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: _____

N. Mec.: _____

Fórmulas:

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

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:

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
$n^2 \log n^2$		
$\sum_{k=1}^{n} \left(k + \frac{1}{k}\right)$		
$10^{100} n^{42} + 1.001^n$		
$n^3 + 0.999^{n/2}$		
$\frac{10^n}{n!}$		

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 = [          ];
  int high = [          ];
  while(1)
  {
    if(low [     ] high)
      return [     ];
    int middle = [                    ];
    if(a[middle] == v)
      return middle;
    if(a[middle] [     ] v)
      [          ] = middle + 1;
    else
      [          ] = 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		
[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:

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:

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);
}
```

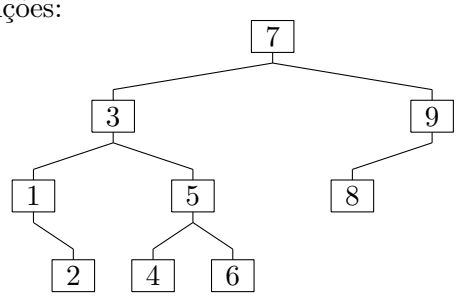
```
int cnt = 0;

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);
  }
}
```

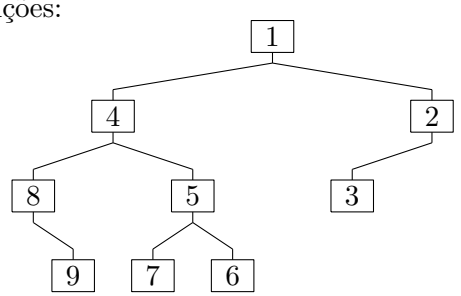
```
void f4(tree_node *n)
{
  if(n != NULL)
  {
    f4(n->left);
    visit(n);
    f4(n->right);
  }
}
```

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

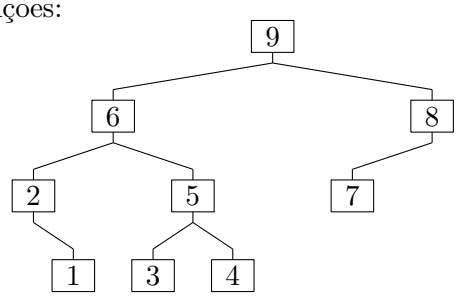
Funções:



Funções:



Funções:



Funções: