**Segundo teste de Algoritmos e Estruturas de Dados**

**21 de Dezembro de 2020** **09h45m – 11h00m**

**Responda a todas as perguntas no enunciado do teste. Justifique todas as suas respostas.**

**Nome:** André Almeida Oliveira

**N. Mec.:** 107637

**Fórmulas:**

- $\sum_{k=1}^{n} 1 = n$
- $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$
- $\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$
- $\sum_{k=1}^{n} k^3 = \left(\frac{n(n+1)}{2}\right)^2$
- $\sum_{k=1}^{n} \frac{1}{k} \approx \log n$
- $n! \approx n^n e^{-n}\sqrt{2\pi n}$

**2.5** (5 min) **1:** Qual a complexidade computacional da seguinte função?

```
double f(int n)
{
  if(n < 2)
    return 1.0;
  else
    return (double)n * f(n - 2);
}
```

Resposta: $O\left(\left\lfloor \frac{n}{2} \right\rfloor + 1\right) = O(n)$

**2.5** (5 min) **2:** Ordene as seguintes funções por ordem crescente de ritmo de crescimento. Responda nas duas colunas da direita da tabela. Na coluna da ordem, coloque o número 1 na função com o ritmo de crescimento menor (e, obviamente, coloque o número 5 na com o ritmo de crescimento maior).

| função | termo dominante | ordem |
|---|---|---|
| $n^2 \log n^2$ | $n^2 \log n$ | 3 |
| $\sum_{k=1}^{n} \left(k + \frac{1}{k}\right)$ | $\frac{n(n+1)}{2}$ | 2 |
| $10^{100} n^{42} + 1.001^n$ | $1{,}001^n$ | 5 |
| $n^3 + 0.999^{n/2}$ | $n^3$ | 4 |
| $\frac{10^n}{n!}$ | $\frac{10^n}{n!}$ | 1 |

**3.0** (10 min) **3:** Pretende-se que a seguinte função implemente uma pesquisa binária. Complete-a (isto é, preencha as caixas).

```
int binary_search(int n,int a[n],int v)
{
  int low = [ 0 ];
  int high = [ n-1 ];
  while(1)
  {
    if(low [ > ] high)
      return [ -1 ];
    int middle = [ (low + high) / 2 ];
    if(a[middle] == v)
      return middle;
    if(a[middle] [ < ] v)
      [ low ] = middle + 1;
    else
      [ high ] = middle - 1;
  }
}
```

**3.0** (10 min) **4:** Explique como está organizado um *min-heap*. Para o *min-heap* apresentado a seguir, insira primeiro número 3 e depois o número 6. Não apresente apenas o resultado final; mostre, passo a passo, o que acontece ao *array* durante a inserção. Em cada linha, basta escrever as entradas do *array* que foram alteradas.

Respostas:

| 0 | 4 | 1 | 5 | 8 | 2 | 9 | 7 | 3 | |
|---|---|---|---|---|---|---|---|---|---|
| | | | 3 | | | | | 5 | |
| | 3 | | 4 | | | | | | |
| 0 | 3 | 1 | 4 | 8 | 2 | 9 | 7 | 5 | 6 |
| | | | | 6 | | | | | 8 |
| 0 | 3 | 1 | 4 | 6 | 2 | 9 | 7 | 5 | 8 |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] |

**3.0** (10 min) **5:** Explique como é que funciona o *insertion sort*. Indique qual é a sua complexidade computacional e indique qual é o seu melhor e pior caso (maior e menor tempo a ordenar).

Resposta:

Insertion sort é um algoritmo de ordenação que divide um conjunto de dados em dois subconjuntos, ou seja, um ordenado e outro desordenado. A cada iteração, o algoritmo faça no próximo elemento do subconjunto desordenado e insere-o na sua posição correta do subconjunto ordenado. Isto é feito comparando o elemento com os já ordenados, deslocando-os para a direita ou esquerda (depende do caso) se necessário.

Pior caso : $O(n^2)$ -> array for ordem inversa
Melhor caso : $O(n)$ -> array já ordenado

**3.0** (10 min) **6:** Um programador inexperiente escreveu a seguinte *hash function*:

```
int hash_function(int n,char data[n],int hash_table_size)
{
  int sum = 0;
  for(int i = 0;i < n;i++)
    sum += (int)data[i];
  int idx = sum % hash_table_size;
  if(idx < 0)
    idx = -idx;
  return idx; // 0 <= idx < hash_table_size
}
```

Que problemas tem esta *hash function*?

Resposta:

Esta função retorna uma gama de valores muito baixa, logo as keys não são distribuídas uniformemente pela hash table. Além disso, esta função não é capaz de distinguir qualquer anagrama.

