  |  |  |  |  |  let node be a new node
  |  |  |  |  |  node.l = T[l.l-1][l.r-1] , node.r =  T[r.l-1][r.r-1]
  |  |  |  |  |  store node in table[row][col]
```

**Avaliação Experimental dos Resultados**

Para a avaliação experimental, foram testados 20 cenários distintos. Iniciou-se com uma sequência de tamanho n = 5 e m = 50, aumentando n em incrementos de 5 e m em incrementos de 50 a cada etapa, até atingir n = 100 e m = 1000.

Esperava-se que a função apresentasse um comportamento linear; contudo, devido às otimizações aplicadas, ela assume uma forma logarítmica.




As otimizações incluem evitar a repetição de valores na mesma parcela da tabela, avançar para a próxima etapa quando esta atinge n valores, e encerrar a função quando o valor desejado é encontrado.