1. (2.0) Considere o pseudo-código a seguir

```
Proc(n)
    Se n =1
        Return
    Fim Se
    Para i:=1 até n-1
        Para j:=i+1 até n
            t ← 0
        Fim para
    Fim para
    Proc(n-1)
Fim Proc
```

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 $T(n) = \Theta(f(n))$.

4. (2.0pt) Considere os pseudo-códigos abaixo.

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

Pseudo1

```
t ← 0
Cont ← 1
Para  i=1 até n
    Cont ← cont+1
Fim Para
Enquanto cont ≥ 1
    Cont ← cont/2
    Para j = 1 a n
        t++
    Fim Para
Fim Enquanto
```

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

Pseudo2

```
i ← 0
Enquanto i² ≤ n
    i ++
    t ← 0
    Enquanto t ≤ i
        t++
    Fim Enquanto
Fim Enquanto
```

3. (2.5pt) 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 cuja soma é 0

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

Para i=1,...,n-2
  Para j=i+1,...,n-1
    Para k=j+1,...,n
      Se $a_i+a_j+a_k$ =0
        Return SIM
Return NÃO

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 maior a pontuação.

1. (1 pt) Seja $T(n)$ a complexidade de pior caso de um algoritmo para entradas de tamanho n.

   (a) (0.5 pt) Assuma que $T(n) = O(n^2)$. Podemos afirmar que para n sufficientemente grande existe uma entrada de tamanho n para a qual o algoritmo realiza pelo menos $100n$ operações? Por que?

   (b) (0.5 pt) 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?

4 (2.0pt). 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 é o *tamanho da entrada* deste problema em função de N.

b) Determine a função complexidade de tempo do algoritmo acima para resolver $\mathcal{Q}$. Este algoritmo é polinomial ? Por que? (Assuma que cada teste "N é multiplo de I" é O(1))

$I \leftarrow 2$
Enquanto $I * I \leq N$ faça
  If N é múltiplo de I **then**
    Return SIM, N é composto
  $I \leftarrow I + 1$
Return NÃO, N é primo

3. (2.0pt) 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

Para i=1,...,n^2
$x \leftarrow Rand(1, U)$
Modifique a prioridade do elemento de S com menor prioridade para x (*)
Fim Para

Assuma que a função $RAND(1, U)$ retorna em tempo constante um inteiro aleatório entre 1 e U. Além disso, caso haja mais de um elemento com prioridade mínima na linha (*), qualquer um pode ser escolhido.

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 poderiamos armazenar S de modo a melhorar 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?

1)

Search in 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 was 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 nao contenha numeros repetidos. Obtenha um algoritmo com complexidade assintotica estritamente melhor que O(n) (e.g., O((log n)^2) ou O(log n)). Analise seu algoritmo.

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

Solucao:

2)

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 maximo n)

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

Solucao:

Given two sorted lists of numbers of length m, n. Give an algorithm that finds the k'th 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 iteracao voce pode descartar a metade de uma das listas. Imagine a posicao dos elementos do meio das listas na uniao ordenada das listas.

Solucao: