

**É PROIBIDO UTILIZAR QUALQUER MEIO PARA CONSULTA QUE NÃO SUA PRÓPRIA MEMÓRIA.**

- Serão aceitos algoritmos escritos em linguagem C ou pseudo código.
- As respostas podem ser escritas usando lápis ou caneta.
- Esta folha possui conteúdo no verso!

**V** (15 pontos) Associe a cada uma das descrições o método de ordenação correto (SOMENTE UM MÉTODO POR DESCRIÇÃO).

- (i) *Insertion Sort*
- (ii) *Selection Sort*
- (iii) *MergeSort*
- (iv) *BubbleSort*
- (v) *QuickSort*

- Os elementos são comparados (a partir do segundo elemento) com os elementos disponíveis à esquerda do vetor e então são colocados na posição correta até que não se tenha mais elementos.
- III** ○ O vetor de elementos é dividido em dois subvetores, que são divididos sucessivamente até que cada subvetor tenha apenas um elemento. Os subvetores são reagrupados em ordem via comparação e trocas até que se tenha o vetor original ordenado.
- II** ○ Os elementos são ordenados via inserção do menor elemento na primeira posição e movimentação (cópia) de todos os elementos remanescentes do vetor para as posições adjacentes, com ordem de complexidade quadrática.

**VI** (25 pontos) Considere a função em C abaixo, que recebe como parâmetros um vetor de números inteiros e o tamanho do vetor:

```
void questao2(int *v, int n) {
    int i, j, tmp;

    for(i = 0; i < n; i++) {
        tmp = v[i];
        j = i - 1;
        while((j >= 0) && (v[j] > tmp)) {
            v[j+1] = v[j];
            j = j - 1;
        }
        v[j+1] = tmp;
    }
}
```

- ii) A função “questao2” ordena o vetor. Qual o nome deste algoritmo de ordenação? *InsertionSort*
- iii) Que tipo de vetor de entrada de tamanho “n” pode ser considerado como melhor caso para a função “questao2” em relação ao número de atribuições para elementos do vetor? Apresente a função de custo “C(n)” para o melhor caso.

3. (30 pontos) Dado o vetor  $A = \{11, 9, 8 | 3, 0\}$  e considerando os métodos *QuickSort* com o pivô no ÚLTIMO elemento e *MergeSort* com elemento do meio sendo o piso de  $(p+u)/2$ , responda as questões a seguir:

- Apresente ambas as árvores de chamadas recursivas das funções.
- Quais os parâmetros passados na 3<sup>a</sup> chamada às funções *QuickSort* e *MergeSort*? Considere a chamada inicial com o vetor dado como entrada como a 1<sup>a</sup> chamada, por exemplo, *xSort(A, 0, 4)*.
- Qual o estado do vetor após essa 3<sup>a</sup> chamada em ambos os casos?

- \* (30 pontos) A sequência dos “números de granizo” a partir de um determinado número “x” é dada pelas seguintes regras:

- se “x” é igual a 1, a sequência termina;
- se “x” é par, o próximo número da sequência é “x / 2”;
- se “x” é ímpar, o próximo número da sequência é “3 \* x + 1”

Escreva uma função recursiva em C que recebe um inteiro “x” e imprime a sequência dos “números de granizo” a partir de “x”

10

### CÓDIGOS DE AUXÍLIO

```

void Merge(int *a, int p, int m, int u){
    int n1, n2, i, j, *e, *d, k = 0;
    n1 = m - p + 1;
    n2 = u - m;
    e = malloc(n1 * sizeof(int));
    d = malloc(n2 * sizeof(int));

    for (i = 0; i < n1; i++) {
        e[i] = a[p + i];
    }

    for (j = 0; j < n2; j++) {
        d[j] = a[m + 1 + j];
    }

    i = 0;
    j = 0;
    for (k = p; k <= u; k++) {
        if (j == n2) {
            a[k] = e[i];
            i++;
        } else if (i == n1) {
            a[k] = d[j];
            j++;
        } else if (e[i] <= d[j]) {
            a[k] = e[i];
            i++;
        } else {
            a[k] = d[j];
            j++;
        }
    }
}

int partition(int *a, int p, int u) {
    int x = a[u];
    int i = p-1;
    for (int j = p; j < u; j++) {
        if (a[j] <= x) {
            i++;
            troca(&a[i], &a[j]);
        }
    }
    troca(&a[i+1], &a[u]);
    return i+1;
}

void QuickSort(int *a, int p, int u) {
    if (p <= u) {
        int m = partition(a, p, u);
        QuickSort(a, p, m-1);
        QuickSort(a, m+1, u);
    }
}

```

```

MERGE
↓
[ A, P, U ]

```