

Universidade Federal de Ouro Preto
PCC104 - Projeto e Análise de Algoritmos
Prova - Algoritmos Gulosos

Prof. Rodrigo Silva

July 17, 2023

Orientações

- É obrigatória a entrega do código fonte da prática de branch and bound. Provas sem os códigos fonte não serão corrigidas e terão nota 0.
- A avaliação do código apresentado entra na avaliação das questões relacionadas.

Questões

1. *Análise de Algoritmo Iterativo Simples*

Considerando o algoritmo de ordenação apresentado na Figura (para a versão C++, veja), que implementa o método de "Bubble Sort". Apresente a análise completa e detalhada deste algoritmo?

2. *Análise de Algoritmo Recursivo Simples*

Veja algoritmo de busca binária apresentado na figura (aara a versão em C++ veja) e responda:

- (a) Apresente a análise completa e detalhada da complexidade de tempo do pior caso deste algoritmo.
- (b) Apresente a análise completa e detalhada da complexidade de tempo do melhor caso deste algoritmo.

3. *Análise e Perguntas Teóricas Sobre o Branch and Bound*

- (a) Explique o conceito de branch and bound e sua aplicação na resolução de problemas de otimização.
- (b) Compare o branch and bound com o método de força bruta. Em quais cenários cada um seria preferível e por quê?
- (c) Como a estratégia de branch and bound pode impactar o custo computacional da resolução de um problema?

4. *Perguntas Teóricas Sobre Classes de Problemas (P , NP , NP -completo)*

É possível que $P = NP$? Explique sua resposta, discutindo as implicações se P fosse de fato igual a NP .

```

1     def bubble_sort(lista):
2         for i in range(len(lista)):
3             for j in range(0, len(lista) - i - 1):
4                 if lista[j] > lista[j + 1]:
5                     lista[j], lista[j + 1] = lista[j + 1], lista[j]
6

```

Figure 1: Algoritmo 1 - Versão Python

```

1     void bubble_sort(vector<int>& lista) {
2         int i, j;
3         bool swapped;
4         int n = lista.size();
5
6         for (i = 0; i < n-1; i++) {
7             swapped = false;
8             for (j = 0; j < n-i-1; j++) {
9                 if (lista[j] > lista[j+1]) {
10                    int temp = lista[j];
11                    lista[j] = lista[j+1];
12                    lista[j+1] = temp;
13                    swapped = true;
14                }
15            }
16            if (swapped == false)
17                break;
18        }
19    }
20

```

Figure 2: Algoritmo 1 - Versão C++

```

1     def binary_search(array, low, high, target):
2         if high >= low:
3             mid = (high + low) // 2
4             if array[mid] == target:
5                 return mid
6             elif array[mid] > target:
7                 return binary_search(array, low, mid - 1, target)
8             else:
9                 return binary_search(array, mid + 1, high, target)
10        else:
11            return -1
12

```

Figure 3: Algoritmo 2 - Versão Python

```

1  int binary_search(vector<int>& array, int low, int high, int target) {
2      if (high >= low) {
3          int mid = low + (high - low) / 2;
4          if (array[mid] == target)
5              return mid;
6          else if (array[mid] > target)
7              return binary_search(array, low, mid - 1, target);
8          else
9              return binary_search(array, mid + 1, high, target);
10     } else {
11         return -1;
12     }
13 }
14

```

Figure 4: Algoritmo 2 - Versão Python