

EXAME ESPECIAL, estruturação de 1 a 5 inclusive

11
especial

11

for (int i=0; i < N; i++) J N

for (int j=0; j < 10000; j++)

for (int k=0; k < j; k++)

some ++;

é uma constante

difícil de 10000

Assim a complexidade é de $O(N)$

11

for (int j=1; j < N; j*=2) J $\log_2 N$

for (int i=0; i < N; i++) J N

some ++

Worst case

11

for (int i=0; i < N; i++) J N

for (int k=0; k < i; k++) some ++ J N

for (int j=0; j < N; j++) some ++ J N

$$N(N+N) = N^2 + N^2 = 2N^2 \Rightarrow \underline{N^2} \text{ (complexidade } O(N^2))$$

11

for (int i=0; i < N; i++) J N

for (int j=0; j < i; j++) J N

for (int k=0; k < j; k++) J N

some ++

$$N \times N \times N = N^3 \Rightarrow O(N^3) \text{ o iterador } i \text{ é } \text{muito dependente de } N,$$

$\text{o } j \text{ é } \text{muito dependente de } i, \text{ o } k \text{ é } \text{muito dependente de } j, \text{ assim se } N$

$\text{for for many more nested loops a complexity of } O(N^3)$

① $\text{for (int } i=0; i < N; i++) \text{ JN}$
 $\text{for (int } k=0; k < i; k++) \text{ JN}$
 Some ++;

$\text{for (int } i=0; i < N; i++) \text{ JN}$
 Some ++

$$N \times N + N = N^2 + N = \text{parallel o main} \Rightarrow O(N^2)$$

② $\text{for (int } i=0; i < N; i++) \text{ JN}$
 $\text{int } M = N > 1000 ? 1000 : N;$
 $\text{for (int } j=0; j < M; j++)$
 $\text{for (int } k=0; k < M; k++)$
 Some ++

Minines

$$\frac{N^2}{2}$$

cider

$$N + N \left(N + \frac{N}{2} \right) = N + N \left(\frac{3N}{2} \right) = N + \frac{3N^2}{2} \Rightarrow N^2 \\ O(N^2)$$

③ $\text{int } max = 100; \text{ lim;}$
 $\text{for (int } i=0; i < N; i++) \text{ JN}$
 $\text{if (} i > max \text{) max += 100;}$
 $\text{int } R=0;$
 $\text{while (} R < max \text{) }$
 Some ++; R++

for (ida fn)
 N

$$N \times N = N^2, \text{ for values greater than } N$$

1) $\text{for (long } i=0; i < N; i++) \text{ JN}$
 $\text{for (long } j=0; j < N; j++) \text{ JN}$
 Soma ++

$$N \times N = N^2 \rightarrow O(N^2)$$

$$x \mapsto 4N \times 4N = 16N^2, \text{ tempo constante } 16x$$

2) $\text{for (long } i=0; i < N; i++) \text{ JN}$
 Soma ++

$$x \mapsto 4N = 4N, \text{ tempo constante } 4x$$

3) $\text{for (long } i=0; i < N; i+=2) \text{ JN}$, apesar de $i+=2$ em que
 existe incremento de passo
 de 2, tempo de execução
 $N/2$
 $O(N)$
 $x \mapsto N/2$

4) $\text{for (long } i=0; i < 1000; i++) \text{ J constante}$
 $\text{for (long } j=0; j < N; j++) \text{ JN}$
 Soma ++

ordem linear $O(N)$

$$4x \rightarrow 4N$$

5) $\text{for (long } i=0; i < N; i++) \text{ JN}$
 Soma ++

$\text{for (long } j=0; j < N; j++) \text{ JN}$
 Soma ++

$$N + N = \underline{2N} \rightarrow O(N)$$

constante

1] if (n > 20000) n = 20000] combkt

for (long i=0; i < N; i++) JN

for (long j=0; j < N; j++) JN

Soma ++

$$N > 20000 \quad N \times N = O(N^2)$$

combkt $O(1)$

$N > 20000$ è una costante log è $O(1)$

assim a repita é uma volta de N suficientemente elevada
para $O(1)$

2] for (long i=0; i < N; i++)

for (long j=0; j < N * N; j++)

Soma ++

JN
 $\frac{JN}{JN^2}$

$$N \times (N^2) = N^3 \rightarrow O(N^3)$$

centrica ($O(N^3)$)

$$1 \times \rightarrow \text{Tempo de execução} \rightarrow 4N \times (4N \times 4N) = 64m^3$$

3] for (long i=0; i < N; i++) JN

for (long j=0; j < i; j++) JN

Soma ++

$$N \times N = O(N^2)$$

O ciclo interno é executado $N-1$ iterações gerais!

