

Algoritmos e Estruturas de Dados I



Adaptado dos slides da
Prof^a. Renata Oliveira

Objetivos

- Buscar a eficiência dos algoritmos
 - Espaço de memória: no de variáveis, no e tamanho das estruturas de dados utilizadas
 - Tempo: no de ações elementares realizadas pelo processador
- Mostrar princípios básicos e técnicas de análise de algoritmos

Objetivos

- Despertar para questões como:
 - Quão bom é um algoritmo?
 - Existe uma forma melhor de projetá-lo?
 - Qual é o custo de se usar um dado algoritmo para resolver um problema específico?
 - Qual é o algoritmo de menor custo possível que possa resolver um problema particular?
 - Existem algoritmos similares para resolver o problema?
 - Que estrutura de dados é mais adequada ao problema?

Introdução

“Um conhecimento de técnicas de projeto poderá ajudar a criar novos algoritmos, mas sem as ferramentas da análise de algoritmos não há como determinar a qualidade dos resultados”

Motivação

- Os benefícios de caráter prático podem, só eles, serem suficientes para justificar o estudo dessa disciplina.

“Algoritmos diferentes para resolver o mesmo problema podem diferir dramaticamente em eficiência. Essa diferença pode ser mais significante que a diferença entre um supercomputador e um desktop”

Motivação

- Exemplo: Seja a seguinte situação: ordenar um conjunto de um milhão de elementos
- Caso 1:
 - Supercomputador capaz de processar de 100 milhões de instruções/segundo;
 - Executando Ordenação por Inserção Direta;
 - Melhor programador do mundo programou o algoritmo de inserção usando linguagem de máquina;
 - Programa resultante requer $2*n^2$ instruções para ordenar n elementos.

Motivação

- Caso 2:
 - Desktop capaz de processar de 1 milhão de instruções por segundo
 - Executando Ordenação por Heapsort
 - Programador medíocre programou o algoritmo de heapsort usando linguagem de alto nível, com um compilador não muito eficiente
 - Código resultante usa $50 * n * \lg n$ instruções do computador para ordenar n elementos

Motivação

- Calcule o tempo de execução para cada caso:

- Caso 1: Supercomputador

$$T1 = \frac{2 * (10^6)^2 \text{inst}}{100 * 10^6 \text{inst/seg}} = 5,6 \text{ horas}$$

- Caso 2: PC doméstico

$$T2 = \frac{50 * 10^6 * \log(10^6) \text{inst}}{10^6 \text{inst/seg}} = 16,67 \text{ min}$$

Motivação

- “A tecnologia não evoluiu só em hardware, mas em software também”
- Caso 2 aprox. 20 vezes mais rápido que Caso 1.

Complexidade de Tempo

- O custo de aplicar um algoritmo pode ser medido:
 1. Executando o programa em um computador e medir o tempo de execução.
 - Ex: no de miliseg. Em um IBM Power com 128Gb de memória usando AIX, Linux, ...
 - Concluímos que ele é rápido, lento, muito rápido, muito lento...

Complexidade de Tempo

2. Usar um modelo matemático baseado em uma linguagem de programação ou em um computador idealizado

- O conjunto de operações e o custo de cada uma tem de ser fornecido
- Podemos ignorar o custo de algumas das operações envolvidas. Por exemplo para algoritmos de ordenação consideramos o no de comparações e trocas entre os elementos do conjunto a ser ordenado.

troca

$\left. \begin{array}{l} aux = a \\ a = b \\ b = aux \end{array} \right\}$

Complexidade de Tempo

- Podemos concluir que:
 - O tempo de execução em um computador particular não é interessante.
 - Muito mais interessante é uma comparação relativa entre algoritmos.

