Guilherme S. Borges - 26614

12° Lista de Exercícios

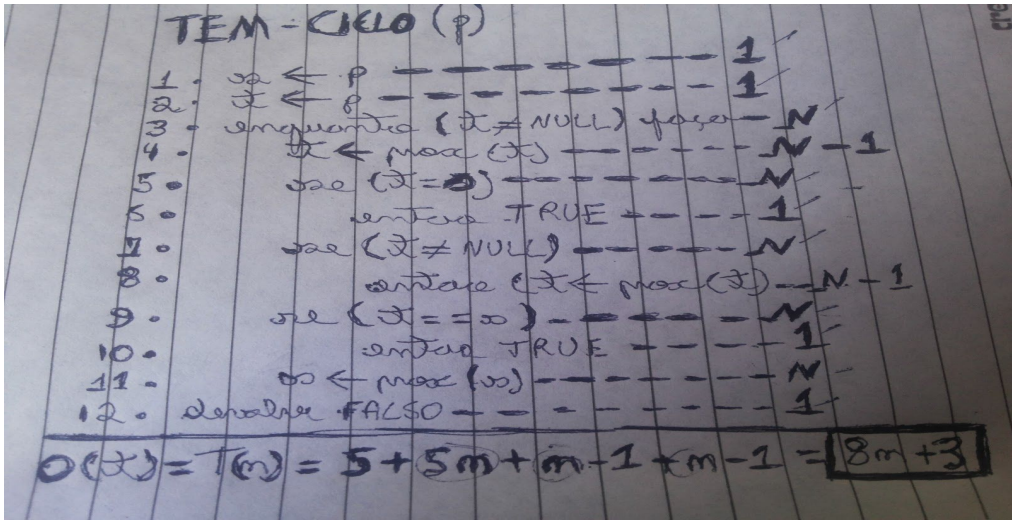
1) O algoritmo abaixo recebe o início de uma lista ligada e devolve VERDADEIRO se a lista ligada possui um ciclo ou FALSO caso contrário:

```
Tem-Ciclo( p )
 1. s ← p
 2. t ← p
 3. enquanto t ≠ NIL faça
 4.       t ← prox( t )
 5.       se  t = s
 6.           então devolve VERDADEIRO
 7.       se  t ≠ NIL
 8.           então t ← prox(t)
 9.       se  t = s
10.           então devolve VERDADEIRO
11.       s ← prox( s )
12. devolve FALSO
```

a) Explique porque o algoritmo está correto:

Porquê na linha 5 se logo na primeira iteração com **t == s** ele já retorna VERDADEIRO para existência do ciclo, assim como para o caso FALSO, em que ele caminha pelo laço avançando **t para prox(t)**, e novamente é testado o ciclo; logo, o algoritmo possui condição de parada, retorna uma resposta e ainda percorre toda a estrutura passada, logo está correto.

b) Denote por n o número de elementos na lista apontada por p. Analise a complexidade do algoritmo, indicando seu comportamento assintótico. Justifique suas respostas:

