Análise Assintótica de Trechos Iterativos

Prof. Juliano Foleis

Análise Assintótica de Trechos Iterativos

Definição: Trecho Iterativo

Um **trecho iterativo** é uma sequência de instruções, sem recursões.

Análise Assintótica de Trechos Iterativos

Definição: Trecho Iterativo

Um **trecho iterativo** é uma sequência de instruções, sem recursões.

Definição: Algoritmo Iterativo

Um **Algoritmo iterativo** é um algoritmo composto de uma sequência de trechos iterativos. Consideramos todo algoritmo iterativo também é um **trecho iterativo**, uma vez que na prática eles são apenas sequências de instruções.

Análise Assintótica de Trechos Iterativos

Análise Assintótica de Trechos Iterativos

- Conforme discutimos nas aulas anteriores, não é necessário contabilizar exatamente o custo de cada linha de um algoritmo, uma vez que assintoticamente o custo será definido pelo termo de maior ordem.

Análise Assintótica de Trechos Iterativos

- Conforme discutimos nas aulas anteriores, não é necessário contabilizar exatamente o custo de cada linha de um algoritmo, uma vez que assintoticamente o custo será definido pelo termo de maior ordem.
- Desta forma, é suficiente analisarmos o custo apenas dos trechos mais custosos de um algoritmo. Chamamos estes trechos de **trechos quentes**.

Análise Assintótica de Trechos Iterativos

- Conforme discutimos nas aulas anteriores, não é necessário contabilizar exatamente o custo de cada linha de um algoritmo, uma vez que assintoticamente o custo será definido pelo termo de maior ordem.
- Desta forma, é suficiente analisarmos o custo apenas dos trechos mais custosos de um algoritmo. Chamamos estes trechos de **trechos quentes**.
  - Como saber quais trechos são mais custosos que outros?

Análise Assintótica de Trechos Iterativos

- Conforme discutimos nas aulas anteriores, não é necessário contabilizar exatamente o custo de cada linha de um algoritmo, uma vez que assintoticamente o custo será definido pelo termo de maior ordem.
- Desta forma, é suficiente analisarmos o custo apenas dos trechos mais custosos de um algoritmo. Chamamos estes trechos de **trechos quentes**.
  - Como saber quais trechos são mais custosos que outros?
  - A idéia é aprender o custo de diferentes padrões e usar a experiência adquirida para classificar trechos em potencial como custosos ou não.

Análise Assintótica de Trechos Iterativos

- Conforme discutimos nas aulas anteriores, não é necessário contabilizar exatamente o custo de cada linha de um algoritmo, uma vez que assintoticamente o custo será definido pelo termo de maior ordem.
- Desta forma, é suficiente analisarmos o custo apenas dos trechos mais custosos de um algoritmo. Chamamos estes trechos de **trechos quentes**.
  - Como saber quais trechos são mais custosos que outros?
  - A idéia é aprender o custo de diferentes padrões e usar a experiência adquirida para classificar trechos em potencial como custosos ou não.
  - Por exemplo, trechos que estão em um nível mais profundo de aninhamento de laços de repetição são sempre suspeitos de serem mais custosos que trechos que estão fora de laços de repetição.

Trecho 0

```
1 int soma = 0;
2 for(int i = 1; i <= n; i++){
3   soma++;
4 }
```

Trecho 0

```
1 int soma = 0;
2 for(int i = 1; i <= n; i++){
3   soma++;
4 }
```

Análise: A variável que indica o tamanho do problema é n. Note que a linha 3 é a mais frequentemente executada do algoritmo. Portanto, avaliar o número de execuções da linha 3 é suficiente para determinarmos o comportamento assintótico da função de custo. Como a linha 3 é executada uma vez por iteração do `for`, que repete $\Theta(n)$ vezes ($n - 1 + 1 = \Theta(n)$), a linha 3 executa $\Theta(n)$ vezes. Como o custo unitário da linha 3 é $\Theta(1)$, o custo total da linha 3 é $\Theta(n) \times \Theta(1) = \Theta(n)$.

Trecho 1

```
int soma = 0;
for(int i = 1; i <= n; i++){
  for(int j = 1; j <= n; j++){
    soma++;
  }
}
```

Trecho 1

```
1 int soma = 0;
2 for(int i = 1; i <= n; i++){
3   for(int j = 1; j <= n; j++){
4     soma++;
5   }
6 }
```

Análise: A variável que indica o tamanho do problema é n. Note que a variável de controle do laço interno (j) não depende da variável de controle do laço externo (i). Portanto, podemos analisar o laço interno independentemente do laço externo.

Trecho 1

```
1 int soma = 0;
2 for(int i = 1; i <= n; i++){
3   for(int j = 1; j <= n; j++){
4     soma++;
5   }
6 }
```

O trecho das linhas 3–5 é o mesmo que o Trecho 0 já estudado, tem custo $\Theta(n)$. Desta forma, em termos de custo, podemos analisar o Trecho 1 como:

```
1 int soma = 0;
2 for(int i = 1; i <= n; i++){
3   //
4   // theta(n)
5   //
6 }
```

Trecho 1

```
1 int soma = 0;
2 for(int i = 1; i <= n; i++){
3   //
4   // theta(n)
5   //
6 }
```

Notamos que o laço externo executa n vezes o laço interno que, conforme mostrado, tem custo $\Theta(n)$. Desta forma, o custo do algoritmo é dado por:

$$\underbrace{\mathbf{n}}_{\text{repetições laço externo}} \cdot \underbrace{\Theta(n)}_{\text{custo for interno}} = \Theta(n^2).$$

Portanto, o custo deste trecho iterativo é $\Theta(n^2)$.

Trecho 2

```
1 int soma = 0;
2 for(int i = 1; i <= n; i++){
3   for(int j = 1; j <= i; j++){
4     soma++;
5   }
6 }
```

Trecho 2

```
1 int soma = 0;
2 for(int i = 1; i <= n; i++){
3   for(int j = 1; j <= i; j++){
4     soma++;
5   }
6 }
```

Análise: A variável que indica o tamanho do problema é n. Neste trecho, os valores que a variável de controle do laço interno (j) assume depende do valor atual da variável de controle do laço externo (i). Portanto, temos que analisar ambos laços conjuntamente.

Trecho 2

```
1 int soma = 0;
2 for(int i = 1; i <= n; i++){
3   for(int j = 1; j <= i; j++){
4     soma++;
5   }
6 }
```

i	Valores de j	Linha 4 executa
1	$1 \ldots 1$	1 vez
2	$1 \ldots 2$	2 vezes
3	$1 \ldots 3$	3 vezes
$\ldots$	$\ldots$	$\ldots$
n	$1 \ldots n$	n vezes

A Tabela acima mostra os valores que j assume para cada valor de i, além de uma contagem de quantas vezes a linha 4 executa para cada valor de i. Note que a coluna “Linha 4 executa” é simplesmente a quantidade de valores que j assume para aquele valor de i. Para computar esta coluna basta usar a fórmula do número de inteiros em um intervalo: $(\text{fim} - \text{ini}) + 1$.

Trecho 2

```
1 int soma = 0;
2 for(int i = 1; i <= n; i++){
3   for(int j = 1; j <= i; j++){
4     soma++;
5   }
6 }
```

i	Valores de j	Linha 4 executa
1	$1 \dots 1$	1 vez
2	$1 \dots 2$	2 vezes
3	$1 \dots 3$	3 vezes
$\dots$	$\dots$	$\dots$
n	$1 \dots n$	n vezes

Note pela Tabela que quantidade total de execuções da linha 4 é diretamente proporcional ao valor de i no laço externo. Para dado valor de i, o laço interno é executado i vezes ($j = 1 \dots i$). Como i varia de 1 até n, a linha 4 executa

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{n^2+n}{2} = \frac{n^2}{2} + \frac{n}{2} = \Theta(n^2) \text{ vezes.}$$

Trecho 2

```
1 int soma = 0;
2 for(int i = 1; i <= n; i++){
3   for(int j = 1; j <= i; j++){
4     soma++;
5   }
6 }
```

i	Valores de j	Linha 4 executa
1	$1 \ldots 1$	1 vez
2	$1 \ldots 2$	2 vezes
3	$1 \ldots 3$	3 vezes
$\ldots$	$\ldots$	$\ldots$
n	$1 \ldots n$	n vezes

Como o custo da linha 4 é $\Theta(1)$, e ela executa $\Theta(n^2)$ vezes, o custo deste trecho é

$$\underbrace{\Theta(1)}_{\text{linha 4}} \cdot \underbrace{\Theta(n^2)}_{\text{exec. linha 4}} = \Theta(n^2).$$

Trecho 3

```
1 int soma = 0;
2 for(int k = 1; k <= n; k*=2){
3   for(int j = 1; j <= n; j++){
4     soma++;
5   }
6 }
```

Trecho 3

```
1 int soma = 0;
2 for(int k = 1; k <= n; k*=2){
3   for(int j = 1; j <= n; j++){
4     soma++;
5   }
6 }
```

Análise: A variável que indica o tamanho do problema é n. Note que a variável de controle do laço interno (j) não depende da variável de controle do laço externo (i). Portanto, podemos analisar o laço interno independentemente do laço externo.

Trecho 3

```
1 int soma = 0;
2 for(int k = 1; k <= n; k*=2){
3   for(int j = 1; j <= n; j++){
4     soma++;
5   }
6 }
```

O trecho das linhas 3–5 é o mesmo que o Trecho 0 já estudado, tem custo $\Theta(n)$. Desta forma, em termos de custo, podemos analisar o Trecho 3 como:

```
1 int soma = 0;
2 for(int k = 1; k <= n; k*=2){
3   //
4   // theta(n)
5   //
6 }
```

Trecho 3

```
int soma = 0;
for(int k = 1; k <= n; k*=2){
  //
  // theta(n)
  //
}
```

Trecho 3

```
1 int soma = 0;
2 for(int k = 1; k <= n; k*=2){
3   //
4   // theta(n)
5   //
6 }
```

Supondo que n é potência de 2, o valor de k do laço externo assume os valores $1, 2, 4, 8, \ldots n$, ou seja, $2^0, 2^1, 2^2, 2^3, \ldots n$. Logo, na última iteração, $k = n = 2^i, i \in \mathbb{Z}$. Portanto, na i-ésima iteração, $n = 2^i \Leftrightarrow \lg(n) = i$. Assim, o laço externo executa $\lg(n)$ vezes o laço interno, que tem custo $\Theta(n)$. Desta forma, o custo do trecho todo é

$$\underbrace{\Theta(n)}_{\text{laço interno}} \cdot \underbrace{\Theta(\lg(n))}_{\text{reps. for externo}} = \Theta(n\lg(n))$$

Trecho 4

```
1 int soma = 0;
2 for(int k = 1; k <= n; k*=2){
3   for(int j = 1; j <= k; j++){
4     soma++;
5   }
6 }
```

Trecho 4

```
1 int soma = 0;
2 for(int k = 1; k <= n; k*=2){
3   for(int j = 1; j <= k; j++){
4     soma++;
5   }
6 }
```

Análise: A variável que indica o tamanho do problema é n. Neste trecho, os valores que a variável de controle do laço interno (j) assume depende do valor atual da variável de controle do laço externo (k). Portanto, temos que analisar ambos laços conjuntamente.

Trecho 4

```
1 int soma = 0;
2 for(int k = 1; k <= n; k*=2){
3   for(int j = 1; j <= k; j++){
4     soma++;
5   }
6 }
```

k	Valores de j	Linha 4 executa
1	$1 \ldots 1$	1 vez
2	$1 \ldots 2$	2 vezes
4	$1 \ldots 4$	4 vezes
8	$1 \ldots 8$	8 vezes
$\ldots$	$\ldots$	$\ldots$
n	$1 \ldots n$	n vezes

A Tabela acima mostra os valores que j assume para cada valor de k, além de uma contagem de quantas vezes a linha 4 executa para cada valor de k. Note que a coluna “Linha 4 executa” é simplesmente a quantidade de valores que j assume para aquele valor de k. Para computar esta coluna basta usar a fórmula do número de inteiros em um intervalo: $(\text{fim} - \text{ini}) + 1$.

Trecho 4

```
1 int soma = 0;
2 for(int k = 1; k <= n; k*=2){
3   for(int j = 1; j <= k; j++){
4     soma++;
5   }
6 }
```

k	Valores de j	Linha 4 executa
1	$1 \dots 1$	1 vez
2	$1 \dots 2$	2 vezes
4	$1 \dots 4$	4 vezes
8	$1 \dots 8$	8 vezes
$\dots$	$\dots$	$\dots$
n	$1 \dots n$	n vezes

Somando a coluna “Linha 4 executa” acima nos dá a quantidade total de execuções da linha 4, uma vez que ela contabiliza quantas vezes o laço interno repete para cada valor de k do laço externo. Desta forma, no total a linha 4 executa
$1 + 2 + 4 + 8 + \cdots + n = 2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^i$ vezes, tal que $2^i = n$. Desta forma, resolvendo para i, temos que
$2^i = n \Leftrightarrow i = \lg(n)$. Desta forma, sabemos que k varia de 0 até $\lg(n)$.

Trecho 4

```
1 int soma = 0;
2 for(int k = 1; k <= n; k*=2){
3   for(int j = 1; j <= k; j++){
4     soma++;
5   }
6 }
```

Assim, a linha 4 executa

$$\begin{aligned}\sum_{i=0}^{lg(n)} 2^i &= \frac{2^{lg(n)+1} - 1}{2 - 1} \\ &= 2.2^{lg(n)} - 1 \\ &= 2.n^{lg(2)} - 1 \\ &= 2n - 1 = \Theta(n) \text{ vezes.}\end{aligned}$$

Como o custo da linha 4 é $\Theta(1)$ e ela é executada $\Theta(n)$ vezes, o custo deste trecho é $\Theta(n)$.

Trecho 5

```
1 f(n); // custo: theta(n)
2 g(n); // custo: theta(n lg n)
```

Trecho 5

```
1 f(n); // custo: theta(n)
2 g(n); // custo: theta(n lg n)
```

Análise: Ambas linhas são executadas. Portanto, o custo deste trecho é

$$\underbrace{\Theta(n)}_{\text{linha 1}} + \underbrace{\Theta(n\lg(n))}_{\text{linha 2}} = \Theta(n\lg(n)).$$

Trecho 6

```
1 if(par(n)){
2   f(n); // custo: theta(n)
3 }
4 else{
5   g(n); // custo: theta(n lg n)
6 }
```

Trecho 6

```
1 if(par(n)){
2   f(n); // custo: theta(n)
3 }
4 else{
5   g(n); // custo: theta(n lg n)
6 }
```

Análise: A variável que indica o tamanho do problema é n. No melhor caso, n é par e $f(n)$ é executada, portanto o custo é $\Theta(n)$. Por outro lado, no pior caso, n é ímpar e $g(n)$ é executada, portanto ao custo é $\Theta(n\,lg(n))$. Note que o custo da linha 1 é constante, portanto não influencia no comportamento assintótico da função de custo.

SelectionSort

```
 1 void SelectionSort(int* V, int n){
 2   for(int i = 0; i < (n-1); i++){
 3     int menor = i;
 4     for(j = i+1; j <= n; j++){
 5       if(v[menor] > v[j]){
 6         menor = j;
 7       }
 8     }
 9     troca(V, i, menor); // custo: theta(1)
10   }
11 }
```

SelectionSort

```
1  void SelectionSort(int* V, int n){
2    for(int i = 0; i < (n-1); i++){
3      int menor = i;
4      for(j = i+1; j <= n; j++){
5        if(v[menor] > v[j]){
6          menor = j;
7        }
8      }
9      troca(V, i, menor); // custo: theta(1)
10   }
11 }
```

Análise: Por se tratar de um algoritmo de ordenação, o tamanho do problema, n, indica o número de elementos do vetor sendo ordenado. O comportamento do `SelectionSort` não depende da entrada, assim não há noção de melhor caso ou pior caso. Isto é evidenciado no fato que o laço externo varre sempre o vetor do início ao fim, e sua variável de controle nunca é alterada em seu interior. Além disso, o laço interno depende diretamente do laço externo e sua variável de controle nunca é alterada em seu interior.

SelectionSort

```
1  void SelectionSort(int* V, int n){
2    for(int i = 0; i < (n-1); i++){
3      int menor = i;
4      for(j = i+1; j <= n; j++){
5        if(v[menor] > v[j]){
6          menor = j;
7        }
8      }
9      troca(V, i, menor); // custo: theta(1)
10   }
11 }
```

Note também que o fato da condicional da linha 5 ser verdadeira ou falsa dependendo do vetor de entrada não altera a complexidade assintótica do algoritmo, uma vez que o custo do trecho das linhas 5–7 é $\Theta(1)$, independentemente da condição ser verdadeira ou falsa. Além disso, a linha 3 e o trecho das linhas 9–11 tem custo $\Theta(1)$ pois não dependem de n. Portanto, em termos de custo, podemos analisar o algoritmo como:

SelectionSort

```
 1 void SelectionSort(int* V, int n){
 2   for(int i = 0; i < (n-1); i++){
 3     // theta(1)
 4     for(j = i+1; j <= n; j++){
 5       //
 6       // theta(1)
 7       //
 8     }
 9     // theta(1)
10   }
11 }
```

Note que o trecho das linhas 5–7 é o mais executado durante a execução do algoritmo. Considerando também que o custo das linhas 3 e 9 também é $\Theta(1)$, o custo assintótico do algoritmo pode ser obtido ao estimar o custo total do trecho das linhas 5–7.

SelectionSort

```
1  void SelectionSort(int* V, int n){
2    for(int i = 0; i < (n-1); i++){
3      // theta(1)
4      for(j = i+1; j <= n; j++){
5        //
6        // theta(1)
7        //
8      }
9      // theta(1)
10   }
11 }
```

i	Valores de j	Trecho l.5–7 executa
1	$2 \ldots n$	$n-1$ vezes
2	$3 \ldots n$	$n-2$ vezes
3	$4 \ldots n$	$n-3$ vezes
$\ldots$	$\ldots$	$\ldots$
n	$n-1 \ldots n$	1 vez

As linhas 5–7 estão dentro do laço interno, e a quantidade total de vezes que o laço interno executa depende diretamente do valor atual de i, a variável de controle do laço externo. A Tabela acima mostra os valores que j assume para cada valor de i, além de uma contagem de quantas vezes o trecho das linhas 5–7 executa para cada valor de i. Note que a coluna “Trecho l.5–7 executa” é simplesmente a quantidade de valores que j assume para aquele valor de i.

SelectionSort

```
1  void SelectionSort(int* V, int n){
2    for(int i = 0; i < (n-1); i++){
3      // theta(1)
4      for(j = i+1; j <= n; j++){
5        //
6        // theta(1)
7        //
8      }
9      // theta(1)
10   }
11 }
```

i	Valores de j	Trecho l.5–7 executa
1	$2 \dots n$	$n-1$ vezes
2	$3 \dots n$	$n-2$ vezes
3	$4 \dots n$	$n-3$ vezes
$\dots$	$\dots$	$\dots$
n	$n-1 \dots n$	1 vez

Portanto, somando todas vezes que o trecho das linhas 5–7 executa, para todo i, obtemos a quantidade total de vezes que o trecho das linhas 5–7 executa:

$$\begin{aligned}\sum_{i=1}^{n-1} i &= \frac{(n-1)(n-1+1)}{2} \\ &= \frac{(n-1)(n)}{2} \\ &= \frac{n^2-n}{2} \\ &= \Theta(n^2).\end{aligned}$$

SelectionSort

```
1  void SelectionSort(int* V, int n){
2    for(int i = 0; i < (n-1); i++){
3      // theta(1)
4      for(j = i+1; j <= n; j++){
5        //
6        // theta(1)
7        //
8      }
9      // theta(1)
10   }
11 }
```

i	Valores de j	Trecho l.5–7 executa
1	$2 \ldots n$	$n-1$ vezes
2	$3 \ldots n$	$n-2$ vezes
3	$4 \ldots n$	$n-3$ vezes
$\ldots$	$\ldots$	$\ldots$
n	$n-1 \ldots n$	1 vez

Assim, as linhas 5–7 executam $\Theta(n^2)$ vezes. Como o custo das linhas 5–7 é $\Theta(1)$, o custo total do algoritmo `SelectionSort` é

$$\underbrace{\Theta(n^2)}_{\text{exec. l. 5-7}} \cdot \underbrace{\Theta(1)}_{\text{l. 5-7}} = \Theta(n^2).$$

BubbleSort

```
1  void BubbleSort(int* V, int n){
2    int trocou = 1;
3    while(trocou){
4      trocou = 0;
5      for(int i = 0; i < n-1; i++){
6        if(v[i] > V[i+1]){
7          trocar(v, i, i+1); // custo: theta(1)
8          trocou = 1;
9        }
10     }
11   }
12 }
```

Análise: Por se tratar de um algoritmo de ordenação, o tamanho do problema, n, indica o número de elementos do vetor sendo ordenado. Note que o custo das linhas 2 e 4 e do trecho das linhas 6–9 é $\Theta(1)$, pois não dependem de n.

BubbleSort (Melhor Caso)

```
1  void BubbleSort(int* V, int n){
2    int trocou = 1;
3    while(trocou){
4      trocou = 0;
5      for(int i = 0; i < n-1; i++){
6        if(v[i] > V[i+1]){
7          trocar(v, i, i+1); // custo: theta(1)
8          trocou = 1;
9        }
10     }
11   }
12 }
```

Análise (Melhor Caso): O melhor caso acontece quando o vetor já está ordenado. Neste caso, o laço `while` é executado apenas uma vez. Como o vetor está ordenado, a condição da linha 6 nunca é verdadeira. Logo, a variável troca nunca recebe 1. Assim, o laço `while` executa apenas uma vez o trecho das linhas 4–10. O custo do trecho das linhas 4–10 é o custo total do `for` das linhas 5–10. O laço `for` repete $n - 1$ vezes as operações de tempo constante das linhas 6–9. Portanto, o custo do trecho das linhas 4–10 é $\Theta(n)$. Como no melhor caso o `while` executa apenas uma vez as linhas 4–10 que tem custo $\Theta(n)$, o custo do algoritmo neste caso é $\Theta(n)$.

BubbleSort (Pior Caso)

```
1  void BubbleSort(int* V, int n){
2    int trocou = 1;
3    while(trocou){
4      trocou = 0;
5      for(int i = 0; i < n-1; i++){
6        if(v[i] > V[i+1]){
7          trocar(v, i, i+1); // custo: theta(1)
8          trocou = 1;
9        }
10     }
11   }
12 }
```

Análise (Pior Caso): O pior caso acontece quando o vetor está ordenado em ordem decrescente. Neste caso, o laço externo executará $n+1$ vezes, uma vez para colocar o i-ésimo maior elemento em sua posição final, com uma passada extra pra determinar se houve uma troca. Desta forma, `while` executará $n+1$ vezes as linhas 4–10, de custo unitário $\Theta(n)$, conforme argumentado na análise do melhor caso. Desta forma, o custo do algoritmo no pior caso é dado por:

$$\underbrace{(n+1)}_{\text{reps. while}} \cdot \underbrace{\Theta(n)}_{\text{linhas 4-10}} = \Theta(n^2).$$

Trecho 7

```
for(int j = 1; j<=n; j++){
  f(n); // custo: theta(n)
  g(n); // custo: theta(lg(n))
}
```

Trecho 7

```
1 for(int j = 1; j<=n; j++){
2   f(n); // custo: theta(n)
3   g(n); // custo: theta(lg(n))
4 }
```

Análise: Ambas $f(n)$ e $g(n)$ são executadas a cada iteração. Logo, o custo de cada iteração é:

$$\Theta(n) + \Theta(\lg(n)) = \Theta(n).$$

Como o trecho das linhas 2 e 3 é executado n vezes pelo laço `for`, o custo do trecho é:

$$\underbrace{n}_{\text{exec. for}} \cdot \underbrace{\Theta(n)}_{\text{linhas 2-3}} = \Theta(n^2).$$

Trecho 8

```
a = 0;
for(int i = 0; i < n; i++){
  f(n); // custo: theta(n^2)
  for(int j = 0; j < n; j++){
    a++;
  }
}
```

Trecho 8

```
1 a = 0;
2 for(int i = 0; i < n; i++){
3   f(n); // custo: theta(n^2)
4   for(int j = 0; j < n; j++){
5     a++;
6   }
7 }
```

Análise: Por hipótese, o custo de $F(n)$ é $\Theta(n^2)$. A variável de controle do FOR das linhas 4–6 é independente do valor atual da variável de controle do FOR das linhas 2–7. Portanto, podemos analisá-los separadamente. A linha 5 tem custo unitário $\Theta(1)$ e executa n vezes. Portanto, o custo unitário do laço interno é $\Theta(1) \cdot n = \Theta(n)$. O custo de cada iteração do laço externo (trecho das linhas 3–6) é

$$\underbrace{\Theta(n^2)}_{\text{linha 3}} + \underbrace{\Theta(n)}_{\text{linhas 4-6}} = \Theta(n^2).$$

Trecho 8

```
1 a = 0;
2 for(int i = 0; i < n; i++){
3   f(n); // custo: theta(n^2)
4   for(int j = 0; j < n; j++){
5     a++;
6   }
7 }
```

Como o laço externo repete n vezes o trecho das linhas 3–6, o custo total do trecho é

$$\underbrace{n}_{\text{FOR ext.}} \cdot \underbrace{\Theta(n^2)}_{\text{linhas 3-6}} = \Theta(n^3).$$

Trecho 9

```
1 a = 0;
2 for(int i = 1; i <= n; i*=2){
3   f(n); // custo: theta(lg(n))
4   for(int j = 0; j < i; j++){
5     a++;
6   }
7 }
```

Trecho 9

```
1 a = 0;
2 for(int i = 1; i <= n; i*=2){
3   f(n); // custo: theta(lg(n))
4   for(int j = 0; j < i; j++){
5     a++;
6   }
7 }
```

Análise: Os valores que a variável de controle do laço FOR das linhas 4–6 assume depende diretamente do valor atual da variável de controle do FOR das linhas 2–7. Portanto, é necessário analisá-los conjuntamente. Além disso, temos 2 linhas críticas: a linha 3, que, por hipótese, tem custo unitário $\Theta(n)$ e a linha 5, que tem custo unitário $\Theta(1)$.

Trecho 9

```
1 a = 0;
2 for(int i = 1; i <= n; i*=2){
3   f(n); // custo: theta(lg(n))
4   for(int j = 0; j < i; j++){
5     a++;
6   }
7 }
```

i	Valores de j	Linha 5 executa
1	$0 \ldots 0$	1 vez
2	$0 \ldots 1$	2 vezes
4	$0 \ldots 3$	4 vezes
8	$0 \ldots 7$	8 vezes
$\ldots$	$\ldots$	$\ldots$
n	$0 \ldots n-1$	n vezes

A Tabela acima mostra quantas vezes a linha 5 é executada para cada valor de i. Notamos também que a variável i do laço externo assume os valores $1, 2, 4, 8, \ldots, n = 2^0, 2^1, 2^2, 2^3, \ldots, 2^k$, tal que $k \in \mathbb{Z}$. Assumindo que n é potência de 2, $2^k = n \Leftrightarrow k = \lg(n)$. Portanto, o laço externo executa $\lg(n)$ vezes.

Trecho 9

```
1 a = 0;
2 for(int i = 1; i <= n; i*=2){
3   f(n); // custo: theta(lg(n))
4   for(int j = 0; j < i; j++){
5     a++;
6   }
7 }
```

i	Valores de j	Linha 5 executa
1	$0 \ldots 0$	1 vez
2	$0 \ldots 1$	2 vezes
4	$0 \ldots 3$	4 vezes
8	$0 \ldots 7$	8 vezes
$\ldots$	$\ldots$	$\ldots$
n	$0 \ldots n-1$	n vezes

Somando todas as execuções da linha 5 (ao lado), temos que ela executa $\Theta(n)$ vezes.

$$\begin{aligned}\sum_{i=0}^{\lg(n)} 2^i &= 2^{\lg(n)+1} - 1 \\ &= 2 \cdot 2^{\lg(n)} - 1 \\ &= 2 \cdot n^{\lg(2)} - 1 \\ &= 2 \cdot n^1 - 1 \\ &= 2 \cdot n - 1 \\ &= \Theta(n)\end{aligned}$$

Trecho 9

```
1 a = 0;
2 for(int i = 1; i <= n; i*=2){
3   f(n); // custo: theta(lg(n))
4   for(int j = 0; j < i; j++){
5     a++;
6   }
7 }
```

i	Valores de j	Linha 5 executa
1	$0 \ldots 0$	1 vez
2	$0 \ldots 1$	2 vezes
4	$0 \ldots 3$	4 vezes
8	$0 \ldots 7$	8 vezes
$\ldots$	$\ldots$	$\ldots$
n	$0 \ldots n-1$	n vezes

Como o custo unitário da linha 5 é $\Theta(1)$ e ela executa $\Theta(n)$ vezes, seu custo total é $\Theta(n)$. Como a linha 3 executa $\lg(n)$ vezes e seu custo unitário é $\Theta(n)$, seu custo total é $\Theta(n \lg(n))$. Assim, o custo do trecho é

$$\underbrace{\Theta(n \lg(n))}_{\text{linha } 3} + \underbrace{\Theta(n)}_{\text{linhas } 4\text{--}6} = \Theta(n \lg(n)).$$