



Estruturas de Dados 1 - Gabarito da Prova Teórica 2

Nome:

Matrícula:

Data:

---

Observações:

- (A) A prova é individual e sem consulta, sendo vedado o uso de calculadoras e de telefones celulares.
  - (B) A interpretação dos comandos das questões faz parte da avaliação.
  - (C) Nas questões de múltipla escolha, apenas uma alternativa deve ser escolhida. Questões com múltiplas marcações ou sem marcação serão desconsideradas.
- 

1. A função  $f(n) = 3n^2 + 100n + 2n^3 + 5000$  é

- (A)  $O(5000)$
- (X)  $O(n^3)$
- (C)  $O(n^2)$
- (D)  $O(n)$

2. Se existem  $c$  e  $N$  positivos tais que  $f(n) \geq cg(n)$  para todos os valores  $n$  maiores ou iguais a  $N$ , então  $f(n)$  é

- (A) uma cota inferior para  $g(n)$
- (B)  $\Theta(g(n))$
- (X)  $\Omega(g(n))$
- (D)  $O(g(n))$

3. Considere a implementação da função `is_number()`, que verifica se uma string  $s$  com  $N$  caracteres representa ou não um número em base decimal:

```
1 bool is_number(const string& s)
2 {
3     int N = s.size();
4
5     for (int i = 0; i < N; ++i)
6         if (!isdigit(s[i]))
7             return false;
8
9     return true;
10 }
```

Qual dentre as descrições abaixo representa o pior caso deste algoritmo?

- (A) O valor de  $N$  é muito grande (maior do que 1.000)
- (B) O primeiro elemento de  $s$  não é um dígito decimal
- (X) A string  $s$  representa um número em base decimal
- (D) O valor de  $N$  é igual a 1

4. Considere a implementação da função `grayscale()`, que gera uma imagem  $B_{n \times n}$ , em escala de cinza, a partir de uma imagem colorida  $A_{n \times n}$ :

```
1 void grayscale(Image& B, const Image& A, int n)
2 {
3     for (int i = 0; i < n; ++i)
4         for (int j = 0; j < n; ++j)
5             B[i][j] = (A[i][j].r + A[i][j].g
6                       + A[i][j].b)/3;
7 }
```

Seja a função  $t = f(n)$  contabiliza o número de atribuições  $t$  que a função `grayscale()` realiza de acordo com o valor de  $n$ . Assim,  $f(n)$  é igual a

- (X)  $2n^2 + 2n + 1$   
 (B)  $n^2 + n + 1$   
 (C)  $1 + 4n$   
 (D)  $n^2$
5. Na definição de um vetor como um tipo abstrato de dados, qual das operações abaixo deve ter complexidade  $O(1)$ ?

- (X) acesso aleatório  
 (B) remoção  
 (C) inserção  
 (D) busca

6. Considere o vetor de inteiros `xs` declarado a seguir:

```
1 vector<int> xs { 1, 2, 3, 4, 5 };
```

Os índices que correspondem ao quarto e ao primeiro elemento são, respectivamente,

- (A) 4 e 0  
 (B) 4 e 1  
 (X) 3 e 0  
 (D) 3 e 1
7. Se a implementação do vetor mantiver o registro do número de elementos armazenados, a implementação mais eficiente para a remoção de todos os elementos tem complexidade
- (A) quadrática  
 (B) linear  
 (C) logarítmica  
 (X) constante
8. Considere a implementação abaixo da busca binária em vetores de inteiros:

```
1 int binary_search(int *xs, int N, int value)
2 {
3     int a = 0, b = N - 1, m;
4
5     while (a <= b)
6     {
7         m = a + (b - a)/2;
8
9         if (xs[m] == value)
10            return m;
11        else if (xs[m] < value)
12            a = m + 1;
13        else
14            b = m - 1;
15    }
16
17    return -1;
18 }
19
20 int main()
21 {
22     int xs[] { 2, 3, 5, 7, 11, 13, 17, 19, 23 };
23
24     binary_search(xs, 9, 17);
25
26     return 0;
27 }
```

O total de atribuições feitas na execução da chamada da linha 24 é igual a

- (A) 5  
 (X) 4  
 (C) 3  
 (D) 2
9. Considere que  $f(n) = 1 + 2n$  e  $g(n) = 7 + 2 \log_2(n)$  representem o número de atribuições feitas em duas implementações da busca linear e da busca binária, respectivamente. Qual é o maior inteiro positivo  $n$  para o qual a busca linear é mais eficiente do que a busca binária neste caso?