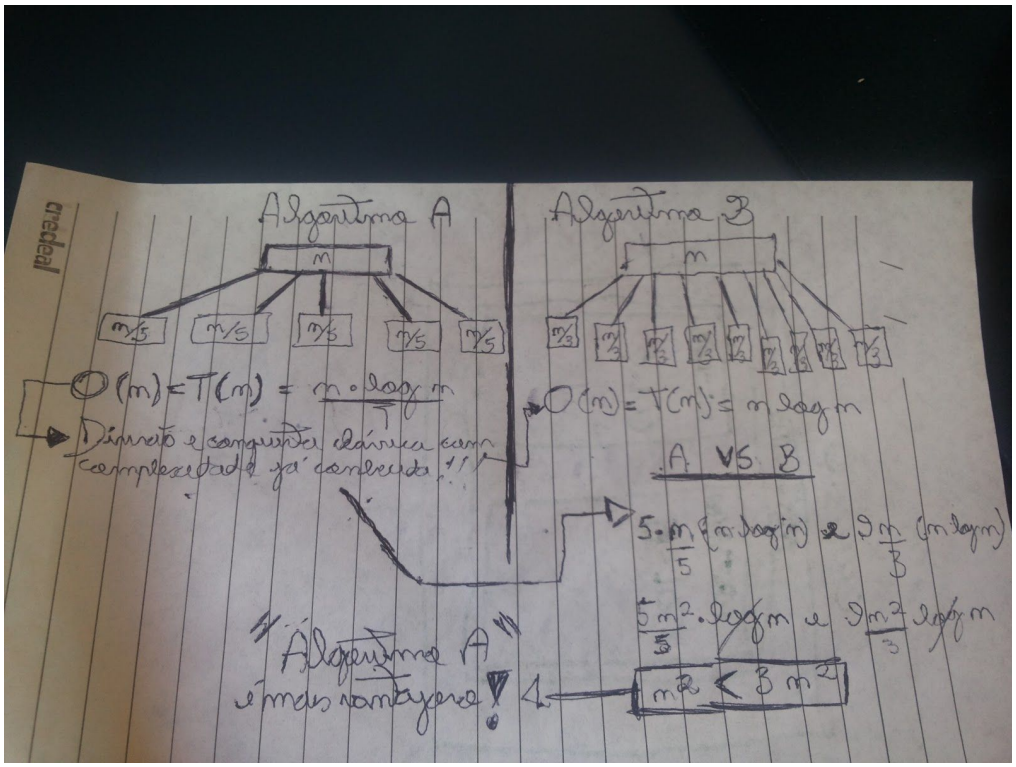


2) Suponha que, para entradas de tamanho n, você tenha que escolher entre os algoritmos A e B: *

- **Algoritmo A resolve problemas dividindo-os em cinco problemas de metade do tamanho, recursivamente resolve cada subproblema e então combina as soluções em tempo O(n).**
- **Algoritmo B resolve problemas dividindo-os em nove subproblemas de tamanho n/3, recursivamente resolve cada subproblema e então combina as soluções em tempo O(n2).**

Estime o consumo de tempo de cada um desses algoritmos? Qual algoritmo é assintoticamente mais eficiente no pior caso? Justifique suas respostas.

Conforme veremos na imagem abaixo, **o Algoritmo A é o mais eficiente no pior caso** (segue justificação na figura):



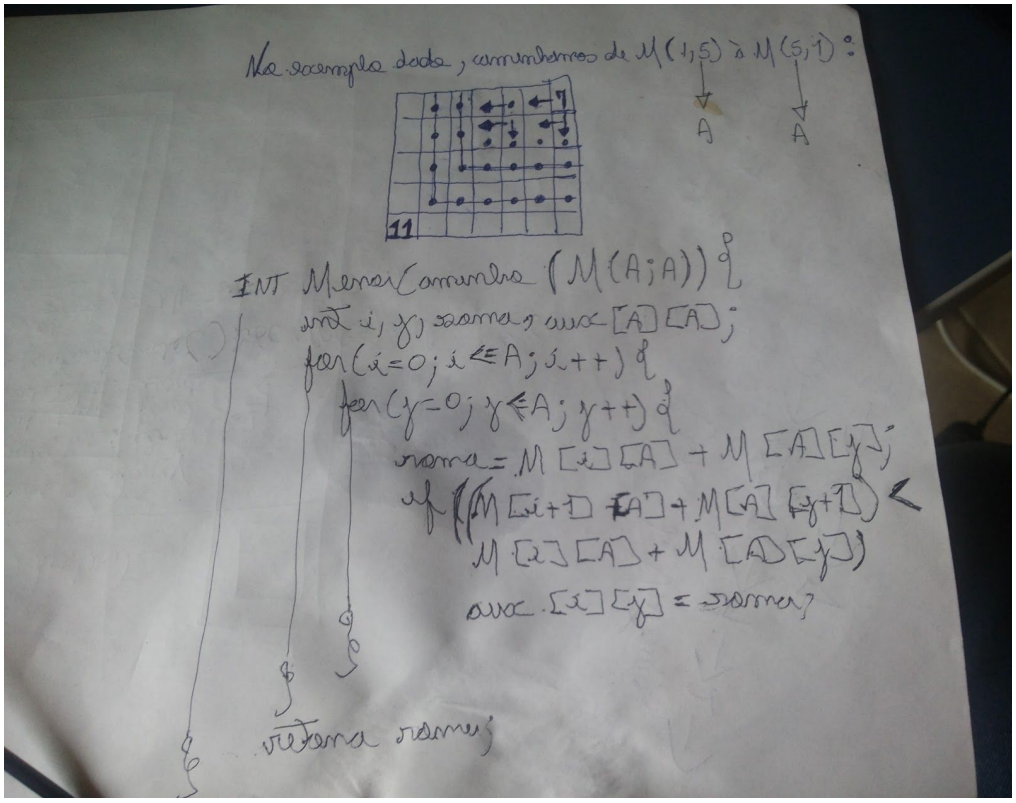
3) Seja A[1..n] um vetor de inteiros. Um segmento A[i..k] (1 i k n) é não decrescente se A[i] A[i+1] . ... A[k-1] A[k]. O comprimento de um tal segmento é k-i+1. Escreva um algoritmo que recebe um vetor A[1..n] e devolve o comprimento do segmento não decrescente de comprimento máximo de A. Por exemplo, se A = (1, 4, 8, 9, 2, 3, 4, 5, 6, 8, 9, 1, 2, 3) então o segmento não decrescente máximo é M = (2, 3, 4, 5, 6, 8, 9) com comprimento 7:

```
P = (1, 4, 8, 8, 2, 3, 4, 5, 6, 8, 9, 1, 2, 3)
M = (2, 3, 4, 5, 6, 8, 9,)
L = 0 //Variável auxiliar
para (i = P.tam()) até 0 faça i--:
  // Busca binária pelo menor negativo no intervalo j ≤ L
  // A ideia é percorrer X[M[j]] < P[i]
  menor = 1
  maior = L
  enquanto menor ≤ maior:
    meio = arredondaDiv((menor+maior)/2)
    se P[M[meio]] < M[i]:
      menor = meio+1
    senao
      maior = meio-1

  //Após a busca, "menor" é uma unidade maior que X[maior]
  novoL = menor
  P[i] = M[novoL-1]
  M[novoL] = i

  se novoL > L:
    L = novoL

 // Reconstrói a maior subsequência decrescente
S = array[L]
k = M[L]
para i; L--; até 0:
  S[i] = P[k]
  k = P[k]

retorna S;
```

