UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

PROCESSO SELETIVO – 02/2026

GABARITO - PROVA ESCRITA

Área de Conhecimento de Engenharias ou Ciências Exatas e da Terra – B.

Questão 1 (20%)

Explique os princípios básicos de sistemas de computação relacionados à execução de algoritmos e programas. Em particular:

a) (7%) Defina o que é um algoritmo e o que é um programa de computador, destacando as diferenças entre ambos.

b) (7%) Descreva como um algoritmo escrito em uma linguagem de alto nível é transformado em um programa executável no computador (mencionando o papel de compiladores ou interpretadores, e do hardware).

c) (6%) Comente sobre a importância das estruturas de controle (sequência, seleção e repetição) na construção de algoritmos e na lógica de programação.

Resposta. Baseado em Forbellone & Eberspächer, Lopes & Garcia e Deitel & Deitel.

a) Algoritmo é uma descrição definida de passos para resolver um problema ou executar uma tarefa que seja independente de uma linguagem específica. Programa de computador é a implementação concreta de um algoritmo (ou de vários) em uma linguagem de programação específica que obedece às regras de sintaxe da linguagem, de modo que possa ser traduzido e executado em um computador.

b) O processo começa com a escrita do algoritmo em uma linguagem de alto nível, gerando o código-fonte. Um compilador traduz o código-fonte para uma forma de baixo nível e, em seguida, ocorre o linking (coloquialmente “linkagem”) com bibliotecas para geral um executável. Para a execução, o sistema operacional carrega o programa na memória e a CPU executa as instruções (em linguagem de máquina), manipulando dados em registradores e memória e realizando operações de entrada e saída quando necessário.

c) As estruturas de controle são fundamentais pois expressam a lógica de um algoritmo de forma

estruturada. A sequência define a execução das instruções na ordem em que são descritas. Estruturas de seleção (if/else, switch/case) permitem tomar decisões e seguir caminhos diferentes conforme condições lógicas, o que torna o algoritmo adaptável a diferentes entradas e casos. Estruturas de repetição (for, while, do-while) permitem executar um conjunto de passos múltiplas vezes, seja por um número conhecido de iterações, seja até uma condição ser satisfeita. Isso é essencial para validação de entradas e para processamento de dados (vetores e matrizes), além de viabilizar algoritmos clássicos como busca, contagem, e ordenação.

Questão 2 (20%)

Leia uma sequência de inteiros até EOF (fim de arquivo). Para cada inteiro x:
- se x = 0, ignore o valor e prossiga (não conta em nada);
- se x < 0, encerre o processamento imediatamente;
- se x > 0, acumule a soma apenas dos múltiplos de 3.

Ao final, imprima a soma.

Resposta. Segue um padrão de resposta em linguagem C.

```
#include <stdio.h>

int main(void) {
    long soma = 0;
    int x;

    while (scanf("%d", &x) == 1) {
        if (x == 0)
            continue;
        if (x < 0)
            break;
        if (x % 3 == 0)
            soma += x;
    }

    printf("%ld\n", soma);
```

```
    return 0;
}
```

Questão 3 (20%)

Considere um vetor (array) de n números inteiros desordenados.

a) (7%) Apresente um algoritmo para ordenar o vetor em ordem crescente;

b) (6%) Indique qual método de ordenação foi utilizado e analise a complexidade de tempo no pior caso do seu algoritmo;

c) (7%) Caso exista, comente uma melhoria ou um algoritmo de ordenação alternativo que tenha melhor eficiência, mencionando sua complexidade.

Resposta.

a) Segue uma resposta em linguagem C.

```
#include <stdio.h>

static void insertion_sort(int v[], int n) {
    for (int i = 1; i < n; i++) {
        int chave = v[i];
        int j = i - 1;
        while (j >= 0 && v[j] > chave) {
            v[j + 1] = v[j];
            j--;
        }
        v[j + 1] = chave;
    }
}

int main(void) {
    int n;

    if (scanf("%d", &n) != 1) return 0;
```

```
    int v[n];
    for (int i = 0; i < n; i++) scanf("%d", &v[i]);

    insertion_sort(v, n);

    for (int i = 0; i < n; i++) {
        if (i) printf(" ");
        printf("%d", v[i]);
    }
    printf("\n");
    return 0;
}
```

b) No exemplo acima, o método utilizado foi o insertion sort. No pior caso (vetor em ordem decrescente), para cada posição i o algoritmo pode deslocar aproximadamente i elementos, totalizando cerca de 1 + 2 + ... + (n-1) comparações, o que leva a complexidade de tempo O(n^2) no pior caso.

c) Um algoritmo com melhor eficiência no caso geral é o mergesort, com complexidade O(n log n) no pior caso.

