Discente: Marco Silva

Resposta do exercício

1) Para cada um dos trechos de código abaixo, analise o tempo estimado de

execução no melhor e no pior caso, considerando o modelo RAM. Considere

que as variáveis n, m e vetor sejam dados de entrada.

a)

int soma = 0;

for (int i=0; i<n; i++)

soma = soma + i;

**Resposta:** Melhor caso: 1 + n

Pior caso: 1 + n

b)

int soma1 = 0;

int soma2 = 0;

for (int i=0; i<n; i++){

soma1 = soma1 + 1;

soma2 = soma2 + i;

}

**Resposta:** Melhor caso: 2 + 2n

Pior caso: 2 + 2n são ambos

c)

int soma = 0;

for (int i=0; i<n; i++){

if ( vetor[i] % 2 == 0) //se for par

soma = soma + vetor[i];

}

**Resposta:** Melhor caso: 1 + n (vetor com números ímpares)

Pior caso: 1 + 2n (vetor com números pares)

d)

int soma1 = 0;

for (int i=0; i<n; i++){

soma1 = soma1 + 1;

}

for (int j=0; j<n;j++){

soma1 = soma1 + j;

}

**Resposta:** Melhor caso: 1 + 2n

Pior caso: 1 + 2n são ambos

o segundo loop

for(int j=0; j<n; j++){

soma1 = soma1 + j;

} Complexidade O(n), este loop executa com a mesma coisa da primeira parte.

No melhor caso, o tempo total de execução é a soma das complexidades dos dois loops, mas ambos são O(n), então a complexidade total é (n + n) = n.

No pior caso: a análise é a mesma, pois o tempo de execução é linear no pior caso também independentemente dos valores dos dados.

portanto, tanto no melhor quanto no pior caso, a complexidade de tempo de execução é O(n).

e)

int soma = 0;

for (int i=0; i<n; i++){

for (int j=0; j<n; j++){

soma = soma + 1;

}

}

**Resposta:** Melhor caso: 1 + n^2

Pior caso: 1 + n^2

Tem um loop externo e o outro interno, sendo dentro e fora. O loop externo executa ‘n’ vezes e para cada iteração do loop externo, o loop interno executa ‘n’ vezes. Portanto o número total de iterações da operação soma = soma + 1 e n \* n que é ‘n^2’.

Melhor caso e pior caso, o tempo de execução é O‘n^2’ em ambos os casos, pois o número de iterações não depende das condições internas dos dados; ele sempre executa ‘n^2’ operações.

Portanto, esse código, tanto no melhor quanto no poir caso, a complexidade do tempo de execução é O‘n^2’.

f)

int soma = 0;

for (int i=0; i<n; i++){

for (int j=0; j<m; j++){

soma = soma + 1;

}

}

**Resposta:**

Melhor caso: 1 + n\*m

Pior caso: 1 + n\*m

Neste código possui um loop externo, outro loop interno é uma operação interna. Sendo o loop externo executa ‘n’ vezes, para cada iteração do loop externo, o loop interno executa ‘m’ vezes. Portanto, o número total de iteração da operação ‘soma = soma + 1’ é n \* m

Melhor e pior caso, a complexidade do tempo de execução é (n\*m) em ambos casos, pois o número total de iteração é sempre ‘n \* m’, independemente das condições específicas dos dados.

Portanto, para esse código, tanto no melhor quanto no pior caso, a complexidade do tempo de execução é n\*m.

g)

int menor = MAIOR-INTEIRO;

for (int i=0; i<n; i++){

if (vetor[i] < menor)

menor = vetor[i];

}

**Resposta:** Melhor caso: 1 + n (caso vetor esteja preenchido totalmente com MAIOR-INTEIRO)

Melhor caso: 2 + n (vetor em ordem crescente)

Pior caso: 1 + 2n (vetor em ordem decrescente)

h)

int v[][] = new int[n][n];

for (int i=0; i<n; i++){

for (int j=0; j<n; j++){

v[i][j] = i \* j;

}

}

**Resposta:**

Melhor caso: 1 + n^2

Pior caso: 1 + n^2

Esse trecho de código aloca uma matriz ‘n x n’. A alocação da matriz em si é n^2 porque você esta criando ‘n^2’ elementos, mas a análise da complexidade de tempo para a execução do código se concentra nas operações realizadas após a alocação. O loop externo executa ‘n’ iterações, o loop interno também executa ‘n’ iterações para cada iteração do loop externo. Dentro do loop interno, a operação ‘**v[i][j] = i \* j**’ é uma operação constante.

Melhor e pior caso, a complexidade de tempo de execução é **n^2** em ambos os casos, pois o número de iterações é sempre proporcional a ‘**n^2**’, independemente dos valores específicos dos dados.

i)

int menor = MAIOR-INTEIRO;

for (int i=0; i<n; i++){

if (vetor[i] < menor)

menor = vetor[i];

}

if (menor < 0){

for (int i=0; i<n; i++){

menor = menor \* (i+1);

}

}

**Resposta:**

Melhor caso: 3 + n (vetor em ordem crescente e menor elemento maior ou igual a zero)

Pior caso: 2 + 3n (vetor em ordem decrescente e menor elemento menor do que zero)

j)

int menor = MAIOR-INTEIRO;

for (int i=0; i<n; i++){

if (vetor[i] < menor)

menor = vetor[i];

}

if (menor < 0){

for (int i=0; i<n; i++){

menor = menor \* (i+1);

}

}else if (menor > 0){

for (int i=0; i<n\*n; i++)

printf(“%d\n”, menor);

} else {

printf(“%d\n”, menor);

}

**Resposta:**

Melhor caso: 5 + n (vetor em ordem crescente e menor elemento é o zero)

Pior caso: 3 + 2n + n^2 (vetor em ordem decrescente e menor elemento é positivo)

2) Dado o método de busca a seguir, analise o tempo estimado de execução no melhor e no pior caso para cada um dos trechos de código, considerando o modelo RAM. Lembre que size() é um método que retorna a quantidade de

elementos de uma lista.

Pessoa busca(String nome){

for (int i = 0; i< pessoas.size(); i++){

if (pessoas.get(i).getNome().equals(nome))

return pessoas.get(i);

}

return null;

}

Melhor caso: 2

Pior caso: 1 + n

a)

void exibir(String nome){

Pessoa p = busca(nome);

if (p != null){

p.exibirDados();

}

else{

System.out.println(“Pessoa não encontrada”);

}

}

**Resposta:** Melhor caso: 5

Pior caso: 4 + n

b)

void exibir(String nome){

if (busca(nome) != null){

busca(nome).exibirDados();

}

else{

System.out.println(“Pessoa não encontrada”);

}

}

**Resposta:** Melhor caso: 6

Pior caso: 4 + 2n

c)

void atualizar(String nome, int idade, float salario)

{ Pessoa p = busca(nome);

if (p != null){

p.setIdade(idade);

p.setSalario(salario);

}

else{

System.out.println(“Pessoa não encontrada”);

}

}

**Resposta:**

Melhor caso: 6

Pior caso: 5 + n

Analisando o método busca, setIdade e setSalario.

Tempo:

Melhor caso o metodo busca é (n), e as operações setIdade e setSalario. Portanto, a complexidade total é (n)

No pior caso onde o método busca percorre todos os elementos, a complexidade total ainda é (n).

d)

void atualizar(String nome, int idade, float salario)

{ if (busca(nome) != null){

busca(nome).setIdade(idade);

busca(nome).setSalario(salario);

}

else{

System.out.println(“Pessoa não encontrada”);

}

}

**Resposta:**

Melhor caso: 9

Pior caso: 6 + 3n

O método busca (n), método setIdade e método setSalario.

Tempo:

No melhor caso o método busca é chamado três vezes. Cada chamada é O(n), e as operações setIdade e setSalario. Assim, no melhor caso, a complexidade total n + n + n = n.

No pior caso o método busca é chamado três vezes, cada vez com complexidade O(n). A complexidade total ainda é n + n + n = n, pois as operações adicionais não alteram a ordem de grandeza.