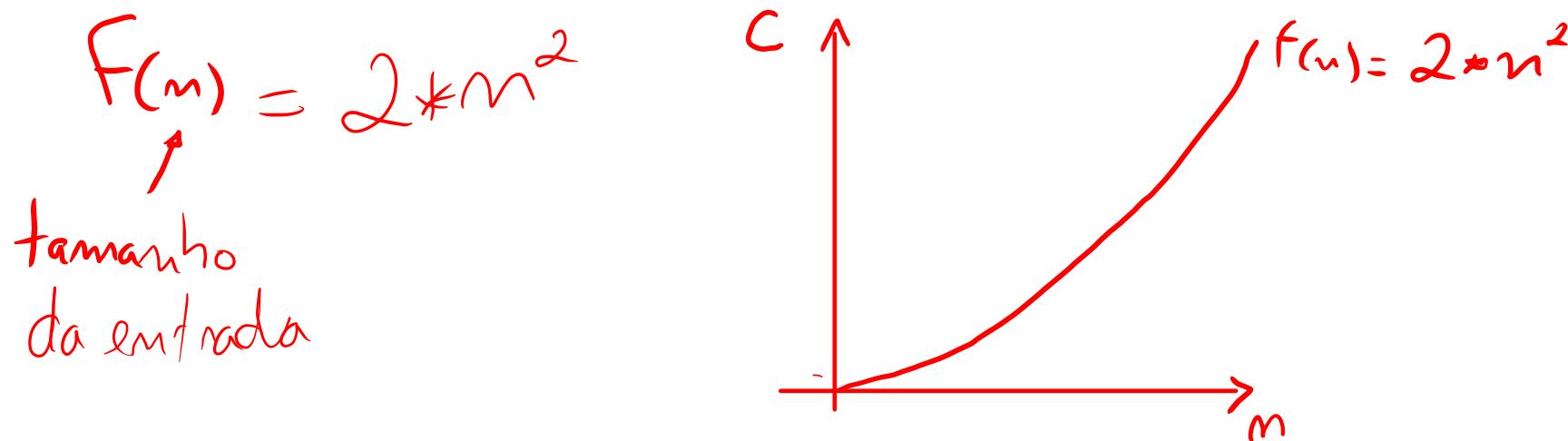
Complexidade de Tempo

Supondo que:

- As operações são todas executadas sequencialmente. A execução de toda e qualquer operação toma uma unidade de tempo. A memória do computador é infinita.
- Assim nos sobram duas grandezas:
 - Tempo = número de operações executadas
 - Quantidade de dados de entrada.

Complexidade de Tempo

- Podemos expressar de forma abstrata a eficiência de um algoritmo, descrevendo o seu tempo de execução como uma função do tamanho do problema (quantidade de dados). Isto é chamado de complexidade de tempo



Exemplo: Dois algoritmos de ordenação

```
1-begin
2- for i := 2 to n do
3-   for j := n downto i do
4-     if ( a[j-1] > a[j] )
5-       begin
6-         x = a[j-1];
7-         a[j-1] = a[j];
8-         a[j] = x;
9-       end;
10-end.
```

```
1- begin
2-   for i = 1 to n-1
3-     begin
4-       k = i;
5-       x = a[i];
6-       for j = i+1 to n
7-         begin
8-           if (a[j] < x)
9-             begin
10-               k = j;
11-               x = a[k];
12-             end
13-         end
14-         a[k] = a[i];
15-         a[i] = x;
16-       end
17-end
```

Exemplo: Ordenação A

- Raciocínio:

- (quadro)

$$L \rightarrow m - L = 2 \rightarrow m$$

+ tempo: comparações

- Algoritmo:

```
1-begin  
2-  for i := 1 to n-1 do  
3-      for j := n-1 downto i do  
4-          if ( a[j-1] > a[j] )  
5-              begin  
6-                  x = a[j-1];  
7-                  a[j-1] = a[j];  
8-                  a[j] = x;  
9-              end;  
10-         for-end  
11-     for-end  
10-end.
```

$$F(n) = \frac{m^2 - m}{2}$$

Memória: movimentações

Tnoca ou 3mov.

$$m \rightarrow \emptyset = m + L$$

$$m \rightarrow L = m$$

$$m \rightarrow 2 = m - 1 = m - 2 + L$$

$$m \rightarrow 3 = m - 2 = m - 3 + L$$

$$m \rightarrow i = m - i + L$$

$$\emptyset \rightarrow L \emptyset = LL$$

$$1 \rightarrow L \emptyset = LO$$

$$2 \rightarrow L \emptyset = OJ$$

$$3 \rightarrow L \emptyset = 8$$

Exemplo: Ordenação A

- Análise por número de comparações:

$$[4] \quad C(n) = \underline{1}$$

$$[3,10] \quad C(n) = n - i + 1$$

$$[2,11] \quad C(n) = \sum_{\substack{i=2 \\ i=3}}^n (n - i + 1) = \sum_{l=1}^{n-1} (n - l + 1)$$

$$C(n) = \underbrace{(n-1) + (n-2) + \dots + 1}_{\downarrow} = \frac{n(n+1)}{2} - n = \frac{n^2 + n}{2} - n = \frac{n^2 - n}{2}$$

$$C(n) = \frac{(n-1)+1}{2} * (n-1)$$

$$C(n) = \frac{n^2 - n}{2} = O(n^2)$$

$$\left. \begin{aligned} & 1 + 2 + 3 + \dots + (n-2) + (n-1) + n = \\ & \frac{n(n+1)}{2} \end{aligned} \right\}$$

