Eficiência de algoritmos e programas

Aula 3

Diego Padilha Rubert

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação II

Conteúdo da aula

Motivação

- Tenho mais de um algoritmo/programa para solucionar um problema computacional. Qual devo escolher?
  - Aquele que gasta *menos* tempo?
  - Aquele que gasta *menos* memória?
  - Aquele que faz *menos* comunicação entre processos?
  - Aquele que usa/acessa *menos* portas lógicas?

Motivação

- Tenho mais de um algoritmo/programa para solucionar um problema computacional. Qual devo escolher?
  - Aquele que gasta *menos* tempo?
  - Aquele que gasta *menos* memória?
  - Aquele que faz *menos* comunicação entre processos?
  - Aquele que usa/acessa *menos* portas lógicas?

Motivação

- Tenho mais de um algoritmo/programa para solucionar um problema computacional. Qual devo escolher?
  - Aquele que gasta *menos* tempo?
  - Aquele que gasta *menos* memória?
  - Aquele que faz *menos* comunicação entre processos?
  - Aquele que usa/acessa *menos* portas lógicas?

Motivação

- Tenho mais de um algoritmo/programa para solucionar um problema computacional. Qual devo escolher?
  - Aquele que gasta *menos* tempo?
  - Aquele que gasta *menos* memória?
  - Aquele que faz *menos* comunicação entre processos?
  - Aquele que usa/acessa *menos* portas lógicas?

Motivação

- Tenho mais de um algoritmo/programa para solucionar um problema computacional. Qual devo escolher?
  - Aquele que gasta *menos* tempo?
  - Aquele que gasta *menos* memória?
  - Aquele que faz *menos* comunicação entre processos?
  - Aquele que usa/acessa *menos* portas lógicas?

Algoritmos e programas

- Algoritmo ou programa é uma sequência bem definida de passos (descritos em uma linguagem de programação específica) que transforma um conjunto de valores – a entrada – em um outro conjunto de valores – a saída
- Algoritmo é uma ferramenta para solucionar um problema computacional

Algoritmos e programas

- Algoritmo ou programa é uma sequência bem definida de passos (descritos em uma linguagem de programação específica) que transforma um conjunto de valores – a entrada – em um outro conjunto de valores – a saída
- Algoritmo é uma ferramenta para solucionar um problema computacional

Algoritmos e programas

Problema da busca

Dado um número inteiro n, com $1 \leqslant n \leqslant 100$, um conjunto C de n números inteiros e um número inteiro x, verificar se x encontra-se no conjunto C

Algoritmos e programas

- Se um programa pára com a resposta correta então dizemos que o programa é correto
- Um programa correto soluciona o problema computacional associado
- Um programa incorreto pode sequer parar, para alguma entrada, ou pode parar mas com uma resposta indesejada

Algoritmos e programas

- Se um programa pára com a resposta correta então dizemos que o programa é correto
- Um programa correto soluciona o problema computacional associado
- Um programa incorreto pode sequer parar, para alguma entrada, ou pode parar mas com uma resposta indesejada

Algoritmos e programas

- Se um programa pára com a resposta correta então dizemos que o programa é correto
- Um programa correto soluciona o problema computacional associado
- Um programa incorreto pode sequer parar, para alguma entrada, ou pode parar mas com uma resposta indesejada

Algoritmos e programas

```
#include <stdio.h>
#define MAX 100
int main(void)
{
   int n, i, C[MAX], x;

   scanf("%d", &n);
   for (i = 0; i < n; i++)
      scanf("%d", &C[i]);
   scanf("%d", &x);

   for (i = 0; i < n && C[i] != x; i++)
      ;
   if (i < n)
      printf("%d está na posição %d de C\n", x, i);
   else
      printf("%d não pertence ao conjunto C\n", x);
   return 0;
}
```

Análise de algoritmos

- Fixar um modelo de computação: máquina de acesso aleatório
- Ferramentas matemáticas para expressar o comportamento de um algoritmo para uma dada entrada
- Um algoritmo depende do número de elementos fornecidos na entrada
  - Procurar um elemento x em um conjunto C com milhares de elementos certamente gasta mais tempo que em um conjunto C com 3 elementos
  - Mesmo para dois conjuntos diferentes mas com a mesma quantidade de elementos, uma busca pode ser mais demorada que a outra

Análise de algoritmos

- Fixar um modelo de computação: máquina de acesso aleatório
- Ferramentas matemáticas para expressar o comportamento de um algoritmo para uma dada entrada
- Um algoritmo depende do número de elementos fornecidos na entrada
  - Procurar um elemento x em um conjunto C com milhares de elementos certamente gasta mais tempo que em um conjunto C com 3 elementos
  - Mesmo para dois conjuntos diferentes mas com a mesma quantidade de elementos, uma busca pode ser mais demorada que a outra

