PRINCIPAIS FATORES ENVOLVIDOS NA ANÁLISE DE COMPLEXIDADE DE ALGORITMOS

Fatores para análise de algoritmos

Algoritmos:

Conjunto e estrutura de operações destinada a geração de resultados esperados.

Dependente do conjunto de dados de entrada (tanto em sua distribuição como em seu tamanho).

Associados à classes de problemas, com diversidade de opções de utilização (em geral existem várias soluções, vários algoritmos para o mesmo problema).

Objetivo de analisar algoritmos:

Análise proporciona oportunidade de projetar algoritmos eficientes.

Mecanismos de análise permitem desenvolver um algoritmo e depois disso :

- Avaliar sua eficiência
- Comparar com outros algoritmos

Mecanismos de análise apoiam a seleção de metodologias adequadas para concepção de algoritmos eficientes

Aspectos analisados, em geral:

Espaço de armazenamento utilizado

- Quantidade de memória, seja principal ou secundária.

Capacidade de processamento necessária

- Tempo de processamento envolvido nos casos médios e no pior caso

Fatores para análise de algoritmos

Aspectos importantes, mas que não serão tratados neste momento:

- Verificação da correção do algoritmo
  - Prova de que o processamento é correto, para qualquer um dos valores de entrada.

Aspectos importantes na escolha de medidas:

Utilização do tempo absoluto (minutos, milissegundos) não é adequada, pois varia de acordo com:

- Processador e sua velocidade
- Quantidade de processadores
- Quantidade e tipo de memória
- Sistema operacional
- Quantidade de processos simultâneos no sistema operacional
- Compilador, linguagem
- .....

Aspectos importantes na escolha de medidas:

Necessário uso de medidas independentes de aspectos de hardware e software.

Utilização de:

- Quantidade de instruções relevantes consideradas no algoritmo.
- Dimensão do volume de dados tratado.

Impactos associados com instâncias do problema:

Volume de dados e a sua configuração podem afetar o resultado obtido pelo algoritmo.

Por exemplo: na busca em sequência por um valor em um vetor com um total de n valores aleatórios, a posição do valor desejado irá gerar resultados diferenciados. O número de operações de comparação pode variar entre 1 e n.

Este fato demanda a utilização de:

- Análise de melhor caso
- Análise do caso médio
- Análise do pior caso

Fatores para análise de algoritmos

Abordagem inicial para análise de algoritmos

Estimativa do tempo de execução, com base em aspectos do código:

- Tipo das instruções envolvidas;
- Freqüência de ocorrência das instruções.

Fatores para análise de algoritmos

Abordagem inicial (1)

Observando o código a seguir, identificam-se:

- Instruções de atribuição de valores;
- Instruções de multiplicação;
- Instruções de soma;
- Instruções de repetições (laço com while)

Código fonte

- Avaliação do tempo de execução:

Código	Quantidade	Operações
1. int aux1 = 1;	1	op1
2. int aux2 = 1;	1	op1
3. aux1 = aux1 * aux2;	1	op2 e op1
4. aux2 = aux2 + 1;	1	op3 e op1

- Tempo de execução resultante:

$$T = 4\ op1 + 1\ op2 + 1\ op3$$

Fatores para análise de algoritmos

Abordagem inicial (2)

Após a identificação das instruções, estas podem ser indicadas (coluna “operações”):

- Instruções de atribuição de valores: op1;
- Instruções de multiplicação: op2;
- Instruções de soma: op3;

Código fonte - Operações

CÓDIGO FONTE

```
public void teste(int n){
      int aux1=1;
      int aux2=1;
      int i=0;
      while (i < n) {
          aux1 = aux1 * aux2;
          aux2 = aux2 + 1;
          i = i + 1;
      }
      i = 0;
}
```

Fatores para análise de algoritmos

Abordagem inicial (3)

Agora, as instruções, estas podem ser quantificadas (coluna "quantidade").

- Utiliza-se o parâmetro "n" para indicar a quantidade de repetições;
- Utiliza-se o número de vezes quando a instruções ocorre apenas uma vez.

