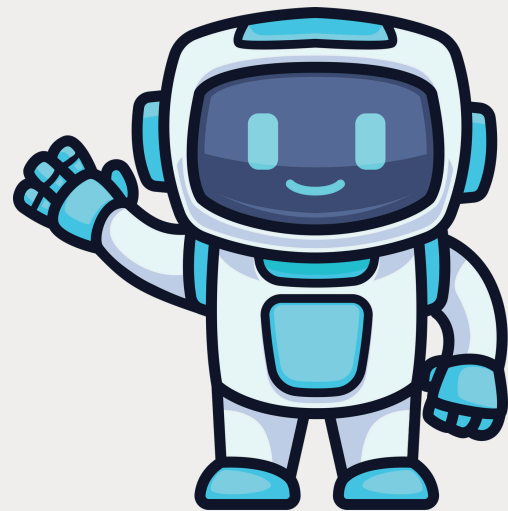




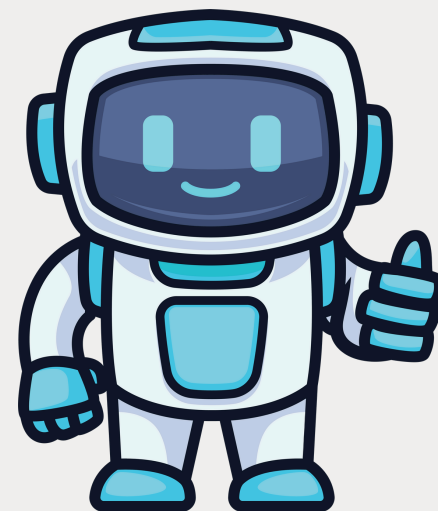
BIN PACKING (BP)

Algoritmos e heurísticas para problemas clássicos

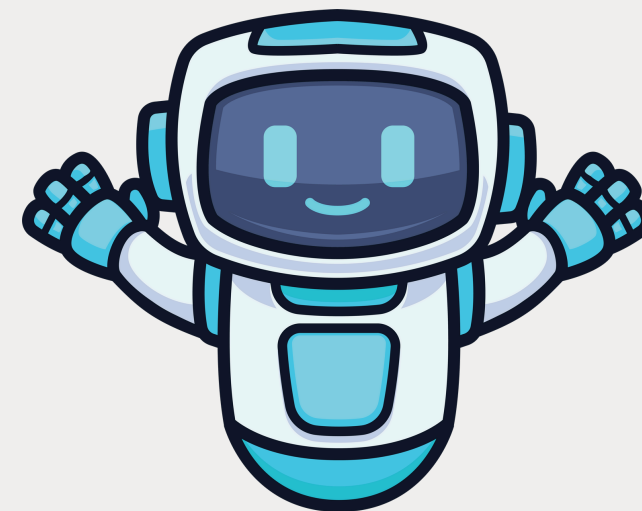
NOSSO TIME



Arthur Hauck



Caio Diniz



Izabelle Tome



O QUE É O PROBLEMA DO BIN PACKING?

Esse problema clássico da computação é um problema onde:

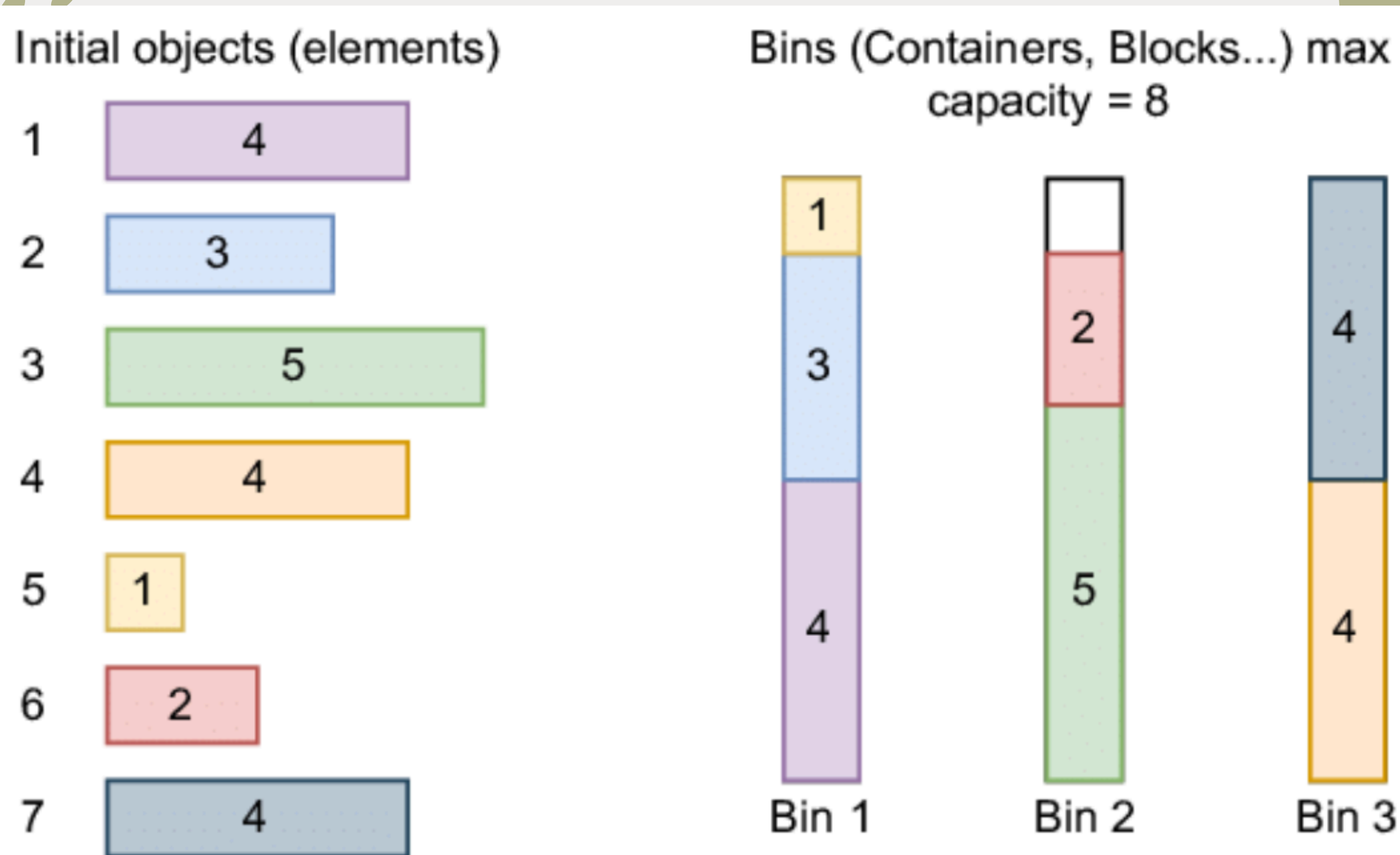
Dado um número qualquer de itens, eles devem ser guardados na menor quantidade de caixas possível.



Essas caixas possuem uma restrição de peso, onde apenas itens que caibam nelas serão guardados.

Caso o item não caiba na caixa, ele deve ser colocado em outra que o comporte.



O QUE É O PROBLEMA DO BIN PACKING?





O PROBLEMA DO BIN PACKING NO MUNDO REAL

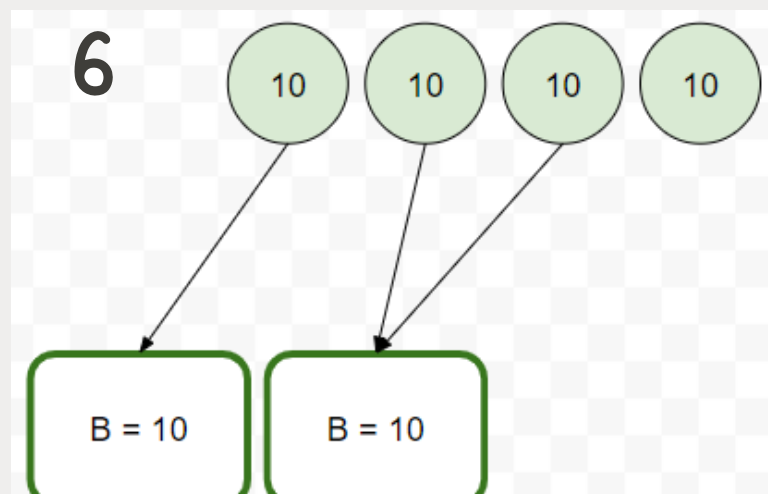
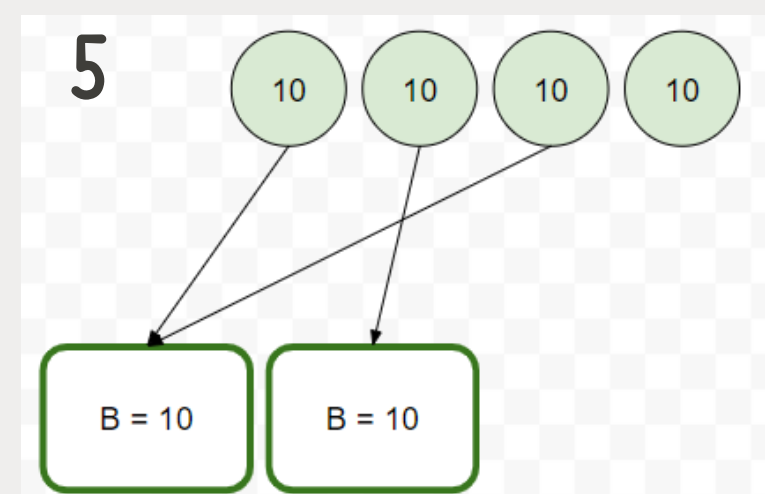
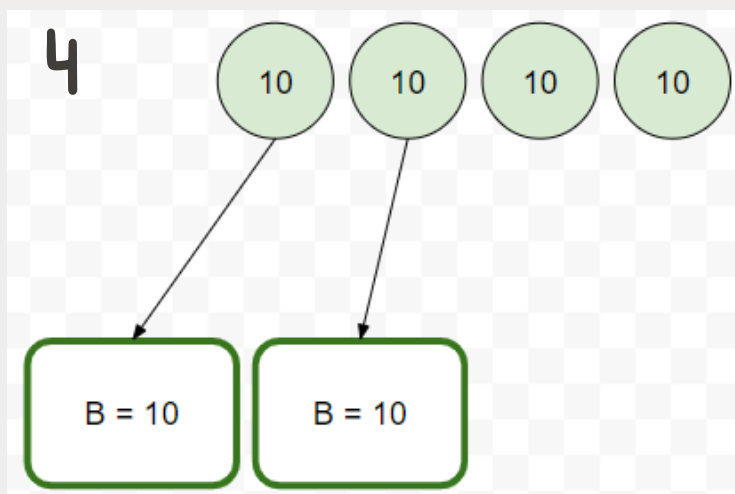
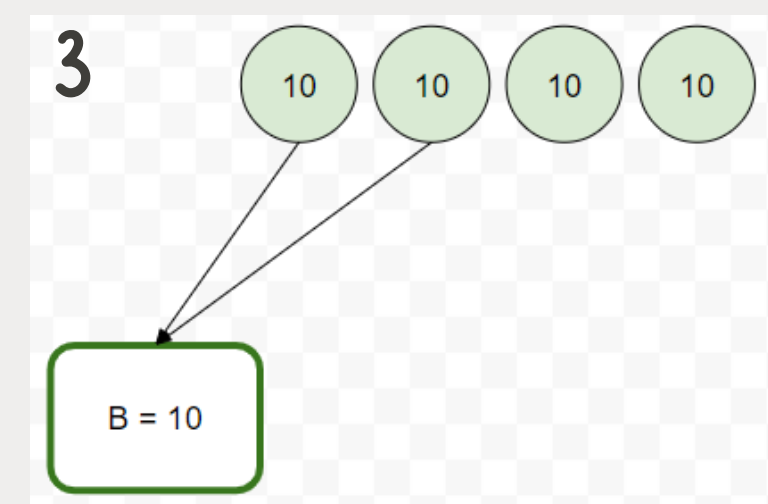
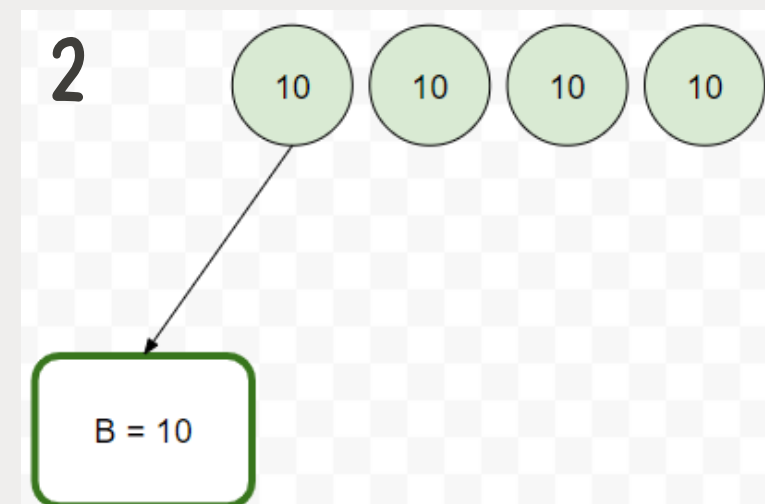
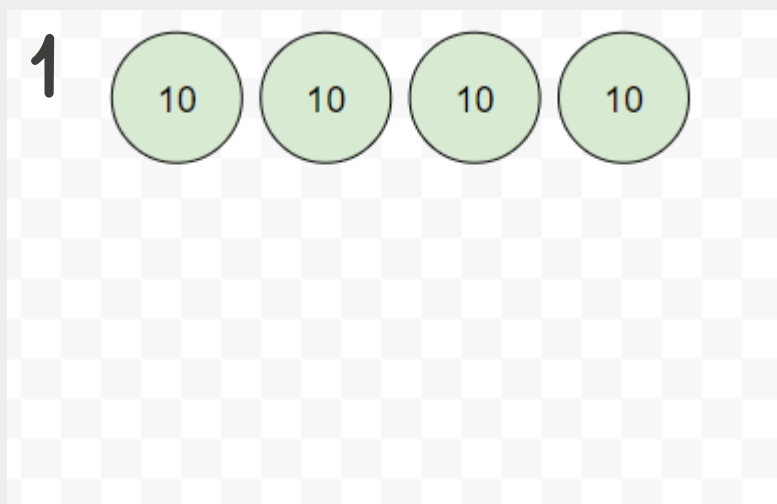
1. Logística e transporte: ajuda a determinar a melhor maneira de empacotar as mercadorias nos containeres para otimizar o espaço e o peso, garantindo que o menor número de containeres seja usado.
2. Armazenamento em Data Centers: ajuda a distribuir as aplicações nos servidores de forma que o uso dos recursos seja otimizado, minimizando o número de servidores ativos e, conseqüentemente, o consumo de energia.
3. Corte de Materiais na indústria: ajuda a determinar a melhor maneira de dispor as peças a serem cortadas na chapa, de modo a minimizar o desperdício de material.
4. Programação de horários em linhas de produção: ajuda a distribuir as tarefas de forma que o uso dos recursos seja otimizado, minimizando o número de operadores ou máquinas necessários.
5. Hospedagem: ajuda a alocar os hóspedes nos quartos de maneira que a ocupação seja maximizada, e o número de quartos desocupados seja minimizado.