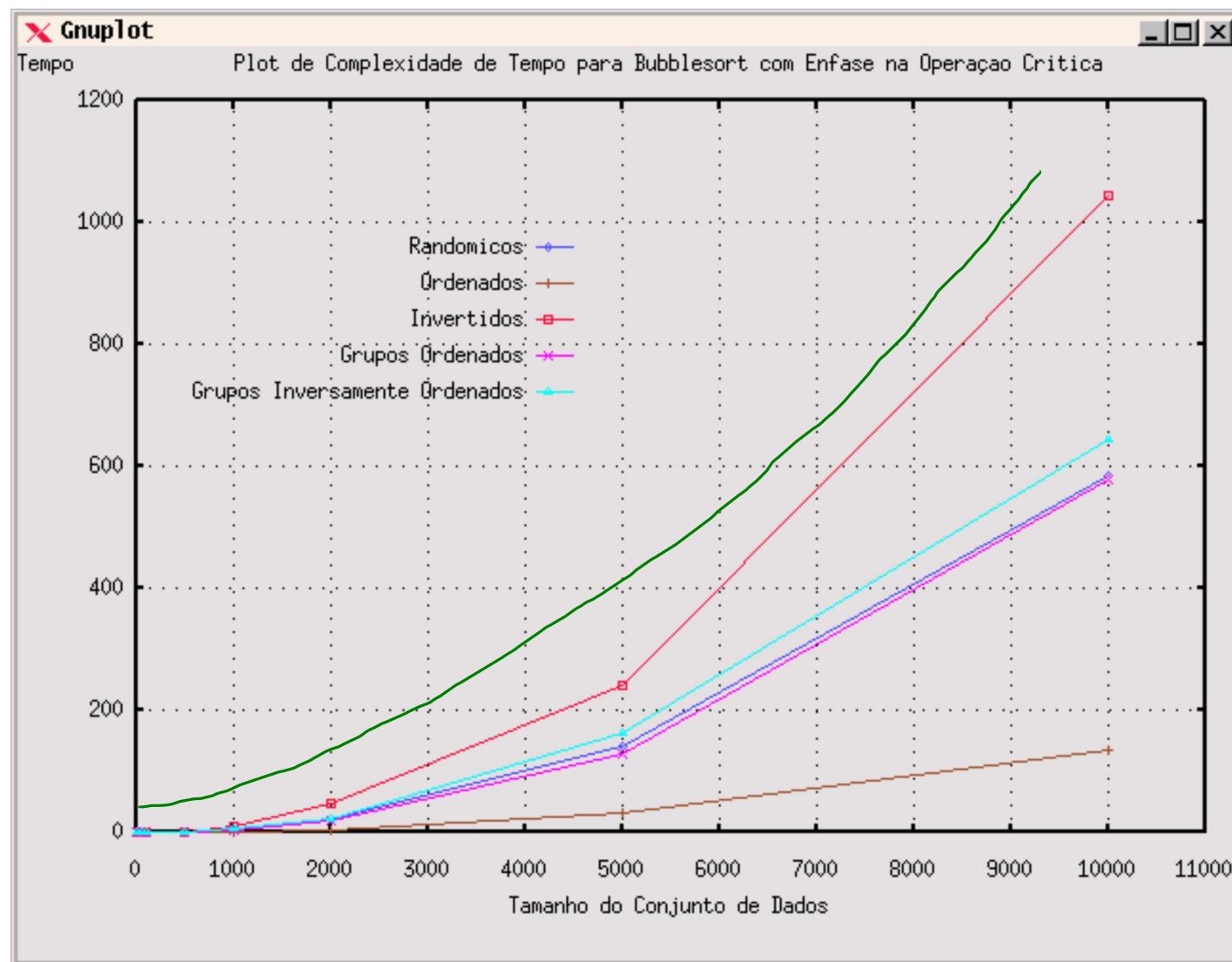
Exemplo: Ordenação A

- Análise por número de trocas:
 - Melhor caso: Vetor já ordenado, não realiza troca = $O(1)$
 - Pior Caso: Ordem decrescente

$$C(n) = \sum_{i=2}^n n - i + 1 = \cancel{O(n^2)} = \frac{n^2 - n}{2} = O(n^2)$$

- Caso Médio:
 - $O(n^2)$ $\frac{n^2 - n}{4} =$

Gráfico A



Exemplo: Ordenação B

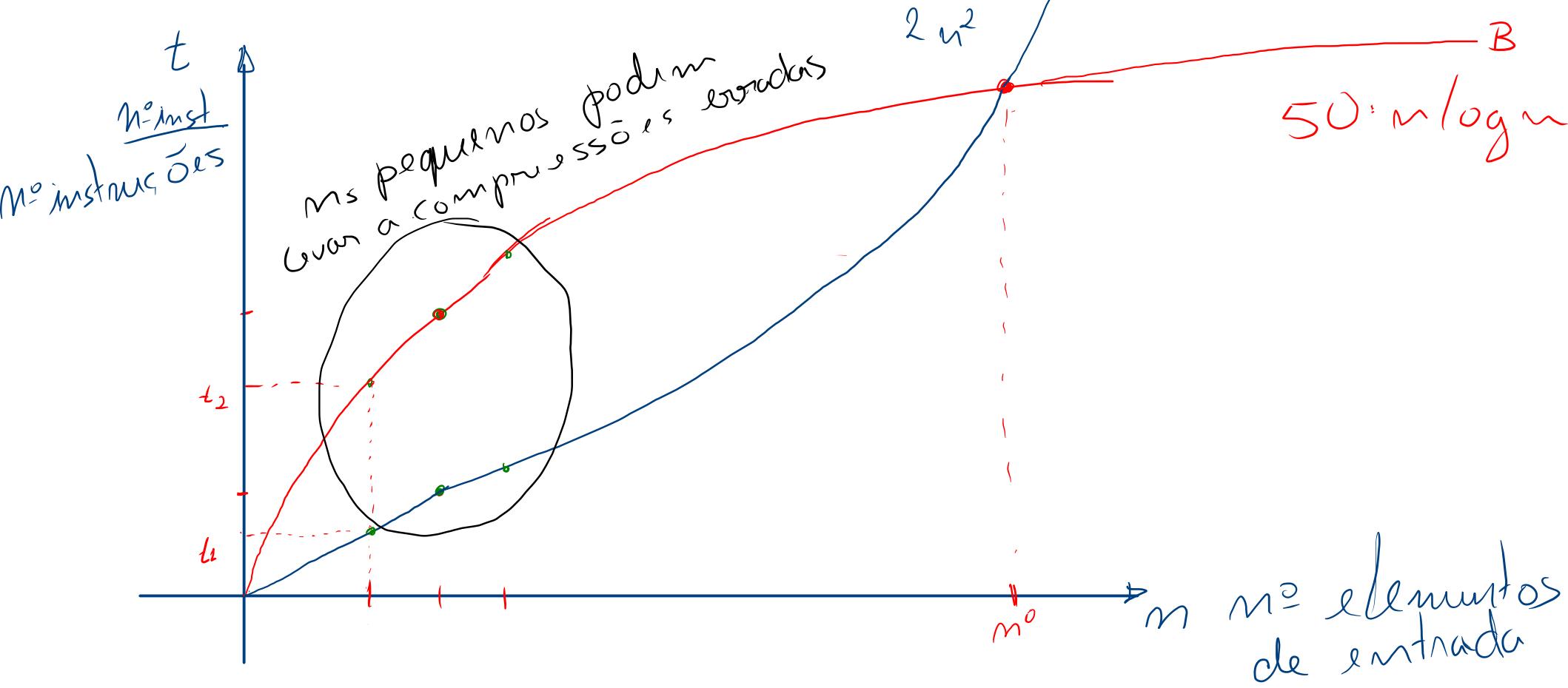
- Raciocínio/Algoritmo:

```
1- begin
2-   for i = 1 to n-1
3-     begin
4-       k = i;
5-       x = a[i];
6-       for j = i+1 to n
7-         begin
8-           if (a[j] < x)
9-             begin
10-               k = j;
11-               x = a[k];
12-             end
13-           end
14-           a[k] = a[i];
15-           a[i] = x;
16-         end
17- end
```