4) Escreva um algoritmo que recebe uma matriz M(n x n) de inteiros e encontra um caminho de M(1,n) até M(n,1) de forma a minimizar a soma dos valores absolutos das diferenças entre elementos consecutivos do caminho. Um caminho é definido por movimentos horizontais ou verticais entre elementos adjacentes da matriz, por exemplo, para a matriz abaixo, o custo mínimo é 66:

1	**17**	**6**	46	**7**
4	58	**15**	**6**	**10**
2	8	61	18	29
9	18	6	23	9
11	6	13	41	12

Para testar a implementação do algoritmo parti da idéia base do Dijkstra, porém alterando a forma de realizar o percurso, já que procuramos o caminho com custo mínimo da matriz vista acima:



5) Projete um algoritmo para construir um heap que contém todos os elementos de dois heaps de tamanho n e m, respectivamente. Os heaps são dados em listas ligadas. No pior caso seu algoritmo deverá rodar em ordem O(lg(m+n)).

```
void maxHeapBusca(int lista[], int n, int index)
{//Usamos a busca na fusão feita mais abaixo
    if (index >= n)
        return;
    int esq = 2 * index + 1;
    int dir = 2 * index + 2;
    int max;
    if (esq < n && lista[esq] > lista[index])
        max = esq;
    else
        max = index;
    if (dir < n && lista[dir] > lista[max])
        max = dir;
    if (max != index) {
        troca(lista[max], lista[index]);
        maxHeapBusca(lista, n, max);
    }
}
void ConstroiMaxHeap(int lista[], int n)
{//Função chamada na hora de fazer o merge
    for (int i = n / 2 - 1; i >= 0; i--)
        maxHeapBusca(lista, n, i);
}
//Combina heap a[N] com a heap b[M] em uniao[]
void FundeHeaps(int uniao,[], int a[], int b[],
                int n, int m)
{
    // Copia a[] em b[] um por um:
    for (int i = 0; i < n; i++)
        uniao[i] = a[i];
```

```
    for (int i = 0; i < m; i++)
        uniao[n + i] = b[i];
    //Constrói a heap de tamanho (n + m)
    ConstroiMaxHeap(merged, n + m);
}
//Testa implementação
int main()
{
    int a[] = { 10, 20, 30, 40 };
    int b[] = { 50, 60, 70, 80 };
    //Captura o tamanho (tam) dos arrays
    int n = sizeof(a) / sizeof(a[0]);
    int m = sizeof(b) / sizeof(b[0]);
    int tam[m + n];
    FundeHeaps(tam, a, b, n, m);
    for (int i = 0; i < n + m; i++)
        cout << uniao[i] << " ";
    return 0;
}
```