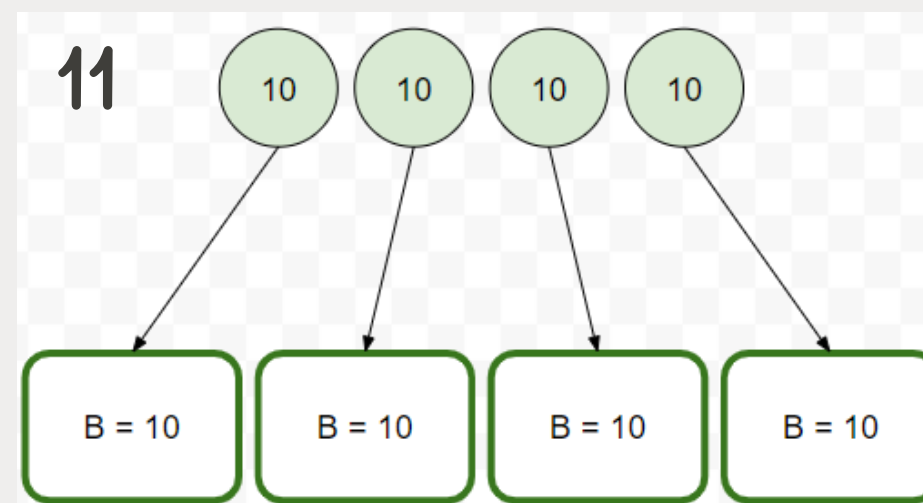
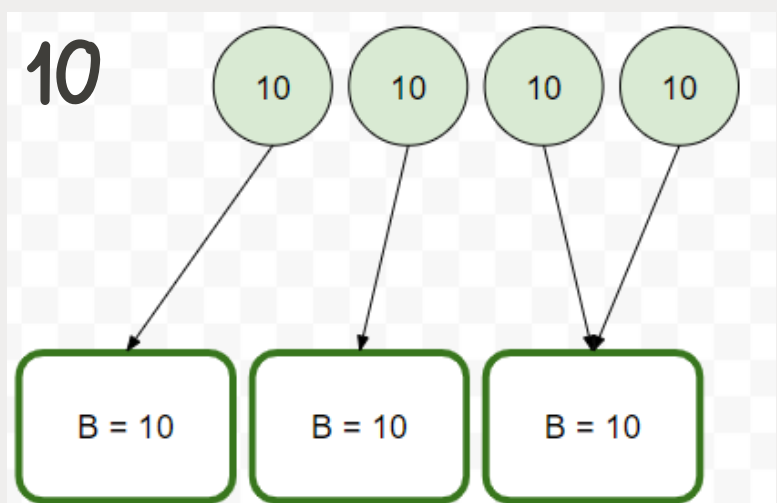
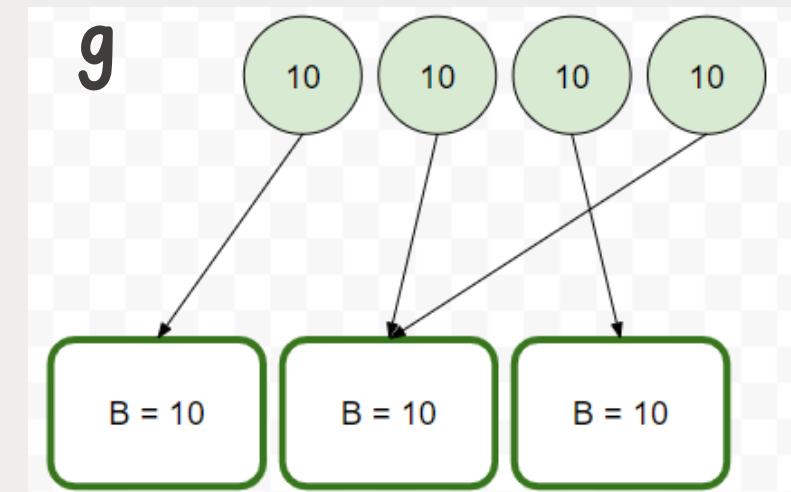
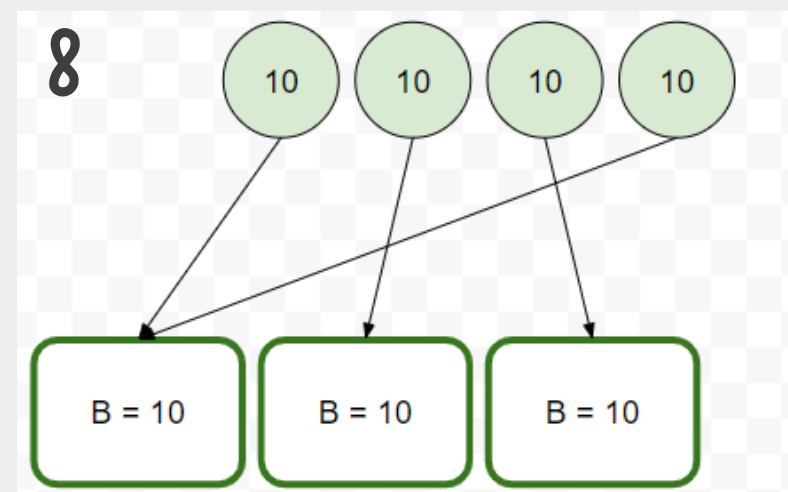
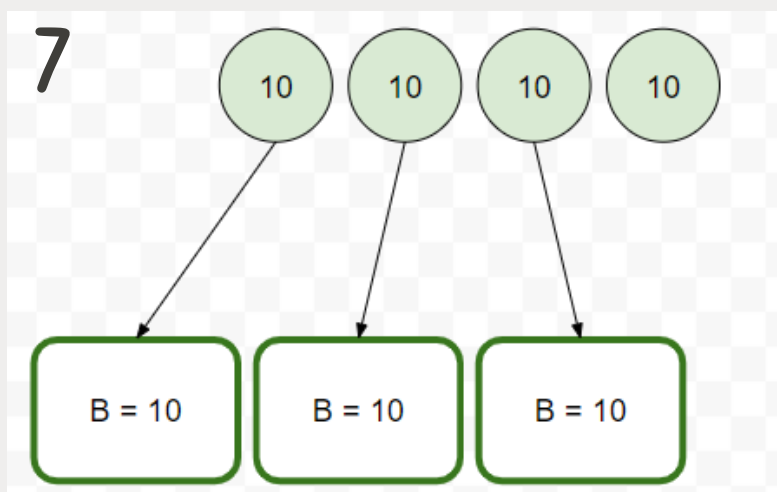
PSEUDOCODIGO: BACKTRACKING

```
1 BP_Backtracking(itens, capacidadeCaixa)
2   melhorSolucao ← nulo
3
4   função éVálido(caixasAtuais)
5       para cada caixa em caixasAtuais
6           somaCaixa ← soma dos itens em caixa
7           se somaCaixa > capacidadeCaixa
8               retornar falso
9       retornar verdadeiro
10
11  função empacotarItens(índice, caixasAtuais)
12      se índice == itens.tamanho()
13          se éVálido(caixasAtuais)
14              se melhorSolucao for nulo ou caixasAtuais.quantidade() < melhorSolucao.quantidade()
15                  melhorSolucao ← cópia profunda de caixasAtuais
16      retornar
17
18      para cada caixa em caixasAtuais
19          caixa.add(itens[índice])
20          empacotarItens(índice + 1, caixasAtuais)
21          caixa.remove(itens[índice])
22
23      novaCaixa ← [itens[índice]]
24      caixasAtuais.add(novaCaixa)
25      empacotarItens(índice + 1, caixasAtuais)
26      caixasAtuais.remove(novaCaixa)
27
28      retornar nulo
29
30  empacotarItens(0, [])
31  retornar melhorSolucao
32
```