**3.0** (10 min) **7:** Apresentam-se a seguir várias funções (`f1` a `f5`) que visitam todos os nós de uma árvore binária, e mostram-se várias ordens pelas quais a função `visit` foi chamada para cada um dos nós (1 significa que o nó correspondente foi o primeiro a chamar a função `visit`, 2 que foi o segundo, e assim por diante). Para cada uma das ordens apresentadas, indique que função, ou funções, deram origem a essa ordem.

```
void f1(tree_node *n)
{
  stack *s = new_stack();
  push(s,n);
  while(is_empty(s) == 0)
  {
    n = pop(s);
    if(n != NULL)
    {
      push(s,n->left);
      push(s,n->right);
      visit(n);
    }
  }
  free_stack(s);
}
```

```
void f2(tree_node *n)
{
  queue *q = new_queue();
  enqueue(q,n);
  while(is_empty(q) == 0)
  {
    n = dequeue(q);
    if(n != NULL)
    {
      enqueue(q,n->right);
      visit(n);
      enqueue(q,n->left);
    }
  }
  free_queue(q);
}
```

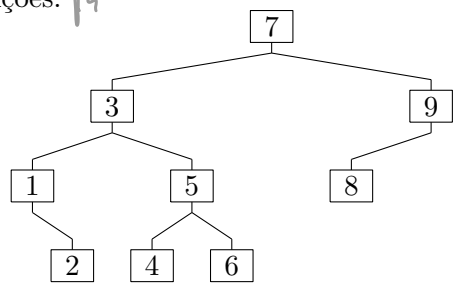
```
int cnt = 0;

void visit(tree_node *n)
{
  printf("%d\n",++cnt);
}
```

```
void f3(tree_node *n)
{
  if(n != NULL)
  {
    visit(n);
    f3(n->right);
    f3(n->left);
  }
}
```

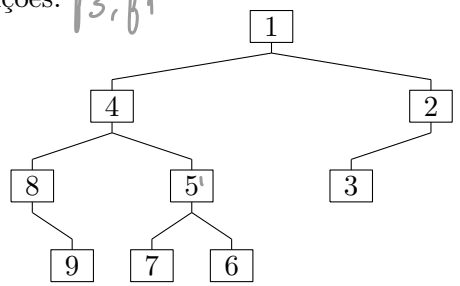
```
void f4(tree_node *n)
{
  if(n != NULL)
  {
    f4(n->left);
    visit(n);
    f4(n->right);
  }
}
```

```
void f5(tree_node *n)
{
  if(n != NULL)
  {
    f5(n->left);
    f5(n->right);
    visit(n);
  }
}
```

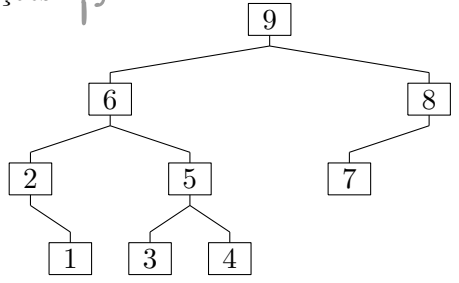
Funções: 4



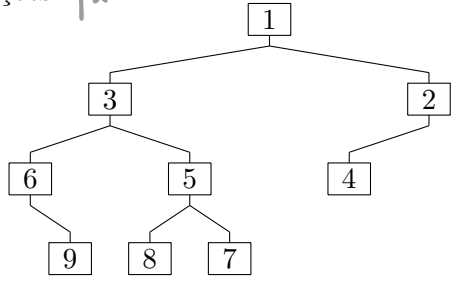

Funções: 3, f1




Funções: 5




Funções: 2



**Terceiro teste de Algoritmos e Estruturas de Dados**

**18 de Janeiro de 2021** **10h00m – 11h00m**

**Responda a todas as perguntas no enunciado do teste. Justifique todas as suas respostas.**
**O teste é composto por 5 grupos de perguntas.**

---

**Nome:** André Almeida Oliveira

**N. Mec.:** 107637

**4.0** **1:** Um aluno descobriu uma maneira inovadora de multiplicar números grandes que consiste no seguinte:

- cada um dos dois números a multiplicar, com $3n$ algarismos cada, é dividido em **3** partes, cada uma com $n$ algarismos
- usando somas e subtrações, a partir dessas partes são calculados **3** números, com $n$ algarismos cada
- usando as partes dos números e os **3** números extra do ponto anterior, são calculados **7** produtos (de números de $n$ algarismos)
- finalmente, são efetuadas **10** somas dos produtos calculados no ponto anterior para se obter o resultado final.

Responda às seguintes perguntas:

1.0 **a)** Que estratégia algoritmica usou o aluno? Divide and Conquer

3.0 **b)** Qual é a complexidade computacional do algoritmo inventado pelo aluno?

Respostas:

b) a = 7 → cálculo de 7 produtos
b = 3 → dividir cada número em 3 partes

$f(n) = O(n^{\log_7 3 - \epsilon})$ → $\log_7 3 > 1$

$T(n) = \Theta(n^{\log_7 3})$

O *master theorem* afirma que se $T(n) = aT(n/b) + f(n)$ então

- se $f(n) = O(n^{\log_b a - \epsilon})$ para um $\epsilon > 0$, então $T(n) = \Theta(n^{\log_b a})$,
- se $f(n) = \Theta(n^{\log_b a})$, então $T(n) = O(n^{\log_b a} \log n)$,
- se $f(n) = \Omega(n^{\log_b a + \epsilon})$ para um $\epsilon > 0$ e se $af(\frac{n}{b}) \leq cf(n)$ para $c < 1$ e $n$ suficientemente grande, então $T(n) = \Theta(f(n))$.