Análise de algoritmos

- Fixar um modelo de computação: máquina de acesso aleatório
- Ferramentas matemáticas para expressar o comportamento de um algoritmo para uma dada entrada
- Um algoritmo depende do número de elementos fornecidos na entrada
  - Procurar um elemento x em um conjunto C com milhares de elementos certamente gasta mais tempo que em um conjunto C com 3 elementos
  - Mesmo para dois conjuntos diferentes mas com a mesma quantidade de elementos, uma busca pode ser mais demorada que a outra

Análise de algoritmos

- Fixar um modelo de computação: máquina de acesso aleatório
- Ferramentas matemáticas para expressar o comportamento de um algoritmo para uma dada entrada
- Um algoritmo depende do número de elementos fornecidos na entrada
  - Procurar um elemento x em um conjunto C com milhares de elementos certamente gasta mais tempo que em um conjunto C com 3 elementos
  - Mesmo para dois conjuntos diferentes mas com a mesma quantidade de elementos, uma busca pode ser mais demorada que a outra

Análise de algoritmos

- Fixar um modelo de computação: máquina de acesso aleatório
- Ferramentas matemáticas para expressar o comportamento de um algoritmo para uma dada entrada
- Um algoritmo depende do número de elementos fornecidos na entrada
  - Procurar um elemento x em um conjunto C com milhares de elementos certamente gasta mais tempo que em um conjunto C com 3 elementos
  - Mesmo para dois conjuntos diferentes mas com a mesma quantidade de elementos, uma busca pode ser mais demorada que a outra

Análise de algoritmos

- O tempo gasto por um algoritmo cresce com o tamanho da entrada, isto é, o número de itens na entrada
- O tempo de execução de um algoritmo sobre uma entrada particular é o número de passos realizados por ele
- Uma linha i de um algoritmo gasta uma quantidade constante de tempo $c_i > 0$

Análise de algoritmos

- O tempo gasto por um algoritmo cresce com o tamanho da entrada, isto é, o número de itens na entrada
- O tempo de execução de um algoritmo sobre uma entrada particular é o número de passos realizados por ele
- Uma linha i de um algoritmo gasta uma quantidade constante de tempo $c_i > 0$

Análise de algoritmos

- O tempo gasto por um algoritmo cresce com o tamanho da entrada, isto é, o número de itens na entrada
- O tempo de execução de um algoritmo sobre uma entrada particular é o número de passos realizados por ele
- Uma linha i de um algoritmo gasta uma quantidade constante de tempo $c_i > 0$

Análise de algoritmos

	Custo	Vezes
`#include <stdio.h>`	c_1	1
`#define MAX 100`	c_2	1
`int main(void)`	c_3	1
`{`	0	1
`    int n, i, C[MAX], x;`	c_4	1
`    scanf("%d", &n);`	c_5	1
`    for (i = 0; i < n; i++)`	c_6	$n+1$
`        scanf("%d", &C[i]);`	c_7	n
`    scanf("%d", &x);`	c_8	1
`    for (i = 0; i < n && C[i] != x; i++)`	c_9	t_i
`        ;`	0	$t_i - 1$
`    if (i < n)`	c_{10}	1
`        printf("%d está na posição %d de C\n", x, i);`	c_{11}	1
`    else`	c_{12}	1
`        printf("%d não pertence ao conjunto C\n", x);`	c_{13}	1
`    return 0;`	c_{14}	1
`}`	0	1

Análise de algoritmos

- O tempo de execução $T(n)$ do algoritmo é dado pela soma dos tempos para cada sentença executada
- Ou seja, devemos somar os produtos das colunas ***Custo*** e ***Vezes***

$$T(n) = c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + c_9 t_i + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} \, .$$

Análise de algoritmos

- O tempo de execução $T(n)$ do algoritmo é dado pela soma dos tempos para cada sentença executada
- Ou seja, devemos somar os produtos das colunas ***Custo*** e ***Vezes***

$$T(n) = c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + c_9 t_i + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} \,.$$

Análise de algoritmos

- O tempo de execução $T(n)$ do algoritmo é dado pela soma dos tempos para cada sentença executada
- Ou seja, devemos somar os produtos das colunas ***Custo*** e ***Vezes***

$$T(n) = c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + \\ c_9 t_i + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} \,.$$