4] for (long i=0; i < N; i++)

for (long j=0; j < i; j++)

Soma ++

JN $\times N$

$$N \times N \times N = O(N^3)$$

$$1 \times \rightarrow (N \times N \times N) = 69 \times$$

1. J. $\text{for} (\text{long } i=1; i < n; i *= 2)$ $\mathcal{J} N$
Soma ++

i duplica a cada iteração
 $O(\log N)$

$q \times \rightarrow$ mais duas iterações

$\log_2 N$ é o número de vezes que é preciso multiplicar por 2 (ou dividir em 2) até chegar a N

1. $\text{for} (\text{int } x=1; x < N; x *= 2)$ $\mathcal{J} \log N$
 $\text{for} (\text{int } i=0; i < 50000; i++)$ $\mathcal{J} \text{constante}$
 $\text{for} (\text{int } k=0; k < N; k++)$ $\mathcal{J} N$
Soma ++
 $O(\log N \times N) \rightarrow O(N \times \log N)$

1. $\text{for} (\text{int } i=1; i < N; i++)$ $\mathcal{J} N$ # teste
 $\text{if } (a[i] == 0)$
 $\text{for} (\text{int } j=0; j < N; j++)$ $\mathcal{J} N$
Soma ++;
 $\text{if } (a[i] == 1)$
break;

↳
 $i \quad N = 10 \quad a[i] == 0$
 $j \quad 0$

$O(N) ??$ será que a N suficientemente alta
e $a[i] \neq 1$

15
Ajustar
} relembrar
} $2^k = N \rightarrow \log_2(N)$

1) $\text{if } (N > 10000)$] constante
 $N = 10000$
 $\text{for (int } i=0; i < N; i++) \quad] N$
 $\text{for (j=2; j < N; j*=2) \quad] \log N}$

para valores de N suficientemente elevados temos $O(1)$

1) $\text{for (int } x=1; x < 100; x++) \quad] \text{constante}$
 $\text{for (int } i=0; i < N; i++) \quad] N$
 $\text{if } (i > 100)$
 $\text{for (int } k=0; k < N-i; k++) \quad] N$
 $\text{soma}++ \quad \checkmark \text{ if van sack certo}$
 orden linear, $O(N)$?

1) $\text{for (int } x=1; x < N; x*=2) \quad] \log N$
 $\text{for (int } i=0; i < \underline{50000}; i++) \quad] \text{constante}$
 $\text{for (int } k=0; k < N; k++) \quad] N$) m.e grande
 $\text{soma}++ \quad \checkmark \text{ ja!}$
 complexidade $O(\log N)$

[7]

especial

2- Assuma que dispõe de um método int pesquisa (int m[], int valor) que efectua uma pesquisa binária. Este método devolve a posição em que o valor procurado se encontra, ou então um valor negativo (-X) caso este não esteja no array indicado. O valor de abs(X+1) indica uma posição em que o valor procurado poderia ser inserido para preservar a ordem. Este método já existe e não precisa de o fazer. Construa um método int proximo(int m[], int valor) que indica qual o valor inteiro do array que é maior do que valor. Caso não exista nenhum elemento nestas condições, deve devolver valor. O array m encontra-se ordenado, e existem, no máximo, duas cópias de cada valor. O método deve ter desempenho logarítmico (outras soluções não são valorizadas).

```

int proximo( int m[], int valor ) {
    if (m.length == 0) {
        return valor;
    }
    int i = procura( m, valor );
    if (i > 0) {
        i = abs( i + 1 );
    }
    if (i == m.length - 1) { // ultime pos
        return m[i];
    }
    if (m[i+1] > m[i]) {
        return m[i+1];
    } else if (m[i+2] > m[i]) {
        return m[i+2];
    }
    return m[i];
}

```

2- Assuma que dispõe de um método int pesquisa (int m[], int valor) que efectua uma pesquisa binária. Este método devolve a posição em que o valor procurado se encontra, ou então um valor negativo (-X) caso este não esteja no array indicado. O valor de abs (X+1) indica uma posição em que o valor procurado poderia ser inserido para preservar a ordem. Este método já existe e não precisa de o fazer. Construa um método int maiores(int m[], int valor) que indica quantos valores maiores do que valor existem em m. O array m encontra-se ordenado, e existem, no máximo, duas cópias de cada valor. O método deve ter desempenho logarítmico.