Código fonte – Quantidade - Operações

Código fonte	Quantidade	Operações
public void teste(int n){		
int aux1=1;	1	op1
int aux2=1;	1	op1
int i=0;	1	op1
while (i < n) {	n+1	op4
aux1 = aux1 * aux2;	n	op1 e op2
aux2 = aux2 + 1;	n	op1 e op3
i = i + 1;	n	op1 e op3
}		
i = 0;	1	op1
}		

Fatores para análise de algoritmos

Abordagem inicial (4)

A partir das quantidades e tipos, uma expressão pode ser montada, para indicar a expectativa de tempo de execução.

- Veja a expressão T(n), agrupando os termos relacionados com as operações.

Código fonte – Quantidade - Operações

Código fonte	Quantidade	Operações
public void teste(int n){		
int aux1=1;	1	op1
int aux2=1;	1	op1
int i=0;	1	op1
while (i < n) {	n+1	op4
aux1 = aux1 * aux2;	n	op1 e op2
aux2 = aux2 + 1;	n	op1 e op3
i = i + 1;	n	op1 e op3
}		
i = 0;	1	op1
}		

T(n) = 4 op1 + (n+1) op4 + 3n op1 + n op2 +2n op3

Código Fonte – Quantidade - Operações

Código fonte	Quantidade	Operações
`public void teste(int n){`		
`  int aux1=1;`	1	op1
`  int aux2=1;`	1	op1
`  int i=0;`	1	op1
`  while (i < n) {`	n+1	op4
`    aux1 = aux1 * aux2;`	n	op1 e op2
`    aux2 = aux2 + 1;`	n	op1 e op3
`    i = i + 1;`	n	op1 e op3
`  }`		
`  i = 0;`	1	op1
`}`		

T(n) = 4 op1 + (n+1) op4 + 3n op1 + n op2 +2n op3

Código fonte – Quantidade - Operações

Código fonte	Quantidade	Operações
`public void teste(int n){`		
`int aux1=1;`	1	op1
`int aux2=1;`	1	op1
`int i=0;`	1	op1
`while (i < n) {`	n+1	op4
`aux1 = aux1 * aux2;`	n	op1 e op2
`aux2 = aux2 + 1;`	n	op1 e op3
`i = i + 1;`	n	op1 e op3
`}`		
`i = 0;`	1	op1
`}`		

T(n) = 4 op1 + (n+1) op4 + 3n op1 + n op2 +2n op3

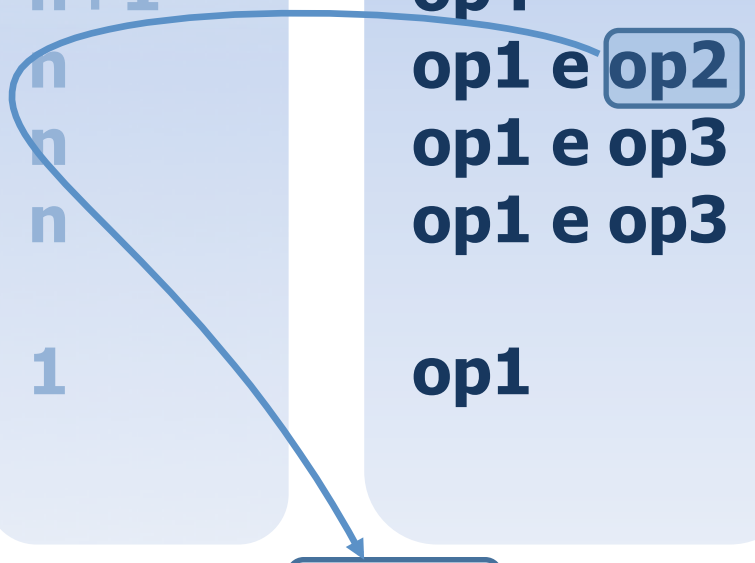
Código fonte – Quantidade - Operações

Código fonte	Quantidade	Operações
`public void teste(int n){`		
`int aux1=1;`	1	op1
`int aux2=1;`	1	op1
`int i=0;`	1	op1
`while (i < n) {`	n+1	op4
`aux1 = aux1 * aux2;`	n	op1 e op2
`aux2 = aux2 + 1;`	n	op1 e op3
`i = i + 1;`	n	op1 e op3
`}`		
`i = 0;`	1	op1
`}`		

T(n) = 4 op1 + (n+1) op4 + 3n op1 + n op2 +2n op3

Código fonte – Quantidade -Operações

Código fonte	Quantidade	Operações
`public void teste(int n){`		
`int aux1=1;`	1	op1
`int aux2=1;`	1	op1
`int i=0;`	1	op1
`while (i < n) {`	n+1	op4
`aux1 = aux1 * aux2;`	n	op1 e op2
`aux2 = aux2 + 1;`	n	op1 e op3
`i = i + 1;`	n	op1 e op3
`}`		
`i = 0;`	1	op1
`}`		



T(n) = 4 op1 + (n+1) op4 + 3n op1 + n op2 +2n op3

Código fonte – Quantidade - Operações

Código fonte	Quantidade	Operações
`public void teste(int n){`		
`int aux1=1;`	1	op1
`int aux2=1;`	1	op1
`int i=0;`	1	op1
`while (i < n) {`	n+1	op4
`aux1 = aux1 * aux2;`	n	op1 e op2
`aux2 = aux2 + 1;`	n	op1 e op3
`i = i + 1;`	n	op1 e op3
`}`		
`i = 0;`	1	op1
`}`		

T(n) = 4 op1 + (n+1) op4 + 3n op1 + n op2 + 2n op3

Fatores para análise de algoritmos

Abordagem inicial (5)

A partir da expressão obtida, retiram-se os termos que possuem pouca expressão no tempo de execução.

- O exemplo, apenas os termos associados ao valor de entrada “n” são importantes.

$$T(n) = 4\ op1 + (n+1)\ op4 + 3n\ op1 + n\ op2 + 2n\ op3$$

$$T(n) = 4\ op1 + op4 + n(3op1 + op2 + 2op3 + op4)$$

Tempo dependente de “n”

T(n) = 4 op1 + op4 + n(3op1 + op2 + 2op3 + op4)
Tempo dependente de "n"

Análise da variação dos valores de "n":
a) "n" = 1
instruções individuais, linhas 2, 3, 4 e 10 : 4 instruções
instruções afetadas pelo valor de "n", que estão nas linhas 5, 6, 7 e 8: 7 instruções.
Instruções fora do laço : 36% do tempo de execução
Instruções dentro do laço: 67% do tempo de execução

b) "n" = 10,
instruções individuais, linhas 2, 3, 4 e 10 : 4 instruções
Instruções dentro do laço : 70 instruções.
Instruções dentro do laço: 94% do tempo de execução

c) "n" = 100
Instruções dentro do laço: 99,4% do tempo de execução

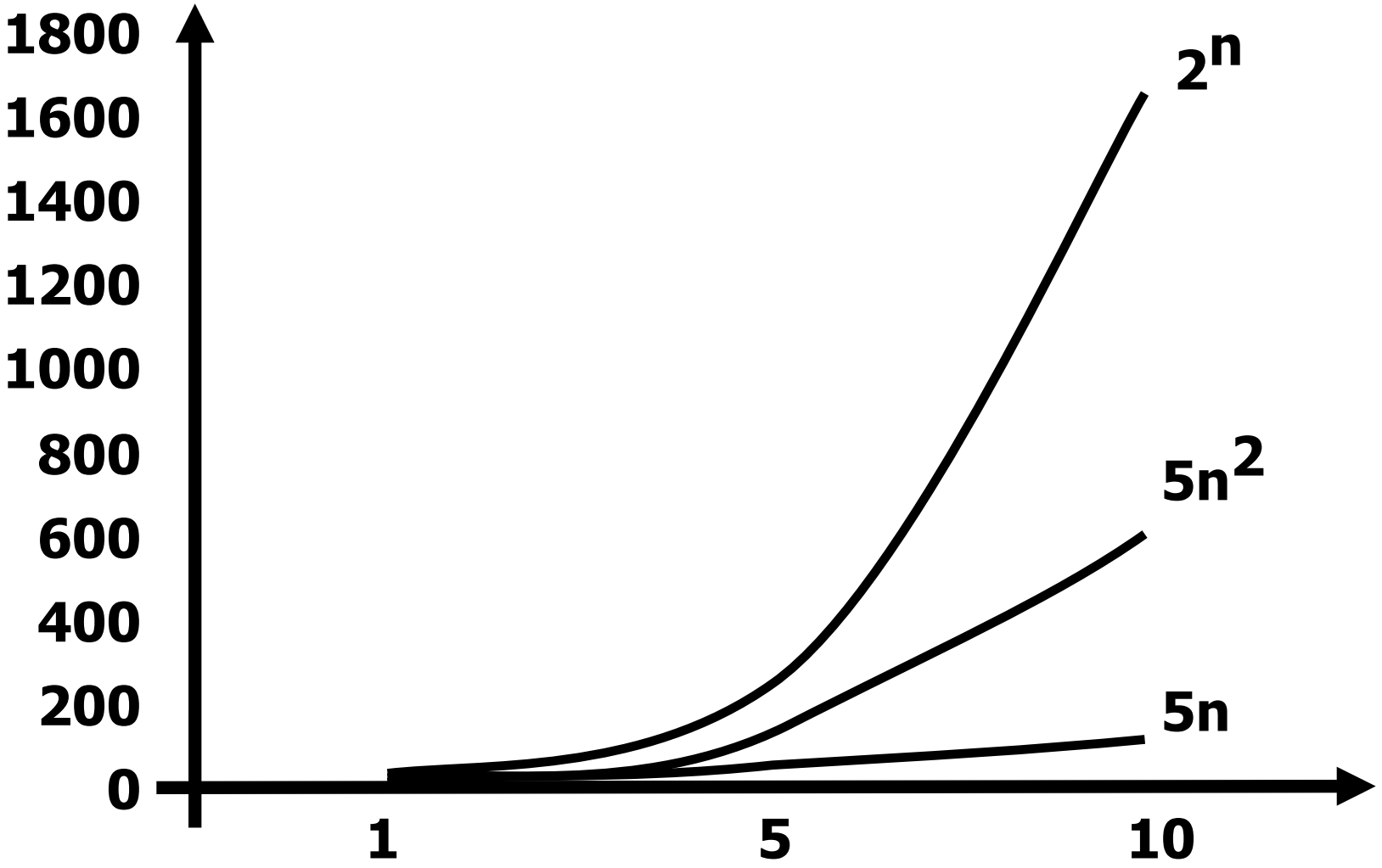
Fatores para análise de algoritmos

Abordagem inicial (6)

Utilização do resultado obtido para avaliar uma tendência de crescimento e assim possibilitar comparações.

Observe as diferenças no crescimento do tempo, para os casos: $T(n)=5n$, $T(n)=5n^2$ e $T(n)=2^n$

Fatores para análise de algoritmos



Análise do crescimento:

Variação de entrada para:

Função linear ($T(n) = 5n$)

Função quadrática ($T(n) = 5n^2$)

Função exponencial ($T(n) = 2^n$)

Análise do crescimento:

Crescimento diferenciado de acordo com valores de entrada.

Os itens “a” e “b” indicam o ponto em que as funções demonstram resultados diferenciados da sua expectativa inicial

	1	2	3	4	5	6	7	8	9
$5n$	5	10	15	20	25	30	35	40	45
$5n^2$	5	20	45	80	125	180	245	320	405
2^n	2	4	8	16	32	64	128	256	512

a) $2^n > 5n$

b) $2^n > 5n^2$

Análise do crescimento:

Avaliação do valor de entrada (n) supera os demais termos.

Quando n → ∞ diz-se análise do comportamento assintótico

- $f_{algoritmo1}(n) = 2n^2 + 5n$
- $f_{algoritmo2}(n) = 500n + 500$
- $f_{algoritmo3}(n) = 5n^n + 30$
- $f_{algoritmo4}(n) = 5n^n + 3000n$

Análise assintótica

Consideram-se notações que identificam tendências de crescimento, utilizadas para comparação e avaliação.

Notações mais usadas:

ômega (Ω) – melhor caso

theta (Θ) – caso médio

O (ou “big O”) – pior caso

Fatores para análise de algoritmos