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) Explique como este algoritmo funciona.

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

   (c) 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.

```python
        def bubble_sort(lista):
            for i in range(len(lista)):
                for j in range(0, len(lista) - i - 1):
                    if lista[j] > lista[j + 1]:
                        lista[j], lista[j + 1] = lista[j + 1], lista[j]

```

Figure 1: Algoritmo 1 - Versão Python

```cpp
    void bubble_sort(vector<int>& lista) {
        int i, j;
        bool swapped;
        int n = lista.size();

        for (i = 0; i < n-1; i++) {
            swapped = false;
            for (j = 0; j < n-i-1; j++) {
                if (lista[j] > lista[j+1]) {
                    int temp = lista[j];
                    lista[j] = lista[j+1];
                    lista[j+1] = temp;
                    swapped = true;
                }
            }

            if (swapped == false)
                break;
        }
}


```

Figure 2: Algoritmo 1 - Versão C++

```python
    def binary_search(array, low, high, target):
        if high >= low:
            mid = (high + low) // 2
            if array[mid] == target:
                return mid
            elif array[mid] > target:
                return binary_search(array, low, mid - 1, target)
            else:
                return binary_search(array, mid + 1, high, target)
        else:
            return -1

```

Figure 3: Algortimo 2 - Versão Python

```
int binary_search(vector<int>& array, int low, int high, int target) {
    if (high >= low) {
        int mid = low + (high - low) / 2;
        if (array[mid] == target)
            return mid;
        else if (array[mid] > target)
            return binary_search(array, low, mid - 1, target);
        else
            return binary_search(array, mid + 1, high, target);
    } else {
        return -1;
    }
}

```

Figure 4: Algortimo 2 - Versão Python