Exemplo: Ordenação B

- Neste algoritmo o número de vezes que a comparação ($a[j] < x$) é executada é expresso por $(n-1)+(n-2)+\dots+2+1 = (n/2) \times (n-1)$.
- O número de trocas $a[k] = a[i]$; $a[i] = x$ é realizado no pior caso, onde o vetor está ordenado em ordem inversa, somente $n-1$ vezes, num total de $2 \times (n-1)$. $O(n)$

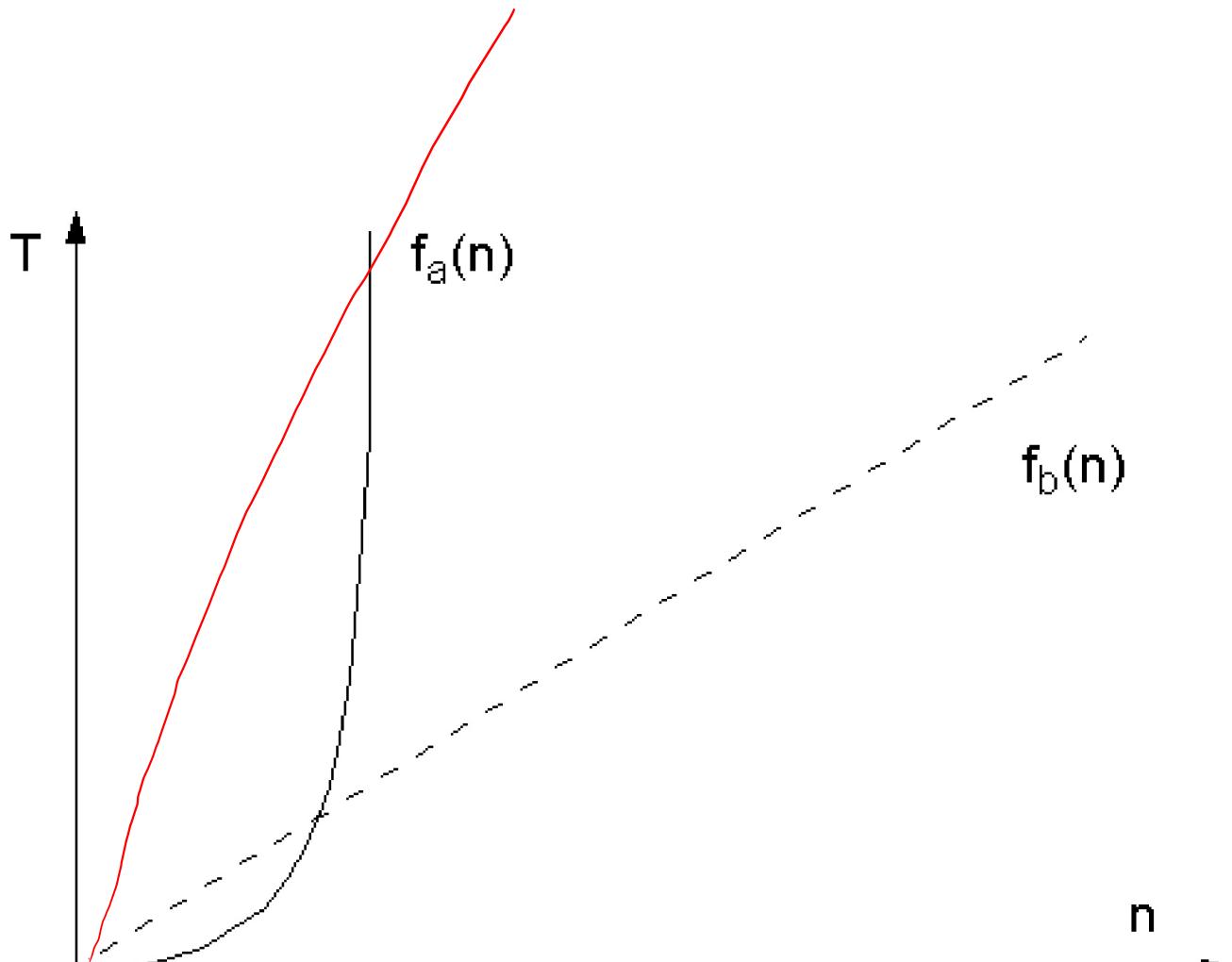
$$C(n) = 2^* n^2$$



Interpretação

- Interesse: Comportamento assintótico da função, ou seja, como a função varia com a variação de n .
- Razão: Para mim é interessante saber como o algoritmo se comporta com uma quantidade de dados realística para o meu problema e o que acontece quando eu vario esses dados.

Gráfico de A x B



Comportamento Assintótico

Interpretação

- Exemplo: Se a complexidade de A um é expressa por $f_a(n) = n^2$ e a de B por $f_b(n) = \cancel{100}n$.
 - Isto significa que A cresce quadraticamente (uma parábola) e que b cresce linearmente (embora seja uma reta bem inclinada).

n	$f_a(n)$	$f_b(n)$	f_b/f_a
10	100	1000	10
20	400	2000	5
30	900	3000	2,5

Interpretação

- Se eu uso esses algoritmos para um conjunto de 30 elementos, o segundo com $T_b=3000$ é pior do que o primeiro com $T_a=900$.
- Se eu os uso para um conjunto de 30.000 elementos, porém, terei $T_a=900.000.000$ e $T_b=3.000.000$.
- Isto porque o comportamento assintótico dos dois é bem diferente.

Áreas de Abrangência

- Cinco áreas importantes e distintas da pesquisa:
 - Como projetar um algoritmo: o ato de criar um algoritmo é uma arte que nunca poderá ser totalmente automatizada.
 - Como expressar algoritmos: a programação estruturada, por exemplo, tem como finalidade a expressão clara e concisa de um algoritmo em linguagem de programação.
 - Como verificar a corretude de um algoritmo: é o processo no qual se verifica se o algoritmo fornece saída correta para toda entrada de dados possível. Esta verificação deve ser feita independentemente do código de programação usado na implementação.

Áreas de Abrangência

- Como analisar algoritmos: refere-se ao processo de determinar o tempo e a memória de um computador que o algoritmo irá requerer. Permite fazer um julgamento qualitativo de um algoritmo sobre outro.
- Como testar um programa: consiste em duas fases – depuração e desempenho.
 - Depuração: consiste em executar o programa sobre um conjunto de dados, para verificar se ele funciona corretamente. É bom lembrar que “uma prova de corretude vale mais que mil testes”.
 - Desempenho: é o processo de executar um programa correto sobre um conjunto de dados para medir o tempo e o espaço gastos. A medida de desempenho comumente é chamada de **BATERIA DE TESTES**.

Exemplos de contribuições da Análise de Algoritmos.

- Problema
 - Criptografia
 - Pesquisa em textos
 - Acesso a banco de dados
 - Otimização
- Problema Abstrato
 - Fatoração de inteiros
 - Casamento de padrão
 - Pesquisa e Ordenação
 - NP-Completo