18

exercício

desempenho
logarítmico

m na
loop

int maiores (int m[], int valor) {

if (m.length == null) {

return 0;

}

int i = pesquisa (m, valor);

if (i < 0) {

i = abs (i + 1);

}

int qt = m.length - i - 1;

if (m[i + 1] == m[i]) {

qt --;

}

return qt;

}

2- Assuma que dispõe de um método `int pesquisa (int m[], int valor)` **que efectua uma pesquisa binária.** Este método devolve a posição em que o valor procurado se encontra, ou então um valor negativo ($-X$) caso este não esteja no array indicado. O valor de `abs (x+1)` indica uma posição em que o valor procurado poderia ser inserido para preservar a ordem. **Este método já existe e não precisa de o fazer.** Construa um método `int tamanhoExclusão(int m[], int n[])` que devolve a dimensão de um array que inclui todos os valores de `n` que se encontram em `m`.. Cada um dos arrays `m` e `n` estão ordenados e têm elementos repetidos (mas podem existir valores que se encontram em ambos os arrays). **O método deve ter desempenho de ordem $O(N * \log(M))$, onde N e M são a dimensão dos arrays n e m, respectivamente.**


especial

Por exemplo, se `n={1,3,3,4,7}` e `m={1,2,3,3,3,3,4,5,6}`, então o método deve devolver 4 (correspondentes ao número 1, às duas cópias do número 3 e ao 4).

*desempenho $O(N * \log(M))$*

```
int tamanhoExclusão (int m[], int n[]) {  
    if ((m == null) || (n == null))  
        return 0;  
    }  
    int conta = 0;
```

2- Pretende-se criar um método boolean pesquisa (int m[], int valorX, int valorY). Sabe-se que o array contém informação relativa às coordenadas de pontos, sendo a coordenada x guardada numa posição par e a coordenada y do mesmo ponto guardada na posição imediatamente seguinte. O método verifica, em tempo logarítmico, se o array m contém um ponto com as coordenadas (valorx, valory). O array m está organizado da seguinte forma:

10
Mecid

- Todos os pontos estão organizados por ordem crescente da coordenada x
- Todos os pontos que partilham a mesma coordenada x estão ordenados por ordem da coordenada y.

Exemplo:

O array {1,2,1,10,3,0,3,2,4,10} guarda os pontos (1,2), (1,10), (3,0), (3,2), (4,10). Note a ordem crescente da coordenada x e ordem crescente da coordenada y em caso de empate da coordenada x.

Recorda-se que o método deve ter desempenho logarítmico (outras soluções não são valorizadas).

assuma que dispõe de um método int pesquisa(int m[], int valor) que efetua uma pesquisa binária. Este método devolve a posição em que o valor procurado se encontra, ou então um valor negativo (-X) caso este não esteja no array indicado. O valor de abs(X-1) indica uma posição em que o valor procurado poderia ser inserido para preservar a ordem. Este método já existe e não precisa de o fazer.

(11)
aprend

Construa um método que recebe um array m, bem como um valor x e indica se esse valor aparece três ou mais vezes no array.

O array encontra-se ordenado. O método deve ter desempenho logarítmico. Soluções que não respeitem esta restrição não são valorizadas.

Exemplo:

```
int m[] = {2,4,4,6,6,6,9,20,20,20,20};
```

```
triplicado(m, 4); //falso
```

```
triplicado(m, -12); //falso
```

```
triplicado(m, 6); //verdadeiro
```

```
triplicado(m, 20); //verdadeiro
```

```
boolean triplicado (int m[], int valor) {  
    if (m.length == null) {  
        return false;  
    }  
}
```

```
    int pos = pesquisa (m, valor);  
    if (pos < 0) {  
        return false;  
    }  
}
```

```
    if (m.length > pos + 2) {  
        if (m[pos + 1] == valor) {  
            if (m[pos + 2] == valor)) {  
                return true;  
            }  
        }  
    }  
}
```

```
    return false;  
}
```

desempenho
logarítmico

- 2) Utilizando uma função `int pesquisaValor(int [] array, int valor)`, que já está implementada e retorna a posição de `valor` caso exista ou `-X` caso não exista. Faça uma função que recebe 2 arrays de inteiros e faça a contagem dos números intercetados. EX: $n = \{1,2,3,4,5,6\}$; $m = \{3,5,6\}$; Interseção = $\{3,5,6\}$ – Retorno da função será 3.

Esta função deve ter complexidade $O(n \log(M))$.

(14)

específico

complexidade
 $O(n \log(m))$

em
 $\log m$
análise

115
Operações

2- Assuma que dispõe de um método int pesquisa (int m[], int valor) que efectua uma pesquisa binária. Este método devolve a posição em que o valor procurado se encontra, ou então um valor negativo (-X) caso este não esteja no array indicado. O valor de abs(x+1) indica uma posição em que o valor procurado poderia ser inserido para preservar a ordem. Este método já existe e não precisa de o fazer. Construa um método que recebe um array m, bem como um valor x e imprime no ecrã o menor intervalo (em termos de distância entre o limite inferior e superior), cujos limites fechados inferior e superior (que devem ser valores diferentes) estão definidos no array m (ou são abertos para + ou - infinito) e que contêm x (veja o exemplo). O array m encontra-se ordenado e não contém valores duplicados. O método deve ter desempenho logarítmico. Soluções que não respeitem esta restrição não são valorizadas.

