Lista de Exercícios 1

Pergunta 1. Considere o pseudocódigo a seguir

```
procedure PROC(n)
    if n = 1 then
        return
    for i = 1..n − 1 do
        for j = i + 1..n do
            t ← 0
    PROC(n − 1)
```

a) Seja $T(n)$ a complexidade de pior caso do procedimento acima. Ache uma equação de recorrência para $T(n)$.

b) Encontre uma função $f(n)$ tal que $T(n) = \Theta(f(n))$

Pergunta 2. Considere os pseudo-códigos abaixo.

a) Determine para o pseudo código 1 uma função $f(n)$ tal que $T(n) = \Theta(f(n))$

```
procedure PSEUDO1(n)
    t ← 0
    Cont ← 1
    for i = 1..n do
        Cont ← Cont + 1
    while Cont ≥ 1 do
        Cont ← Cont/2
        for j = 1..n do
            t ← t + 1
```

b) Determine para o pseudo código 2 uma função $g(n)$ tal que $T(n) = \Theta(g(n))$

```
procedure PSEUDO2(n)
    i ← 0
    while i^2 ≤ n do
        i ← i + 1
        t ← 0
        while t ≤ i do
            t ← t + 1
```

Pergunta 3. Seja $S = \{a_1, \ldots, a_n\}$ um conjunto de n números reais distintos. Considere o problema $\mathcal{P}$ de determinar se existem três números distintos em S cuuja soma é 0.

a) Seja $T(n)$ a complexidade de pior caso do algoritmo abaixo para resolver $\mathcal{P}$. Encontre $f(n)$ tal que $T(n) = \Theta(f(n))$.

```
procedure P-SOLVE(S)
    for i = 1..n − 2 do
        for j = i + 1..n − 1 do
            for k = j + 1..n do
                if a_i + a_j + a_k = 0 then
                    return SIM
```

b) Projete um algoritmo mais eficiente do que o algoritmo acima em termos de complexidade assintótica. Não é necessário apresentar o pseudo-código, mas sim explicar com clareza os passos que o algoritmo deve realizar e explicar a complexidade. Quanto mais eficiente o algoritmo melhor.

Pergunta 4. Seja $T(n)$ a complexidade de pior caso de um algoritmo para entradas de tamanho n.

a) Assuma que $T(n) = O(n^2)$. Podemos afirmar que para n suficientemente grande existe uma entrada de tamanho n para a qual o algoritmo realiza pelo menos $100n$ operações? Por que?

b) Assuma que $T(n) = \Omega(n^2)$. Podemos afirmar que para toda entrada de tamanho n o algoritmo vai realizar pelo menos $\sqrt{n}$ operações? Por que?

Pergunta 5. Considere o problema $\mathcal{Q}$ definido da seguinte forma: *Entrada* Inteiro $N \geq 3$; *Saída*: `SIM` se N é composto; `NÃO`, caso contrário.

a) Qual *tamanho da entrada* deste problema em função de N.

b) Determine a função complexidade de tempo do algoritmo abaixo para resolver $\mathcal{Q}$. Este algoritmo é polinomial? Por que? **Assuma que a checagem de N ser múltiplo de I é $O(1)$**

```
procedure Q-SOLVE(N)
    I ← 2
    while I^2 ≤ N do
        if N é múltiplo de I then
            return SIM, N é composto
        I ← I + 1
    return Não, N é primo
```

Pergunta 6. Seja S um conjunto de n elementos, onde cada elemento $s \in S$ tem uma prioridade $p(s)$ pertencente ao conjunto dos U menores inteiros positivos. Considere o seguinte procedimento

procedure PROC(S)
  for $i = 1..n^2$ **do**
    $x \leftarrow Rand(1, U)$
    Modifique a propriedade do elemento de S com a menor prioridade para x (caso haja mais de um elemento com a prioridade mínima, qualquer um pode ser escolhido)

Assuma que a função $Rand(1, U)$ retorna em tempo constante um inteiro aleatório entre 1 e U.

a) Analise a complexidade do algoritmo acima quando S esta armazenado como um heap binário de mínimo. Note que após cada mudança de prioridade devemos restaurar a ordenação do heap.

b) Assuma agora que $1 \leq U \leq 5$. como poderíamos armazenar S de modo a melhorar a a complexidade do procedimento? Explique como seria o procedimento para modificar a prioridade do menor elemento para a nova estrutura de armazenamento proposta. Qual seria a complexidade obtida?

Pergunta 7. (Search in a Rotated Array) Given a sorted array of n integers that has been rotated an unknown number of times, write code to find an element in the array. You may assume that the array aws originally sorted in increasing order.

Example:
Input: find 5 in $\{15, 16, 19, 20, 25, 1, 3, 4, 5, 7, 10, 14\}$
Output: 8 (the index of 5 in the array)

Assuma que a lista não contenha números repetidos. Obtenha um algoritmo com complexidade assintótica estritamente melhor que $O(n)$ (e.g, $O((\log n)^2)$ ou $O(\log n)$).

Dica: Note que uma das metades do vetor está ordenado.

Pergunta 8. (Smallest Difference) Given two arrays of integers, compute the pair of values (one value in each array) with the smallest (non-negative) difference. Return the difference.

Example:
Input: $\{1, 3, 15, 11, 2\}, \{23, 127, 235, 19, 8\}$
Output: 3. That is, the pair (11,8)

a) Resolva esse problema em tempo $O(n \log n)$ (onde os dois vetores tem tamanho no máximo n)

b) Assuma que todos os números sejam inteiros entre 1 e $30n$. Resolva o problema em tempo $O(n)$.

Pergunta 9. Given two sorted lists of numbers of length m, n. Give an algorithm that finds the kth smallest number in the union of the lists, in time $O(\log m + \log n)$. (You can assume that all numbers in the input are distinct).

Dica: Em cada iteração você pode descartar a metade de uma das listas. Imagina a posição dos elementos do meio das listas na união ordenada das listas.

Pergunta 10.

2. (2.5 pt) Considere o seguinte algoritmo que dado uma lista A com n números e um número x, verifica se em algum momento o número x aparece na lista e o número $x+1$ já apareceu "recentemente":

```
Algo1(A : lista com n números inteiros, x : número inteiro)
1:     For i = 1 to n
2:         If A[i] = x
3:             For j = 1 to √i
4:                 If A[j] = x + 1
5:                     Return SIM
6:                 End if
7:             End for
8:         End if
9:     End for
10:    Return NAO
11: End algorithm
```

Seja $T(n)$ a complexidade de pior-caso desse algoritmo. Encontre a função $f(n)$ tal que $T(n) = \Theta(f(n))$ e:

(a) (1.25 pts) Justifique a parte $O(f(n))$

(b) (1.25 pts) Justifique a parte $\Omega(f(n))$.