**5.0** **2:** Num cubo de dimensões $N \times N \times N$ pretende-se ir do ponto com coordenadas $(0, 0, 0)$ até ao ponto com coordenadas $(N-1, N-1, N-1)$ usando movimentos que apenas aumentem em uma unidade uma das coordenadas. Logo, a partir de $(x, y, z)$ podemos apenas ir para $(x+1, y, z)$, $(x, y+1, z)$ e $(x, y, z+1)$. Mais concretamente, pretende-se contar o número de maneiras de fazer isso. O código seguinte faz isso.

```
#define N 100
[ int ] count[N][N][N];

int do_count(int x,int y,int z)
{ // (x,y,z) are the coordinates of the DESTINATION
  if(x < 0 || x >= N || y < 0 || y >= N || z < 0 || z >= N)
    return 0;
  if(count[x][y][z] [ == -1 ])
    count[x][y][z] = do_count( [ x-1, y, z ] ) +
                     do_count( [ x, y-1, z ] ) +
                     do_count( [ x, y, z-1 ] );
  return count[x][y][z];
}

int count_all_paths(void)
{
  for(int x = 0;x < N;x++)
    for(int y = 0;y < N;y++)
      for(int z = 0;z < N;z++)
        count[x][y][z] = [ -1 ];
  count[0][0][0] = [ 1 ];
  return do_count( [ N-1, N-1, N-1 ] );
}
```

Responda às seguintes perguntas:

3.0 **b)** Complete o código.

1.0 **b)** Que estratégia algoritmica está a ser usada? Programação dinâmica

1.0 **c)** Qual é a complexidade computacional do algoritmo?

$O(n^3)$, pois no máximo percorremos a tabela toda do array multidimensional count para inicializar toda a matriz

**4.0** **3:** Os **5** vértices, numerados de **0** a **4**, de um grafo têm as seguintes listas de adjacência:

| vértice | lista |
|---|---|
| **0** | → **(1,** fast**)** → **(3,** fast**)** → **(1,** slow**)** → **NULL** |
| **1** | → **(2,** slow**)** → **(3,** fast**)** → **NULL** |
| **2** | → **(0,** fast**)** → **NULL** |
| **3** | → **(1,** slow**)** → **NULL** |
| **4** | → **(0,** fast**)** → **(3,** slow**)** → **NULL** |

Responda às seguintes perguntas:

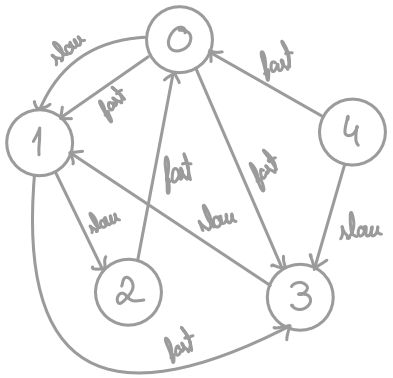
1.2 **a)** Desenhe o grafo.

1.2 **b)** Este grafo é de que tipo? Multigrafo

1.6 **c)** Será possível representar o grafo usando uma matriz de adjacência? Se sim, como? Se não, porque não?

Respostas:

a)




c) Sim, associando o valor 0 a não existir aresta, o valor 1 a fast, o valor 2 a slow e o valor 3 a fast e slow

**3.0** **4:** Explique para que serve e como funciona o algoritmo *union find*.

O algoritmo union find pode encontrar a qual conjunto um certo elemento pertence e unir conjuntos existentes. Este algoritmo é bastante utilizado em problemas relacionados com grafos, como por exemplo, unir dois componentes conexos.

**4.0** **5:** Considere um labirinto desenhado na superfície de uma esfera. Pretende-se ir, a andar, do pólo norte até ao pólo sul. (Considere que a esfera tem um raio relativamente pequeno, pelo que ir a pé não demora meses, mas pode demorar dias.) Responda às seguintes perguntas:

1.0 **a)** Que algoritmo usaria para encontrar uma solução?

1.5 **b)** Que material levaria consigo para o ajudar?

1.5 **c)** Acha que é possível encontrar a solução mais curta de uma forma eficiente? (Não se esqueça que tem de fazer todo o caminho a pé.) Porquê?

Answers:

a) Depth first search

b) Algum objeto para marcar a sitio por onde já tenha passado

c) Não, a única maneira "mais eficiente" neste caso seria usar breath first search, mas teriamos de percorrer uma distância muito grande. Se já conhecessemos o labirinto na totalidade, poderiamos usar o algoritmo dijkstra calculando o caminho mais curto