Análise de Algoritmos

Alunos:
Maurício Barbosa – 1712130037
Milena Nobre – 1722130027

QUESTÃO 01

- Implementar a solução recursiva do SUBSET SUM que informa se existe algum subconjunto que é igual a capacidade informada.

  R: arquivo P1_RECURSIVO.c

- Demonstrar a complexidade de forma matemática através do método da expansão.

  R:

  Fazendo a contagem de instruções temos:

  $T(n) = 2 + T(n-1) + T(n-1)$

  $T(n) = 2T(n-1) + 2$

  Aplicando o método da expansão:

  $T(n) = 2 + 2(T(n-1))$

  $T(n-1) = 2 + 2(T(n-2))$

  $T(n-2) = 2 + 2T(n-3)$

  Então temos:

  $T(n) = 2 + 2(2 + 2T(n-2)) = 2 + 2^2 + 2^2T(n-2)$

  $T(n) = 2 + 2^2 + 2^2(2 + 2T(n-3)) = 2 + 2^2 + 2^3 + 2^3T(n-3)$

  Portanto:

  Consumo de tempo em até **O(2^n)**

- Gráfico da solução recursiva:

  Análise no final.

QUESTÃO 02

- Implementar a solução do SUBSET SUM através de programação dinâmica que informe pelo menos 1 subconjunto que atende a capacidade informada.

  R: Arquivo P1_DINAMICO.cpp

- Demonstrar a complexidade do algoritmo através da contagem de instruções:

  R:

  Fazendo a contagem de instruções temos:

  $C(n) = (2n+1) + (5cn + 3c + 1) + (1)$

  $C(n) = 2n + 3 + 3n + 5cn$

  Portanto:

  Consumo de tempo em até **O(N*C)**

- Gráfico da programação dinâmica:

  Análise no final.

QUESTÃO 03

- Implementar a solução do SUBSET SUM utilizando o método de Backtracking resultando em TODOS os subconjuntos que atendem a capacidade informada.

  R: arquivo P1_BACKTRACKING.cpp

- Demonstrar a complexidade do algoritmo através da contagem de instruções em caso não recursivo, ou através do método da expansão caso recursivo.

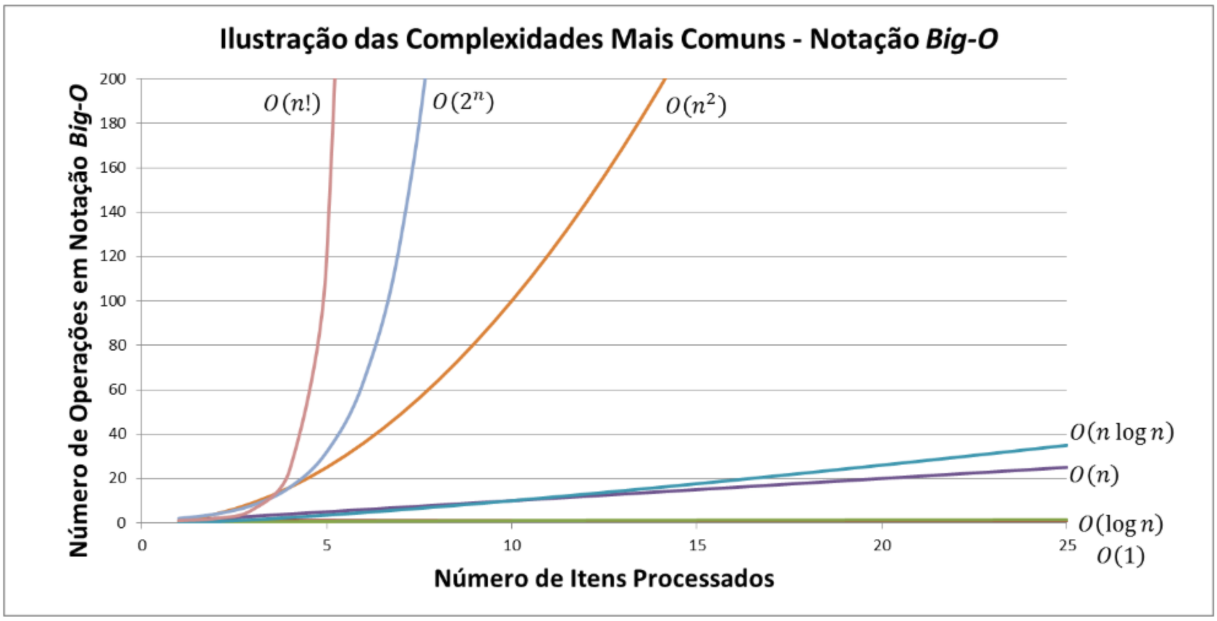
  Consumo de tempo em até **O(2^n)**

- Gráfico da solução Backtracking:

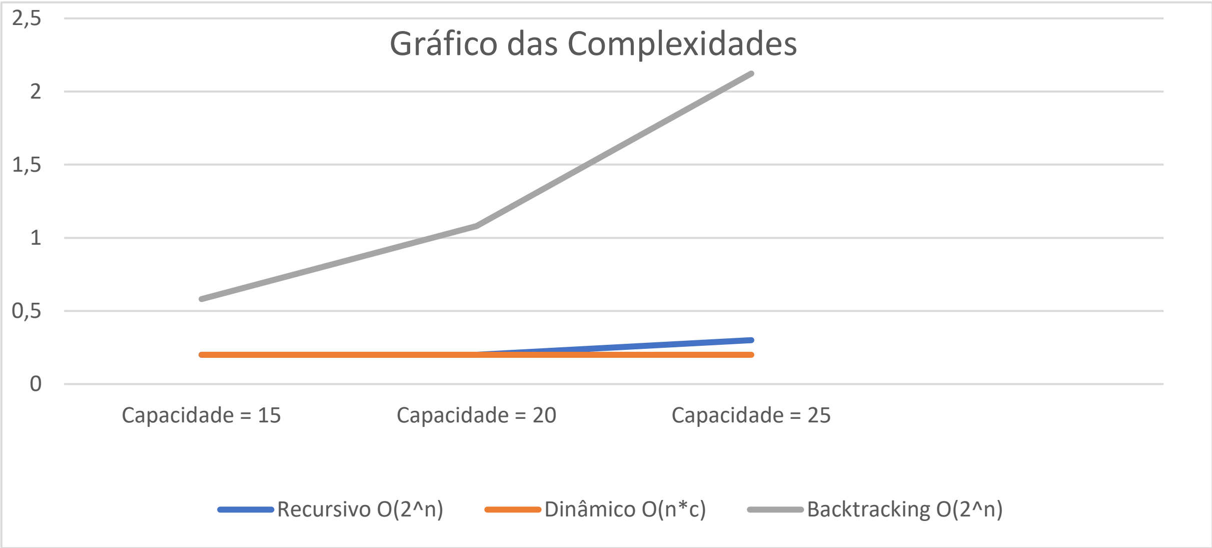
  Análise no final.

ANÁLISE NO GRÁFICO

O gráfico abaixo ilustra as curvas de crescimento mais comuns dando uma visão geral a respeito das complexidades.



No gráfico resultante dos algoritmos desenvolvidos, o eixo x corresponde a capacidade de elementos do conjunto utilizado e no eixo y o tempo em segundos. Ao analisar o gráfico, é possível verificar que no Backtracking obtemos a complexidade O(2^n) que é aquela em que à medida que "n" aumenta, o fator analisado (tempo) aumenta exponencialmente. Isso vale também para o método recursivo.



O algoritmo de programação dinâmica, consiste em preencher uma tabela bidimensional, com C+1 linha e n+1 colunas, cada chamada leva um tempo fixo de O(n) e C chamadas são feitas, então o tempo total será de O(n*c), o que torna o algoritmo exponencial.

Para comparar o desempenho dos 3 casos devemos levar em conta que para cada algoritmo foram pedidas instruções diferentes, o Backtracking destaca-se com maior tempo pois ele mostra todos os subconjuntos existentes que a resultam na somatória. Já o recursivo ele só retorna se existe ou não subconjuntos e para o dinâmico deveria mostrar apenas o primeiro subconjunto existente. O que levou aos dois pouco tempo na execução.