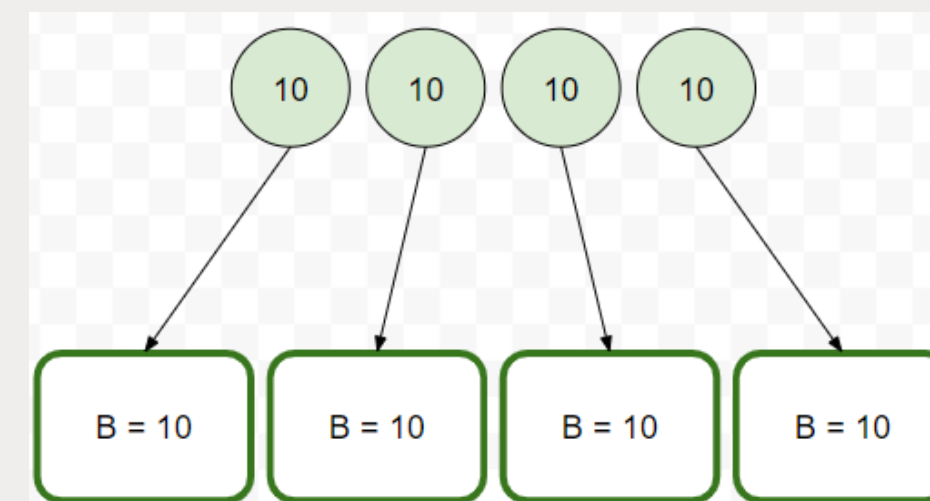

EXECUCAO DO PSEUDOCODIGO: BACKTRACKING



EXECUCAO DO PSEUDOCODIGO: BACKTRACKING



RESULTADO:






ANALISE DO ALGORITMO DE BACKTRACKING

O algoritmo ira analisar a distribuicao dos “n” itens em todos os pacotes.

Por isso, ele tem complexidade:

