1. Analise o seguinte programa descrito na forma de pseudocódigo:

algoritmo

declare X[10], n, i, aux, flag numérico

para i ← 1 até 10 faça

leia X[i]

n ← 1

flag ← 1

enquanto (n ≤ 10 E flag = 1) faça

inicio

flag ← 0

para i ← 1 até 9 faça

inicio

se (X[i] < X[i+1]) então

inicio

flag ← 1

aux ← X[i]

X[i] ← X[i+1]

X[i+1] ← aux

fim\_se

fim\_para

n ← n + 1

fim\_enquanto

para i ← 1 até 10 faça

escreva X[i]

fim\_algoritmo

O programa escrito acima realiza a ordenação decrescente de um vetor de números inteiros. Implemente o algoritmo de ordenação [utilizando bubble sort] no algoritimo será representado no exercicio 01,nele contém a ordenação crescente e decrescente

a função que vai utilizar neste exercício sera

𝑇(𝑛) = (𝑛 - 1) + 1

2. Um programador inexperiente afirma que a seguinte implementação da função de partição rearranja o vetor v[p..r], com p < r, e devolve um índice j ∈ [p, r − 1] tal que

v[p..j] ≤ v[j + 1..r].

int part (int v[], int p, int r) {

int q, i, j, t;

i = p; q = (p + r) / 2; j = r;

do {

**while (v[i] < v[q]) <-**

**i++; erros a serem corrigidos**

**while (v[j] > v[q]) <-**

**j—-;**

if (i <= j) {

t = v[i];

v[i] = v[j];

v[j] = t;

i++;

j–-;

}

} while (i <= j);

return i;

}

**Mostre um exemplo onde essa função não dá o resultado esperado. E se trocarmos return i por return i-1?**

porque a consulta não consegue encontrar o valor a ser ordenado, o algoritmo se perde ao procurar o índice

É possível fazer algumas poucas correções de modo que a função dê o resultado esperado?

Sim, é possível fazer alterações especializadas no assunto que está sendo tratado dependendo do que o algoritmo está necessitando.

No código acima foi necessário efetuar correções nos dois laços que percorriam as partições do vetor, após o ajuste a ordenação pelo método quick sort seguiu corretamente.

3. Faça um teste de mesa com cada método de ordenação estudado até o momento,

utilizando as seguintes sequências de dados de entrada:

1. S1 = {2, 4, 6, 8, 10, 12}

bubblesort -> 30 movimentos

selectsort ->10 movimentos

mergesort ->16 movimentos

quicksort ->10 movimentos

insertsort ->

1. S2 = {11, 9, 7, 5, 3, 1}

bubblesort ->30 movimentos

selectsort ->10 movimentos

mergesort ->16 movimentos

quicksort ->10 movimentos

insertsort ->

1. S3 = {5, 7, 2, 8, 1, 6}

bubblesort ->30 movimentos

selectsort -> 10 movimentos

mergesort -> 16 movimentos

quicksort ->10 movimentos

insertsort ->

1. S4 = {2, 4, 6, 8, 10, 12, 11, 9, 7, 5, 3, 1}

bubblesort -> 132 movimentos

selectsort ->55 movimentos

mergesort ->44 movimentos

quicksort ->28 movimentos

insertsort ->

1. S5 = {2, 4, 6, 8, 10, 12, 1, 3, 5, 7, 9, 11}

bubblesort -> 132 movimentos

selectsort ->55 movimentos

mergesort -> 44 movimentos

quicksort ->28 movimentos

insertsort ->

1. S6 = {8, 9, 7, 9, 3, 2, 3, 8, 4, 6}

bubblesort -> 90 movimentos

selectsort -> 36 movimentos

mergesort ->34 movimentos

quicksort ->

insertsort ->

1. S7 = {89, 79, 32, 38, 46, 26, 43, 38, 32, 79}

bubblesort -> 90 movimentos

selectsort -> 36 movimentos

mergesort ->34 movimentos

quicksort -> 28 movimentos

insertsort ->

Em cada caso, mostre o número de comparações e trocas que realizam na ordenação

de sequências.