Primeira parte do teste final de Algoritmos e Estruturas de Dados

4 de fevereiro de 2022 **16h00m – 16h55m**

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

Nome: _____

N. Mec.: _____

Fórmulas:

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

4.0 **1:** No seguinte código,

```
#include <stdio.h>

int f(int x) { return x * x - 1; }
int g(int x) { return x / 3; }

int main(void)
{
  int c = 0;
  for(int i = 0;i <= 10;i++)
    if( f(i) && g(i) )
    {
      printf("i = %d\n",i);
      c += g(i);
    }
  printf("c = %d\n",c);
}
```

1.0 **a)** para que valores da variável `i` é avaliada a função `g(x)`?

1.0 **b)** que valores de `i` são impressos?

1.0 **c)** que valor de `c` é impresso?

1.0 **d)** neste caso concreto, considerando que um `int` tem 32 *bits*, existe a possibilidade de *overflow* aritmético ao correr o programa?

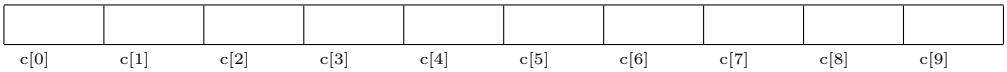
4.0 **2:** Um programador inexperiente escreveu a seguinte função para copiar uma zona de memória com `size` bytes que começa no endereço `src` para uma outra zona de memória que começa no endereço `dst`.

```
void mem_copy(char *src,char *dst,size_t size)
{
  for(size_t i = 0;i < size;i++)
    dst[i] = src[i];
}
```

Responda às seguintes perguntas, considerando que para cada uma das duas primeiras o conteúdo **inicial** do array `c` é `char c[10] = { 0,1,2,3,4,5,6,7,8,9 };`

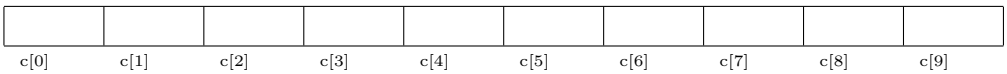
1.3 **a)** Qual o conteúdo do array `c` depois de `mem_copy(&c[4],&c[5],4);` ter sido executado?

Resposta:



1.3 **b)** Qual o conteúdo do array `c` depois de `mem_copy(&c[5],&c[4],4);` ter sido executado?

Resposta:



1.4 **c)** Num dos casos anteriores a cópia do conteúdo de parte do array não foi feita corretamente; sugira uma maneira de corrigir este problema (não é obrigatório escrever código).

Resposta:

4.0 **3:** 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). Na coluna do termo dominante indique, usando a notação *Big Oh*, qual é o termo dominante; por exemplo, se na primeira coluna estivesse $\mathbf{3n + 7}$, na segunda coluna deveria colocar $\boldsymbol{\mathcal{O}(n)}$.

função	termo dominante	ordem
$\mathbf{42\frac{n^n}{n!}}$		
$\mathbf{\sum_{k=1}^{n}\left(k^3 + \frac{1}{k}\right)}$		
$\mathbf{4n^4 \log n^4 + 2022}$		
$\mathbf{1000n^3 + 1.001^n}$		
$\mathbf{\frac{700}{n} + 300}$		

4.0 **4:** A notação *big Oh* é usualmente usada para descrever a complexidade computacional de um algoritmo. Porquê?

Resposta (tente não exceder as 100 palavras):

4.0 **5:** Para a seguinte função,

```
int f(int n)
{
  int r = -20220204;

  for(int i = 0;i <= n;i++)
    for(int j = 2 * i;j <= 2 * n;j++)
      r += j / (i + 1);
  return r;
}
```

3.0 **a)** quantas vezes é executada a linha `r += j / (i + 1);`?

1.0 **b)** qual é a complexidade computacional da função?

Segunda parte do teste final de Algoritmos e Estruturas de Dados

4 de fevereiro de 2022 **17h00m – 17h55m**

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

Nome: _____

N. Mec.: _____

4.0 **1:** Explique como está organizado um *min-heap*. Para o *min-heap* apresentado a seguir, insira o número 3. 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.

1	4	2	6	5	8	7	9	
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

4.0 **2:** Explique como funciona a rotina de ordenação *insertion sort*. Indique a sua complexidade computacional e quais os seus melhores e piores casos.