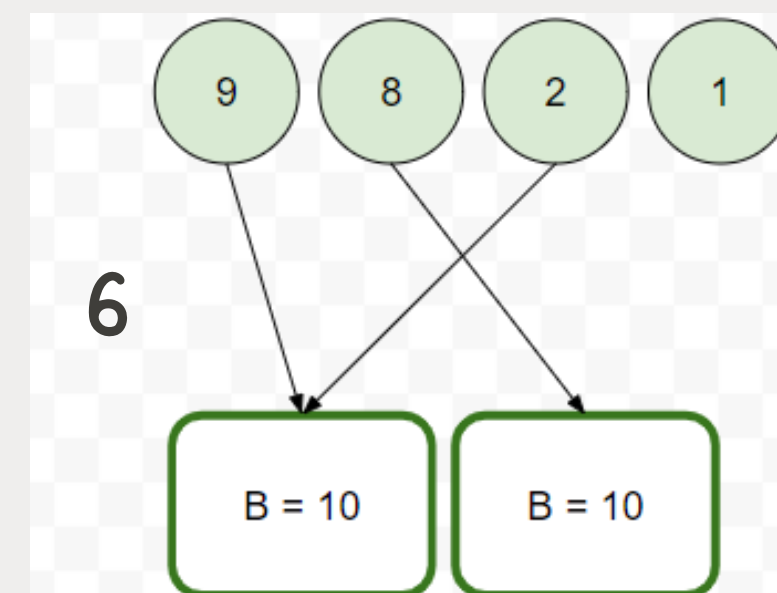
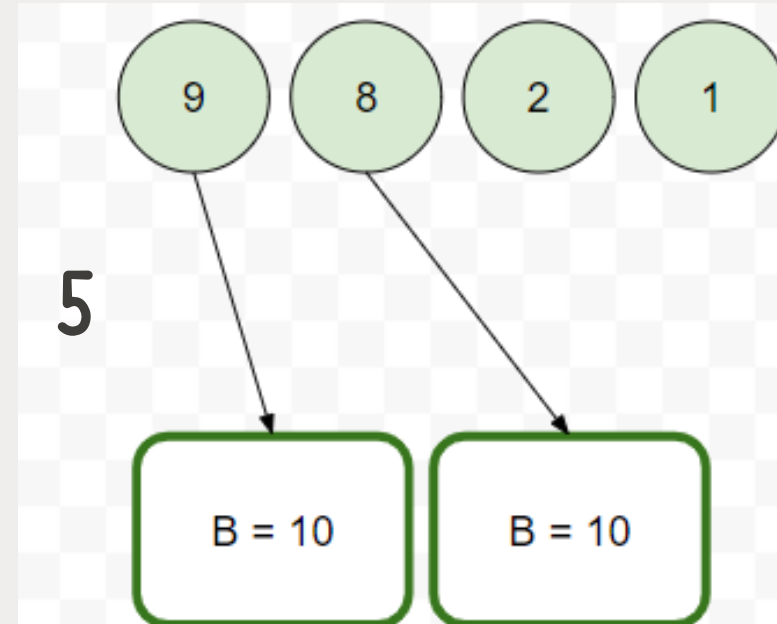
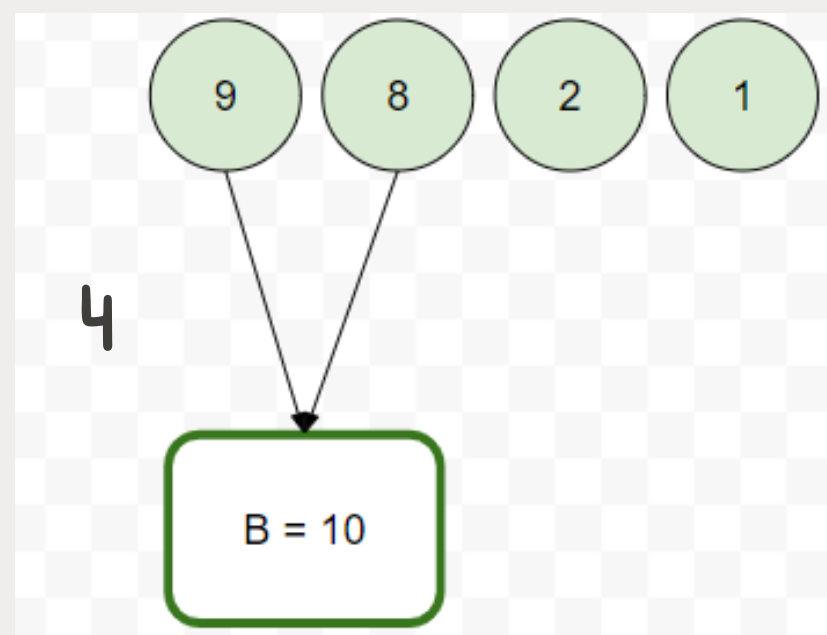
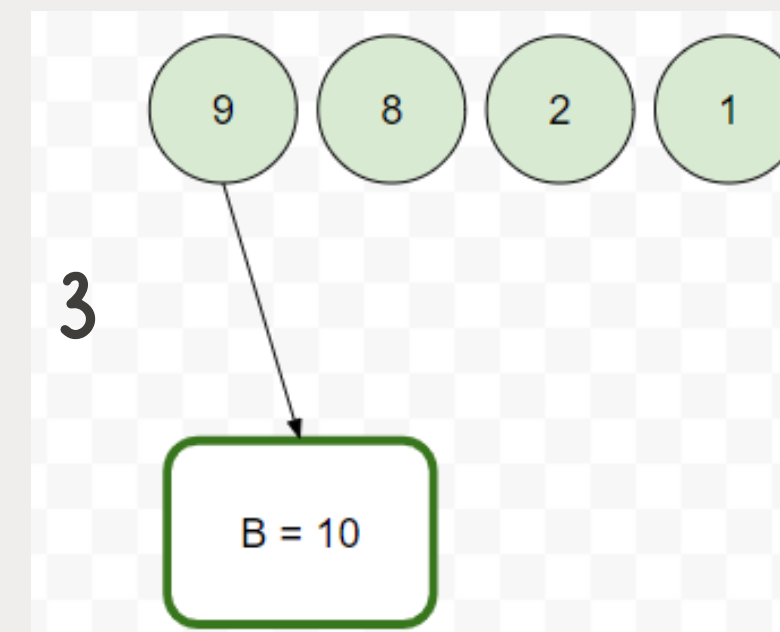
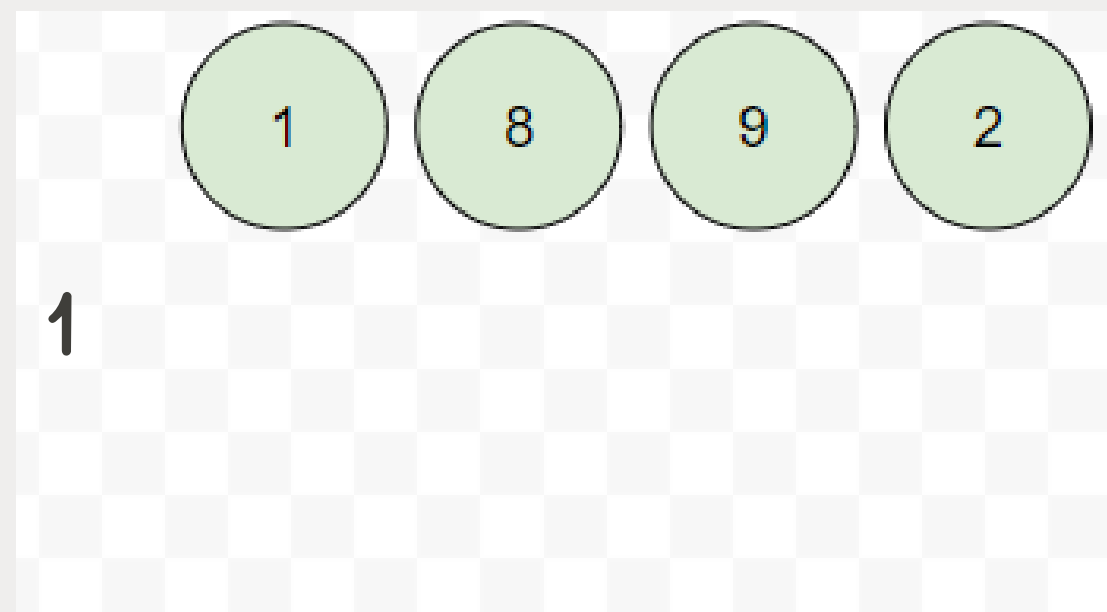
$$O(2^n)$$