- (A) 2  
 (B) 3  
 (C) 4  
 (X) 5

10. Um algoritmo de ordenação que preserva a ordem relativa dos elementos iguais é denominado

(A) *in-place*  
(X) estável  
(C) remoto  
(D) local

11. Considere o código abaixo:

```
1 vector<int> xs {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
2
3 sort(xs.begin(), xs.end(), [](int a, int b) {
4     int ra = a % 3, rb = b % 3;
5     return ra == rb ? a < b : ra > rb;
6 });
```

Após a execução do código, o vetor xs será igual a

(A) { 3, 6, 9, 1, 4, 7, 10, 2, 5, 8 }  
(X) { 2, 5, 8, 1, 4, 7, 10, 3, 6, 9 }  
(C) { 9, 6, 3, 10, 7, 4, 1, 8, 5, 2 }  
(D) { 8, 5, 2, 10, 7, 4, 1, 9, 6, 3 }

12. Qual das linhas abaixo correspondem a busca pelo elemento  $x = 22$  em um vetor ordenado xs com 10 inteiros em C?

(X) `bsearch(&x, xs, 10, sizeof(int), comp);`  
(B) `bsearch(x, xs, 10, sizeof(int), comp);`  
(C) `qsort(&x, xs, 10, sizeof(int), comp);`  
(D) `qsort(x, xs, 10, sizeof(int), comp);`

13. (4 pontos) A função `merge()`, declarada abaixo, recebe dois vetores a e b, ordenados em ordem crescente, e retorna um novo vetor, também ordenado, composto por todos os elementos de a e b:

```
vector<int> merge(const vector<int>& a, const vector<int>& b);
```

Implemente a função `merge()` sem invocar nenhuma rotina de ordenação, isto é, o novo vetor deve ser preenchido com os elementos de a e b de tal forma que também fique ordenado. A complexidade da implementação deve ser  $O(N + M)$ , onde  $N$  e  $M$  são o número de elementos de a e b, respectivamente.

A implementação deve conter, no máximo, 35 linhas. Não é necessário declarar nenhuma diretiva `include`. Use a próxima folha e siga as linhas conforme a numeração indicada. Escreva com letra legível, de preferência em letras de forma, e utilize um lápis. A implementação deve começar com a assinatura da função e deve terminar com o fim do bloco da função.

Assuma que os parâmetros sempre serão válidos e que o número de elementos dos vetores é um número não-negativo. Veja um trecho do código que será utilizado para a correção.

```
9 int main()
10 {
11     vector<int> a { 4, 5 }, b { };
12
13     // Teste #01: primeiro exemplo da prova
14     auto z = merge(a, b);
15     assert(z == a);
16
17     // Teste #02: segundo exemplo da prova
18     b = vector<int>({ 1, 2, 3 });
19     z = merge(a, b);
20
21     assert(z == vector<int>({ 1, 2, 3, 4, 5 }));
22
23     // Teste #03: terceiro exemplo da prova
24     a = vector<int>({ 1, 3, 5 });
25     b = vector<int>({ 2, 4, 6 });
26     z = merge(a, b);
27
28     assert(z == vector<int>({ 1, 2, 3, 4, 5, 6 }));
```

## Solução:

```
1 #include <vector>
2
3 using namespace std;
4
5 vector<int> merge(const vector<int>& a, const vector<int>& b)
6 {
7     int N = a.size(), M = b.size(), i = 0, j = 0;
8     vector<int> c;
9
10    while (i < N and j < M)
11        a[i] < b[j] ? c.push_back(a[i++]) : c.push_back(b[j++]);
12
13    while (i < N)
14        c.push_back(a[i++]);
15
16    while (j < M)
17        c.push_back(b[j++]);
18
19    return c;
20 }
```