4.0 **3:** Explique como pode procurar informação numa lista biligada não ordenada, e indique qual a complexidade computacional do algoritmo que descreveu. O que é que pode fazer para tornar a procura mais eficiente quando alguns itens de informação são mais procurados que outros? Tende não usar mais de 100 palavras.

4.0 **4:** Um programador pretende utilizar uma *hash table* (tabela de dispersão, dicionário) para contar o número de ocorrências de palavras num ficheiro de texto. O programador está à espera que o ficheiro tenha cerca de 6000 palavras distintas, pelo que usou uma *hash table* do tipo *separate chaining* com 10007 entradas, e usou a seguinte *hash function*:

```
unsigned int hash_function(unsigned char *s,unsigned int hash_table_size)
{
  unsigned int sum = 0u;

  for(int i = 0;s[i] != '\0';s++)
    sum += (unsigned int)(i + 1) * (unsigned int)s[i];
  return sum % hash_table_size;
}
```

Infelizmente, as expetativas do programador estavam erradas, e o ficheiro de texto era muito maior que o esperado, tendo cerca de 1000000 palavras distintas. Responda às seguintes perguntas:

1.0 **a)** A *hash function* apresentada não é das piores. Porquê?

3.0 **b)** Com *separate chaining* a *hash table* pode acomodar o milhão de palavras distintas mesmo tendo a *array* apenas 10007 entradas. Explique porquê, e explique o que é que acontece ao desempenho desta estrutura de dados.

4.0 **5:** 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)
{
  queue *q = new_queue();
  enqueue(q,n);
  while(is_empty(q) == 0)
  {
    n = dequeue(q);
    if(n != NULL)
    {
      enqueue(q,n->right);
      enqueue(q,n->left);
      visit(n);
    }
  }
  free_queue(q);
}
```

```
void f2(tree_node *n)
{
  stack *s = new_stack();
  push(s,n);
  while(is_empty(s) == 0)
  {
    n = pop(s);
    if(n != NULL)
    {
      push(s,n->right);
      visit(n);
      push(s,n->left);
    }
  }
  free_stack(s);
}
```

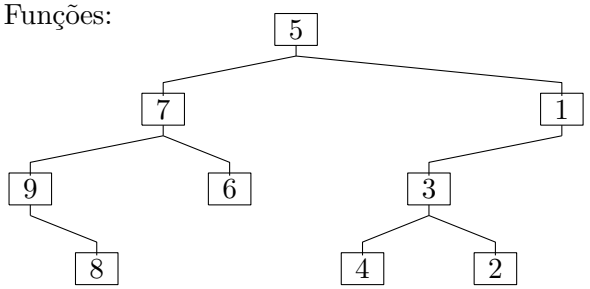
```
int cnt = 0;

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

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

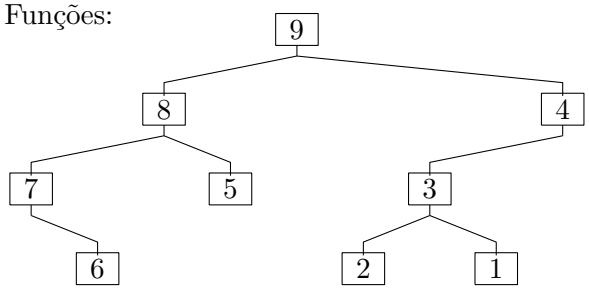
```
void f4(tree_node *n)
{
  if(n != NULL)
  {
    f4(n->right);
    visit(n);
    f4(n->left);
  }
}
```

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

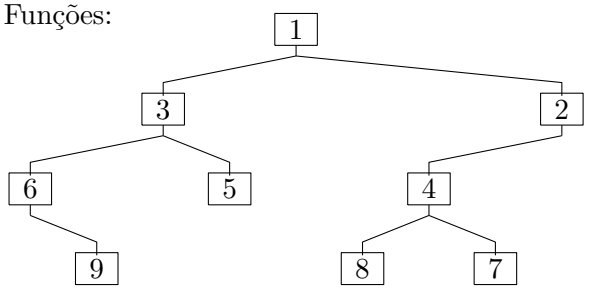
Funções:



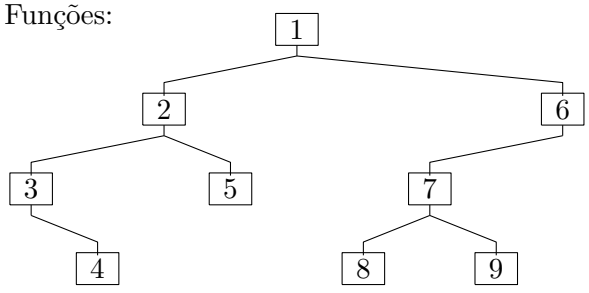
Funções:



Funções:



Funções:



Terceira parte do teste final de Algoritmos e Estruturas de Dados

4 de fevereiro de 2022 **18h00m – 18h55m**

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

Nome: _____

N. Mec.: _____

4.0 **1:** O algoritmo *merge sort* divide o *array* a ser ordenado ao meio, ordena (recursivamente) cada uma das duas partes, e depois junta-as. A sua complexidade computacional é $\boldsymbol{\Theta(n \log n)}$. Um aluno está convencido que se em vez de se dividir o array em duas partes se se dividir em cinco partes (todas mais ou menos do mesmo tamanho), então a complexidade computacional desta variante do *merge sort* será ainda mais baixa. Responda às seguintes perguntas:

1.0 **a)** Que estratégia algoritmica usa o *merge sort*?

2.0 **b)** O aluno tem razão? Justifique.

1.0 **c)** Nesta variante do *merge sort*, a fase de *merge* é mais fácil ou mais complicada que a do algoritmo original?

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

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

4.0 **2:** Num tabuleiro de xadrez, pretende-se ir do canto inferior esquerdo $(0, 0)$ para o canto superior direito $(7, 7)$ fazendo movimentos apenas para a direita e para cima. O seguinte código apresenta uma maneira de calcular o número de maneiras de fazer isso.

```
1   long eval(int x,int y)
    {
2     if(x < 0 || x > 7 || y < 0 || y > 7)
2       return 0L;
3     else if(x == 7 && y == 7)
3       return 1L;
4     else
4       return eval(x + 1,y) + eval(x,y + 1);
    }
5   long count_paths(void)
    {
6     return eval(0,0);
    }
```

Responda às seguintes perguntas:

1.0 **a)** Que estratégia algoritmica é usada por este código?

3.0 **b)** Explique por palavras qual é o objetivo de cada uma das partes numerados do código.

4.0 **3:** Tal como no problema anterior, num tabuleiro de xadrez, pretende-se ir do canto inferior esquerdo $(0, 0)$ para o canto superior direito $(7, 7)$ fazendo movimentos apenas para a direita e para cima. O seguinte código apresenta uma maneira de calcular o número de maneiras de fazer isso.

```
1   long count_data[8][8];
2   long eval(int x,int y)
    {
3     if(x < 0 || x > 7 || y < 0 || y > 7)
3       return 0L;
4     if(count_data[x][y] < 0L)
4       count_data[x][y] = eval(x - 1,y) + eval(x,y - 1);
5     return count_data[x][y];
    }
6   long count_paths(void)
    {
7     for(int x = 0;x < 8;x++)
7       for(int y = 0;y < 8;y++)
7         count_data[x][y] = -1L;
8     count_data[0][0] = 1L;
9     return eval(7,7);
    }
```

Responda às seguintes perguntas:

1.0 **a)** Que estratégia algoritmica é usada por este código?

3.0 **b)** Explique por palavras qual é o objetivo de cada uma das partes numerados do código.

4.0 **4:** Um grafo com 5 vértices, numerados de **1** a **5**, tem uma aresta entre o vértice número i e o número j se e só se $i < j$. Responda às seguintes perguntas:

1.0 **a)** Desenhe o grafo.

1.0 **b)** Represente o grafo usando uma matriz de adjacência.

1.0 **c)** Represente o grafo usando listas de adjacência.

1.0 **d)** É possível representar este grafo usando um único inteiro de **32** *bits*? Se sim, como? Se não, por que não?

4.0 **5:** Você foi capturado pelos Borg e levado para um cubo Borg para ser assimilado.[1] Você conseguiu fugir mas está perdido dentro do cubo Borg, que pode ser considerado um labirinto tridimensional. Responda às seguintes perguntas:

2.0 **a)** Para encontrar uma saída do cubo, usaria *depth search* ou *breadth search*?

2.0 **b)** Que material levaria consigo para o ajudar implementar a estratégia algoritmica que escolheu na alinea anterior?

[1] Os Borg são uma raça alienígena do universo *Star Trek*. As suas maiores naves são os cubos Borg, que têm o volume de **27** kilometros cúbicos. A resistência é fútil!