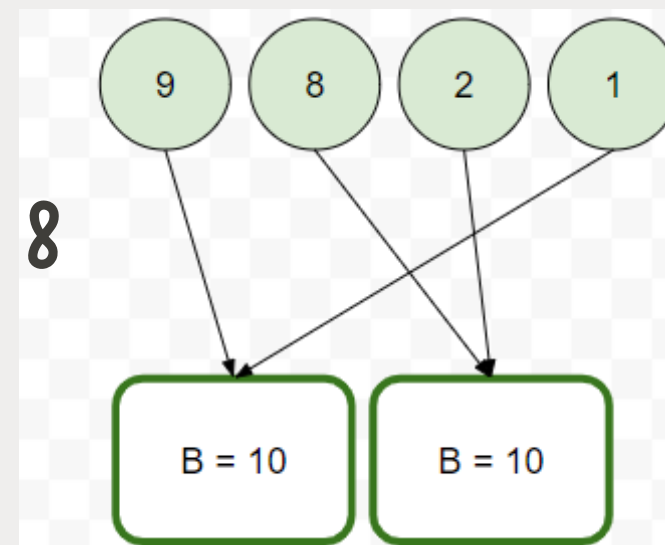
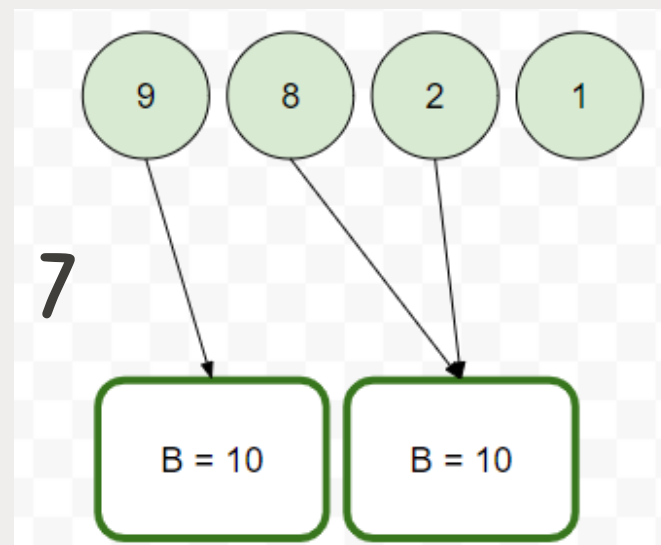
PSEUDOCODIGO: HEURISTICA FIRST-FIT DECREASING (FFD)

```
1  BP_Heurística(itens, capacidadeCaixa)
2      ordenar_itens_decrescente(itens)
3
4      caixas <- []
5
6      para cada item em itens
7          colocado <- falso
8          para cada caixa em caixas
9              se soma(caixa) + item <= capacidadeCaixa
10                 caixa.add(item)
11                 colocado <- verdadeiro
12                 break
13             se não colocado
14                 novaCaixa <- [item]
15                 caixas.add(novaCaixa)
16
17  ✨ retornar caixas
18
```

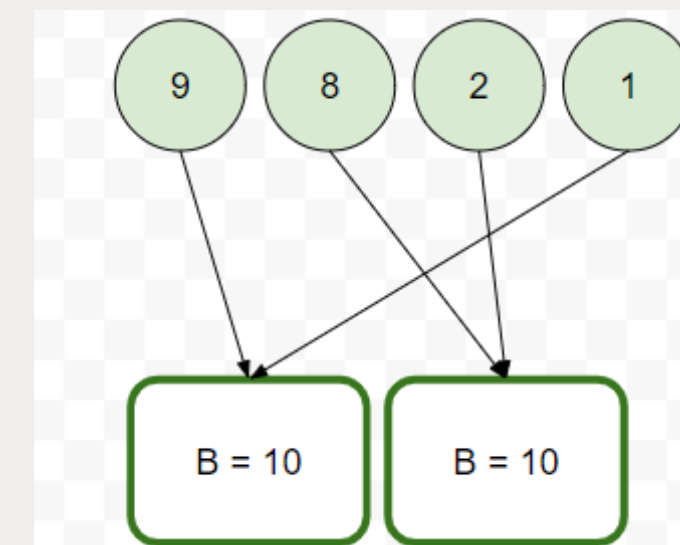
EXECUCAO DO PSEUDOCODIGO: HEURISTICA





EXECUCAO DO PSEUDOCODIGO: HEURISTICA



RESULTADO:



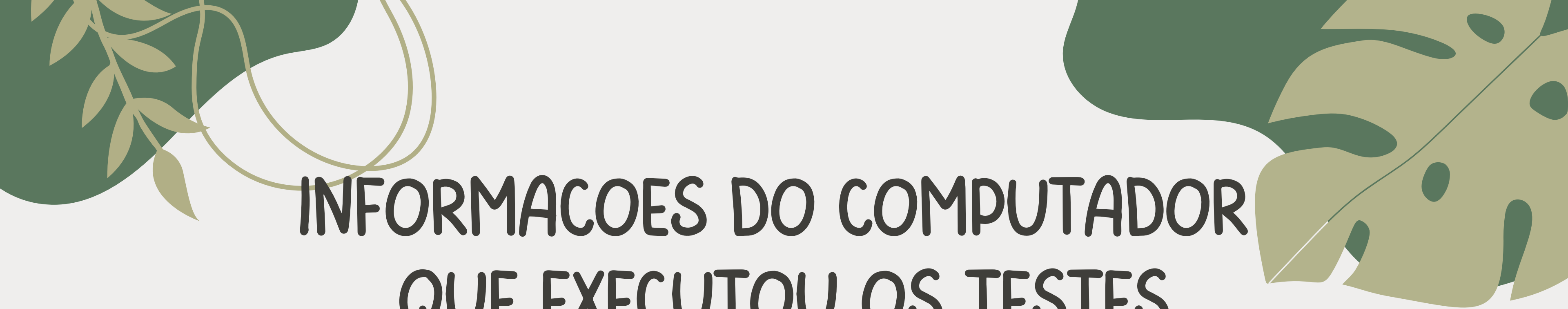


ANALISE DO ALGORITMO DE HEURISTICA

No pior caso, o algoritmo pode vir a analisar cada item para cada pacote existente.

Por isso, ele tem complexidade:

$$O(n^2)$$



INFORMACOES DO COMPUTADOR QUE EXECUTOU OS TESTES



SO: macOS 14.6.1 (23G93)

