Gabarito da Lista de Exercícios Sobre Divisão e Conquista

1. (Análise de Complexidade) Qual a complexidade temporal do seguinte algoritmo?

Algoritmo 0.1 (DC)
Entrada Uma seqüência $s_1, \ldots, s_n$ de números inteiros, $n \geq 1$.

```
if (n = 1) then
  return s_1
end if
v_1 := DC(s_1, ..., s_⌊n/2⌋)
v_2 := DC(s_⌊n/2⌋+1, s_n)
return min(v_1, v_2)
```

Apresente a equação de recorrência que descreve a complexidade do algoritmo, bem como sua solução.

A equação de recorrência do algoritmo é $T(n) = 2T(\frac{n}{2}) + 1$. Esta equação de recorrência foi analisada em sala de aula e tem complexidade O(n).

2. (NewSort) Considere o algoritmo `NewSort()` de ordenação de dados descrito a seguir. O algoritmo `NewSort`(S) recebe como entrada um conjunto S de $|S| = n$ números reais uniformemente distribuídos em R. O algoritmo é um algoritmo recursivo. A cada chamada recursiva o algoritmo

   a) calcula a mediana m dos números da entrada daquela chamada (mediana de um conjunto de n valores é o valor que possui n/2 valores menores que ele e n/2 valores maiores que ele no conjunto). Este procedimento será chamado de `Med`. O cálculo da mediana de um conjunto de n elementos é feita em tempo O(n).

   b) divide os números em menores que m (conjunto A), e números maiores ou iguais a m (conjunto B)

   c) chama recursivamente o algoritmo `NewSort`() duas vezes, uma para o conjunto A e outra para conjunto B

   d) executa um merge dos dados: o `Merge()` junta o resultado das duas chamadas de forma a manter os dados ordenados.

   Se você tivesse que escolher um algoritmo de ordenação, sem informação prévia sobre quais serão os dados de entrada, qual você escolheria entre `NewSort` e `Quicksort`? Porquê? E entre `NewSort` e `MergeSort`? Porquê?

Algoritmo 0.2 (NewSort)
Entrada Uma seqüência $s_1, \ldots, s_n$ de números inteiros, $n \geq 1$.

```
if (n = 1) then
  return s_1
end if
m = Med(s)
s = PartitionNewSort(s, m)
v_1 := DC(s_1, ..., s_⌊n/2⌋)
v_2 := DC(s_⌊n/2⌋+1, s_n)
return Merge(v_1, v_2)
```

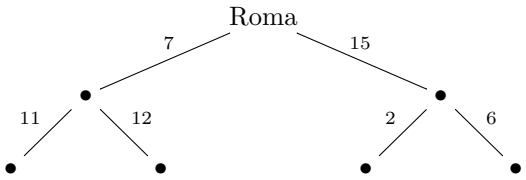
O procedimento `PartitionNewSort(s,m)` organiza o vetor v colocando na primeira metade de v os valores menores que m, e na segunda metade do vetor os valores maiores que m. O procedimento `Merge`(v_1, v_2) faz uma união ordenada dos valores de v_1 e v_2. Ambos procedimentos possuem complexidade O(n). A equação de recorrência do algoritmo é $T(n) = 2T(\frac{n}{2}) + n$. Esta equação de recorrência é igual a equação do mergesorte, ou seja, a complexidade do algoritmo será $O(n \cdot$

logn). Se for comparar em termos de complexidade pessimista, o procedimento é comparável ao Mergesort, com a desvantagem que $f(n)$ neste caso é representado por $(2n)$, equanto que no mergesort é apenas n (do Merge). Se na prática o QuickSort em geral é mais rápido, seria mais rápido que este procedimento também.

3. (NewSort2) Resolva a questão anterior, substituindo o algoritmo `Med` por um algoritmo que calcula a média aritmética dos valores.

   A média dos elementos não vai garantir uma divisão igual dos elementos. Desta forma, o valor médio funcionaria da mesma forma que o pivô funciona para o quicksort, com o custo adicional de calcular a média. A complexidade seria a mesma do quicksort.

4. (Todos os caminhos levam à Roma) Viajando de Roma para o litoral, descobri que existem várias rotas alternativas com distâncias diferentes que levam para cidades diferentes no litoral. Como a cidade concreta não importa para mim, mas o comprimento do caminho sim, eu gostaria de saber o caminho mais curto de Roma para alguma cidade no litoral. O seguinte exemplo mostra tal situação



   com as folhas da árvore sendo as cidades no litoral.

   (a) Qual o caminho mais curto nesse caso?

   (b) Projete um algoritmo, que resolva o problema para árvores binárias completas arbitrárias (enunciado abaixo):

> Caminho mais curto em árvores
>
> **Instância** Uma árvore binária completa, com pesos nos links.
>
> **Solução** O comprimento do caminho mais curto da raiz para alguma folha.

   Observação: existe um algoritmo em tempo linear para resolver este problema.

   (c) Justifique a corretude do algoritmo e analise a complexidade dele.

Seja

$$\text{Bintree} := \text{Node}(l, r, \text{BinTree}, \text{BinTree}) | \text{Leaf}$$

uma representação de uma árvore binária.

(a) O caminho mais curto é (direita,esquerda) com com comprimento 17.

(b) Usando a recorrência

$$D(T) = \begin{cases} \min\{l + D(T_l), r + D(T_r)\} & \text{caso } T = \text{Node}(l, r, T_l, T_r) \\ 0 & \text{caso } T = \text{Leaf} \end{cases}$$

obtemos um algoritmo de divisão e conquista. Exemplo de uma implementação

```
CMRoma(T):=
  case
    T ≡ Bintree(l, r, T_l, T_r):
      return min(l + CMRoma(T_l), r + CMRoma(T_r))
    T ≡ Leaf:
      return 0
  end case
```

(c) O algoritmo é correto, porque o caminho mais curto tem um subcaminho mais curto na árvore esquerda ou direita. A complexidade dele é dado por

$$T(n) = 2T(n/2) + O(1) = O(n).$$

5. (2,5 pontos) Dado um vetor com n números construa um algoritmo de Divisão e Conquista que retorne a posição i em que um dado elemento x ocupa. Suponha que x ocorra no máximo uma vez no vetor.
   a) Escreva o pseudocódigo do algoritmo.

Algoritmo 0.3 (Find(A,i,f))
Entrada Um vetor $s_1, \ldots, s_n$ de números, o índice de início do vetor e o índice de fim do vetor.

```
if (f == i) then
  if (A[i]=x)
  then return i
  else
  m = ⌈(f−i)/2) + i⌉
  v1 := DC(A, i, m − 1)
  v2 := DC(A, m, f)
  end if
end if
```

   b) Qual a equação de recorrência do seu algoritmo? $T(n) = 2 \cdot T(\frac{n}{2}) + 1$

   c) Qual a complexidade do seu algoritmo? (não precisa detalhar o cálculo)

   $\theta(n)$

6. Para as equações de recorrência abaixo, apresente sua solução.
   a) $T(n) = 2 \cdot T(\frac{n}{2}) + n \cdot \log n$
   b) $T(n) = 9 \cdot T(\frac{n}{3}) + n$