Questão 4 (20%)

O n-ésimo número de Fibonacci pode ser definido recursivamente como: F(0)=0, F(1)=1 e F(n)=F(n–1)+F(n–2) para n≥2. Com base nisso:

a) (7%) Escreva um algoritmo recursivo que calcule F(n).

b) (6%) Escreva um algoritmo equivalente de forma iterativa (sem recursão).

c) (7%) Compare as duas abordagens em termos de conceito e desempenho, discutindo vantagens, desvantagens e a ordem de complexidade de tempo de cada uma. Apresente um exemplo ilustrando a diferença de eficiência para valores de n grandes.

Resposta. Conforme DEITEL & DEITEL, segue um exemplo de código para os itens a): função fib_rec e b): função (fib_it).

```c
#include <stdio.h>

long fib_rec(int n) {
    if (n <= 1) return n;
    return fib_rec(n - 1) + fib_rec(n - 2);
}

long fib_it(int n) {
    if (n <= 1) return n;

    long a = 0;
    long b = 1;

    for (int i = 2; i <= n; i++) {
        long c = a + b;
        a = b;
        b = c;
    }
    return b;
}

int main(void) {
    int n;
    if (scanf("%d", &n) != 1) return 0;

    printf("fib_rec(%d) = %ld\n", n, fib_rec(n));
    printf("fib_it(%d)  = %ld\n", n, fib_it(n));
    return 0;
}
```

c) A versão recursiva expressa de forma a definição matemática. A versão iterativa expressa o cálculo como uma acumulação em laço, evitando chamadas recursivas.

A versão recursiva (fib_rec) recalcula repetidamente os mesmos valores. O tempo de execução cresce exponencialmente com n. A versão iterativa (fib_it) executa um laço de 2 até n, com trabalho constante por iteração. O tempo de execução tem ordem O(n).

Como exemplo, para n grande (digamos n = 100), a versão iterativa faz apenas 99 iterações do laço e termina rapidamente.

A versão recursiva faz um grande número de chamadas , tornando-se muito lenta. De modo prática, a diferença é de muitas ordens de grandeza conforme n aumenta o que a torna inviável.

Questão 5 (20%)

Leia dois números inteiros m e n (1 <= m,n <= 20) e uma matriz A[m][n]. Com base nisso:

a) (8%) Imprima a soma de cada linha;

b) (8%) Encontre o maior elemento da matriz e imprima valor e posição (i,j);

c) (4%) Se m = n, calcule a soma da diagonal principal; caso contrário, informe “matriz não quadrada”.

Resposta. Segue um padrão de resposta em linguagem C.

```
#include <stdio.h>

int main(void) {
   int m, n;

   if (scanf("%d %d", &m, &n) != 2)
      return 0;

   int A[20][20];

   for (int i = 0; i < m; i++)
      for (int j = 0; j < n; j++)
         scanf("%d", &A[i][j]);
```

```
    for (int i = 0; i < m; i++) {
        long soma_linha = 0;
        for (int j = 0; j < n; j++)
            soma_linha += A[i][j];
        printf("Soma da linha %d = %ld\n", i + 1, soma_linha);
    }

    int maior = A[0][0];
    int imax = 0, jmax = 0;

    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            if (A[i][j] > maior) {
                maior = A[i][j];
                imax = i;
                jmax = j;
            }
    printf("Maior = %d na posicao (%d,%d)\n", maior, imax + 1, jmax + 1);

    if (m == n) {
        long soma_diag = 0;
        for (int i = 0; i < m; i++)
            soma_diag += A[i][i];
        printf("Soma da diagonal principal = %ld\n", soma_diag);
    } else
        printf("matriz nao quadrada\n");
    return 0;
}
```

Balneário Camboriú, 9 de fevereiro de 2026.

_____

Professor Luiz A. Hegele Jr. – UDESC/CESFI
Presidente da Banca Examinadora