Processador: 2,6 GHz Intel Core i7 6-Core

Memória: 16 GB 2667 MHz DDR4

Arquitetura: x86_64

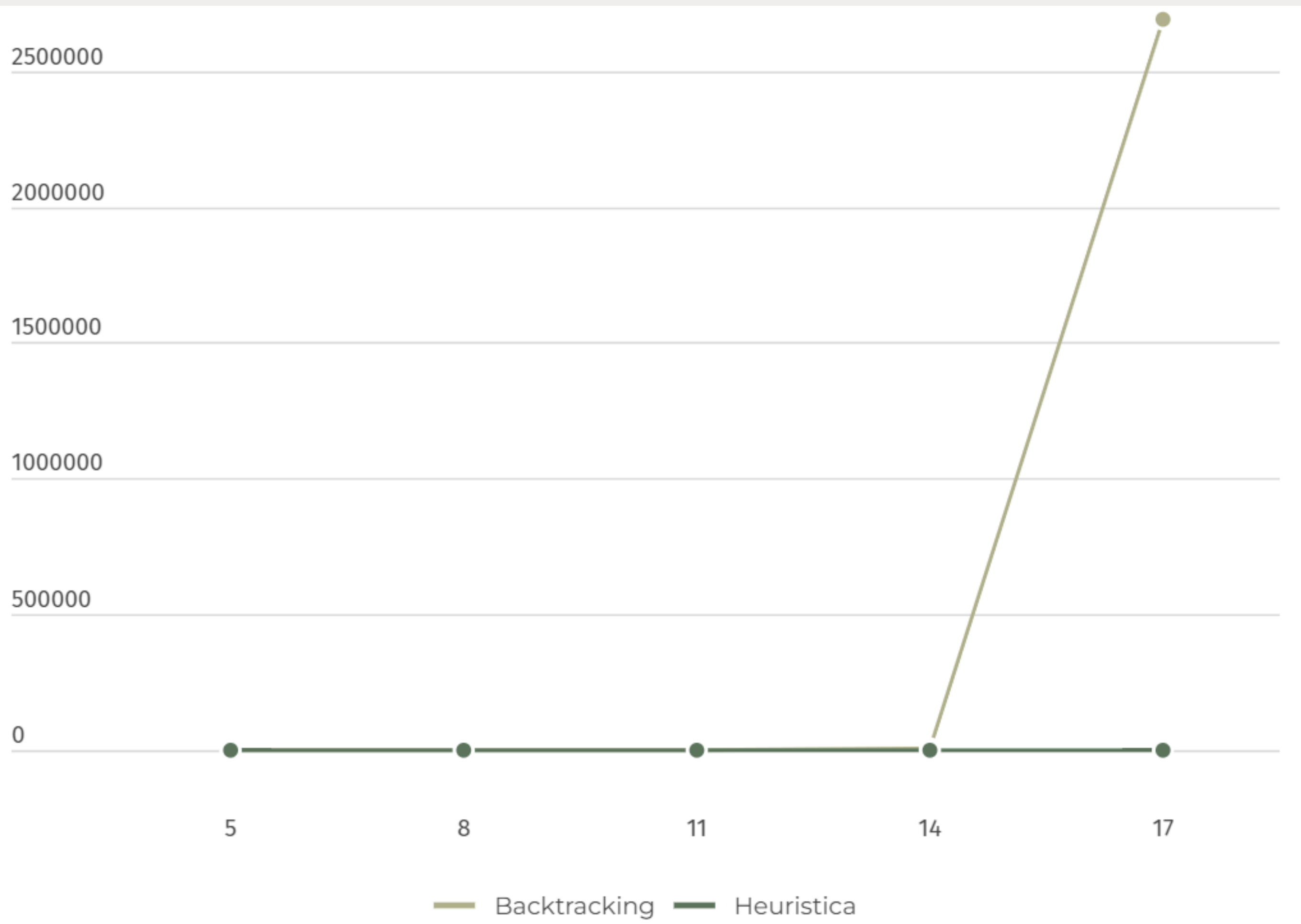
Unidade de armazenamento: Macintosh DH 500Gb



TESTES REALIZADOS:
AUMENTO NO NUMERO DE ITENS

Teste	Solucao Otima	Backtracking: Solucao	Tempo(média 4)	Heuristica: Solucao	Tempo(média 4)
n = 5 s = [5, 6, 4, 10, 8] B = 10	4 caixas E = {{10}, {6,4}, {8}, {5}}	4 caixas E = {{5,4}, {6}, {10}, {8}}	0.33 ms	4 caixas E = {{10}, {8}, {6,4}, {5}}	0.13 ms
n = 8 s = [9, 6, 2, 4, 3, 7, 1, 5] B = 10	4 caixas E = {{9,1}, {6,4}, {7,3}, {5,2}}	4 caixas E = {{9,1}, {6,2}, {4,5}, {3,7}}	4.47 ms	4 caixas E = {{9,1}, {7,3}, {6,4}, {5,2}}	0.13 ms
n = 11 s = [1, 10, 5, 7, 3, 4, 6, 9, 2, 8, 3] B = 10	6 caixas E = {{10}, {9,1}, {8,2}, {7,3}, {6,4}, {5,3}}	6 caixas E = {{1,5,3}, {10}, {7,3}, {4,6}, {9}, {2,8}}	52.14 ms	6 caixas E = {{10}, {9,1}, {8,2}, {7,3}, {6,4}, {5,3}}	0.27 ms
n = 14 s = [7, 4, 2, 9, 1, 5, 6, 3, 10, 8, 4, 7, 1, 2] B = 10	7 caixas E = {{10}, {9,1}, {8,2}, {7,3}, {7,2,1}, {6,4}, {5,4}}	7 caixas E = {{7,2,1}, {4,5,1}, {9}, {6,4}, {3,7}, {10}, {8,2}}	7248,91 ms	7 caixas E = {{10}, {9,1}, {8,2}, {7,3}, {7,2,1}, {6,4}, {5,4}}	0.22 ms
n = 17 s = [10, 5, 9, 6, 1, 2, 7, 4, 8, 3, 10, 6, 2, 1, 8, 4, 7] B = 10	10 caixas E = {{10}, {10}, {9,1}, {8,2}, {8,2}, {7,3}, {7,1}, {6,4}, {6,4}, {5}}	10 caixas E = {{10}, {5,1,2,2}, {9,1}, {6,4}, {7,3}, {8}, {10}, {6,4}, {8}, {7}}	2690279.42 ms	10 caixas E = {{10}, {10}, {9,1}, {8,2}, {8,2}, {7,3}, {7,1}, {6,4}, {6,4}, {5}}	0.48 ms

Diferença no tempo de execucao em milissegundos (ms) dos algoritmos em funcao do numero de itens:



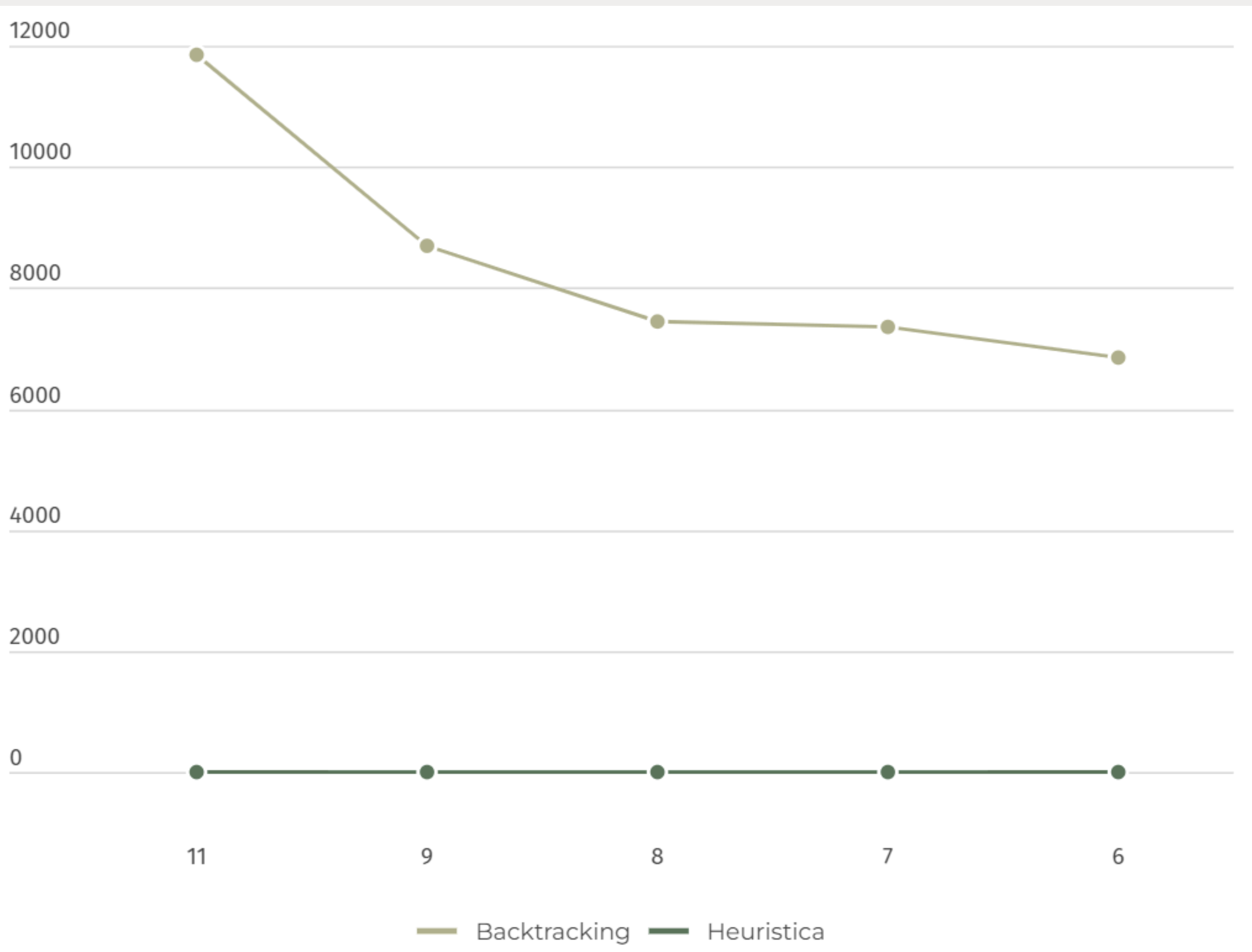
↶↷	A	B	C
1		Backtracking	Heuristica
2	5	0,33	0,13
3	8	4,47	0,13
4	11	52,14	0,27
5	14	7248,91	0,22
6	17	2690279,42	0,48




TESTES REALIZADOS:
DIMINUICAO NO TAMANHO DAS CAIXAS

Teste	Solucao Otima	Backtracking: Solucao Tempo(média 4)		Heuristica: Solucao	Tempo
n = 14 s = [4, 2, 6, 1, 3, 5, 2, 6, 4, 1, 3, 5, 2, 4] B = 11	5 caixas E = {{4,2,1,3,1}, {6,5}, {2,6,3}, {4,5,2}, {4}}	5 caixas E = {{4,2,1,3,1}, {6,5}, {2,6,3}, {4,5,2}, {4}}	11849.27 ms	5 caixas E = {{6,5}, {6,5}, {4,4,3}, {4,3,2,2}, {2,1,1}}	0.27 ms
n = 14 s = [4, 2, 6, 1, 3, 5, 2, 6, 4, 1, 3, 5, 2, 4] B = 9	6 caixas E = {{4,2,1,2}, {6,3}, {5,4}, {6,1,2}, {3,5}, {4}}	6 caixas E = {{4,2,1,2}, {6,3}, {5,4}, {6,1,2}, {3,5}, {4}}	8699.37 ms	6 caixas E = {{6,3}, {6,3}, {5,4}, {5,4}, {4,2,2,1}, {2,1}}	0.27 ms
n = 14 s = [4, 2, 6, 1, 3, 5, 2, 6, 4, 1, 3, 5, 2, 4] B = 8	6 caixas E = {{6,2}, {6,2}, {5,3}, {5,3}, {4,4}, {4,2,1,1}}	6 caixas E = {{4,2,1,1}, {6,2}, {3,5}, {6,2}, {4,4}, {3,5}}	7438.47 ms	6 caixas E = {{6,2}, {6,2}, {5,3}, {5,3}, {4,4}, {4,2,1,1}}	0.22 ms
n = 14 s = [4, 2, 6, 1, 3, 5, 2, 6, 4, 1, 3, 5, 2, 4] B = 7	7 caixas E = {{6,1}, {6,1}, {5,2}, {5,2}, {4,3}, {4,3}, {4,2}}	7 caixas E = {{4,2,1}, {6,1}, {3,4}, {5,2}, {6}, {3,4}, {5,2}}	7357.17 ms	7 caixas E = {{6,1}, {6,1}, {5,2}, {5,2}, {4,3}, {4,3}, {4,2}}	0.29 ms
n = 14 s = [4, 2, 6, 1, 3, 5, 2, 6, 4, 1, 3, 5, 2, 4] B = 6	8 caixas E = {{6}, {6}, {5,1}, {5,1}, {4,2}, {4,2}, {4,2}, {3,3}}	8 caixas E = {{4,2}, {6}, {1,5}, {3,3}, {2,4}, {6}, {1,5}, {2,4}}	6848.57 ms	8 caixas E = {{6}, {6}, {5,1}, {5,1}, {4,2}, {4,2}, {4,2}, {3,3}}	0.31 ms

Diferença no tempo de execucao em milissegundos (ms) dos algoritmos em funcao do tamanho das caixas:



	A	B	C
1		Backtracking	Heuristica
2	11	11849,27	0,27
3	9	8699,37	0,27
4	8	7438,47	0,22
5	7	7357,17	0,29
6	6	6848,57	0,31



FONTE CASOS DE TESTE

Os casos de teste foram montados em parceria entre os integrantes do grupo e a IA ChatGPT.

Os integrantes propunham um conjunto de itens e a capacidade das caixas e o GPT propunha uma solucao que era analisada pelo grupo se seria eleita como uma das possiveis solucoes otimas ou nao.

Caso a solucao otima do GPT nao fosse eleita, o grupo discutia sobre os itens e propunha sua propria solucao, analisada em alto nivel sem uso de tecnologias.

CONCLUSAO

O algoritmo de backtracking conseguiu, para os casos de teste informados, encontrar a solucao otima em todas as execucoes, porem se destacou de forma negativa pelo aumento consideravel de retornar uma resposta ao aumentar o numero de itens analisados.

Enquanto, o algoritmo da heuristica tambem encontrou em todas as execucoes, a solucao otima e mantendo o seu tempo com poucas variacoes mesmo aumento o numero de itens analisados na mesma proporcao do backtracking. Por isso, conclui-se que o algoritmo da heuristica FFD e um algoritmo muito bom para se resolver o problema da Bin Packing.

O tempo medio de execucao da heuristica para as entradas informadas foi de:

0.25 ms

Enquando o tempo medio de execucao do backtracking foi de:

273925.95 ms \cong 5 minutos