Exemplo:

```
int m[] = {1, 4, 6, 9, 26};  
  
intervalo(m, 4); // 4 está contido em [4, 6]  
// (não [4, 4] porque os limites inf e sup têm de ser diferentes)  
// não [1, 4] porque é maior do que [4, 6]: 4-1=3 e 6-4=2 )  
intervalo(m, 2); // 2 está contido em [1, 4]  
intervalo(m, 1); // 1 está contido em [1, 4]  
intervalo(m, 22); // 22 está contido em [20, +inf[
```

(Responda no verso)

(6)

Raf

2- Assuma que dispõe de um método int pesquisa (int m[], int valor) que efectua uma pesquisa binária. Este método devolve a posição em que o valor procurado se encontra, ou então um valor negativo (-X) caso este não esteja no array indicado. Quando o valor devolvido é negativo, abs(X+1) indica a posição em que o valor procurado poderia ser inserido para preservar a ordem. Este método já existe e não precisa de o fazer.

Construa um método que recebe um array m, bem como um valor x. O array ordenado e tem, no máximo, duas cópias de cada valor. O método devolve o número de valores do array que se encontram no intervalo fechado $[x/2, x*2]$, e que são diferentes de x.

O método deve ter desempenho logarítmico. Soluções que não respeitem esta restrição não são valorizadas.

(Responda no verso)

desempenho
logarítmico
solução

int numeroValor (int m[], int valor) {

if (m.length == null) return -1;

int meio, inicio = x/2, fim = x * 2;

while (inicio <= fim) {

meio = (inicio + fim) / 2;

if (m[meio] == x) {

if (m[meio-1] == x || m[meio+1] == x) {

return x*2 - x/2 - 2;

} else {

if (x[meio] < x) { inicio = meio + 1; } else {

fim = meio;

}

return x*2 - x/2;

y

zona que descreve um método int procura (int m[], int valor) que efetua pesquisa binária. Este método devolve a posição em que o valor procurado se encontra, ou um valor negativo (-1) caso este não esteja no array indicado. Quando o valor devolvido é -1, abs(x+1) indica a posição em que o valor procurado poderia ter inserido para preservar a ordem.

é um método que recebe um array A de inteiros, ordenado, assim como dois valores inteiros, com $1 \leq M \leq 10$. O método deve verificar se X ocorre pelo menos M vezes na array. O método tem complexidade logarítmica em função do tamanho do array A. Soluções que não respeitam a restrição não são valorizadas.
(da no verso)

17

Especial

Teste 1

complexidade
logarítmica
binária

```
boolean ocorre(int a[], int vda, int M) {
    if (a.length == 0) {
        return false;
    }
    int pvalor = procura(a, vda);
    if (pvalor < 0) return false;
    int quantable = 1;
    if (a[pvalor] == a[m-1]) {
        return true;
    }
}
```

return false;

$$f(x) = x^2 - 1 = 0$$

(18)
specie

a) Construa um método `contaUnicos` sabendo que

- O método recebe um parâmetro `lista`, que é um `List` de um tipo genérico.
- É possível comparar quaisquer dois valores contidos em `lista`, através de uma instrução como `valor1.compareTo(valor2)`.
- A lista está ordenada.

O método deve contar todos os valores que não estão duplicados.

O método deve ter complexidade $O(1)$ em termos de espaço necessário, e deve ser tão eficiente quanto possível em termos temporais independentemente do tipo de lista que é recebida: (outras soluções não são valorizadas).

Exemplo:

Se $m = \{1, 10, 10, 10, 90, 91, 91, 100\}$

Então `contaUnicos(m) == 3`

Correspondente à contagem dos valores 1, 90 e 100.

b) Indique qual a complexidade (temporal) do método que construiu caso a lista seja um `ArrayList`.

c) Indique qual a complexidade (temporal) do método que construiu caso a lista seja um `LinkedList`.

145
Operad

3 – Construa o um método `f` que recebe dois parâmetros:

- um parâmetro primeiro que é uma List de um determinado tipo genérico `T`
- um parâmetro segundo que é uma lista que contém valores que podem ser comparados com os valores da primeira lista.

O método deve remover de primeiro todos os valores menores do que todos os valores que se encontram em segundo. **Deve assegurar-se que o método é tão eficiente quanto o possível, sem utilizar espaço adicional, independentemente do tipo de lista que é recebida (ArrayList ou LinkedList).**

R:

(20)
Eficiência