Análise de algoritmos

- Tempo de execução $T(n)$ do algoritmo da busca:

$$T(n) = c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + \\ c_9 t_i + c_{10} + c_{11} + c_{12} + c_{13} + c_{14}\,.$$

- Melhor caso: ocorre se x encontra-se na primeira posição do vetor C; então, $t_i = 1$ e assim

$$\begin{aligned} T(n) &= c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + \\ &\quad c_9 + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} \\ &= (c_6 + c_7)n + c_1 + \ldots + c_6 + c_8 + \ldots + c_{14} \\ &= an + b\,. \end{aligned}$$

Análise de algoritmos

- Tempo de execução $T(n)$ do algoritmo da busca:

$$T(n) = c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + c_9 t_i + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} .$$

- Melhor caso: ocorre se x encontra-se na primeira posição do vetor C; então, $t_i = 1$ e assim

$$\begin{aligned} T(n) &= c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + c_9 + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} \\ &= (c_6 + c_7)n + c_1 + \ldots + c_6 + c_8 + \ldots + c_{14} \\ &= an + b . \end{aligned}$$

Análise de algoritmos

- Tempo de execução $T(n)$ do algoritmo da busca:

$$T(n) = c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + c_9 t_i + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} \,.$$

- Melhor caso: ocorre se x encontra-se na primeira posição do vetor C; então, $t_i = 1$ e assim

$$\begin{aligned} T(n) &= c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + c_9 + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} \\ &= (c_6 + c_7)n + c_1 + \ldots + c_6 + c_8 + \ldots + c_{14} \\ &= an + b \,. \end{aligned}$$

Análise de algoritmos

- Tempo de execução $T(n)$ do algoritmo da busca:

$$T(n) = c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + c_9 t_i + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} \,.$$

- Melhor caso: ocorre se x encontra-se na primeira posição do vetor C; então, $t_i = 1$ e assim

$$\begin{aligned} T(n) &= c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + \\ &\quad c_9 + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} \\ &= (c_6 + c_7)n + c_1 + \ldots + c_6 + c_8 + \ldots + c_{14} \\ &= an + b \,. \end{aligned}$$

Análise de algoritmos

- Tempo de execução $T(n)$ do algoritmo da busca:

$$T(n) = c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + c_9 t_i + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} \,.$$

- Pior caso: ocorre se x não se encontra no vetor C; então, $t_i = n + 1$ e assim

$$\begin{aligned} T(n) &= c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + c_9(n+1) + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} \\ &= (c_6 + c_7 + c_9)n + c_1 + \ldots + c_6 + c_8 + \ldots + c_{14} \\ &= an + b \,. \end{aligned}$$

Análise de algoritmos

- Tempo de execução $T(n)$ do algoritmo da busca:

$$T(n) = c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + c_9 t_i + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} \,.$$

- Pior caso: ocorre se x não se encontra no vetor C; então, $t_i = n + 1$ e assim

$$\begin{aligned} T(n) &= c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + c_9(n+1) + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} \\ &= (c_6 + c_7 + c_9)n + c_1 + \ldots + c_6 + c_8 + \ldots + c_{14} \\ &= an + b \,. \end{aligned}$$

Análise de algoritmos

- Tempo de execução $T(n)$ do algoritmo da busca:

$$T(n) = c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + c_9 t_i + c_{10} + c_{11} + c_{12} + c_{13} + c_{14}\,.$$

- Pior caso: ocorre se x não se encontra no vetor C; então, $t_i = n + 1$ e assim

$$\begin{aligned} T(n) &= c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + c_9(n+1) + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} \\ &= (c_6 + c_7 + c_9)n + c_1 + \ldots + c_6 + c_8 + \ldots + c_{14} \\ &= an + b\,. \end{aligned}$$

- Tempo de execução $T(n)$ do algoritmo da busca:

$$\begin{aligned} T(n) = c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + \\ c_9 t_i + c_{10} + c_{11} + c_{12} + c_{13} + c_{14}\,. \end{aligned}$$

- Pior caso: ocorre se x não se encontra no vetor C; então, $t_i = n + 1$ e assim

$$\begin{aligned} T(n) &= c_1 + c_2 + c_3 + c_4 + c_5 + c_6(n+1) + c_7 n + c_8 + \\ &\quad c_9(n+1) + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} \\ &= (c_6 + c_7 + c_9)n + c_1 + \ldots + c_6 + c_8 + \ldots + c_{14} \\ &= an + b\,. \end{aligned}$$

Análise de algoritmos

- Estamos interessados no tempo de execução de pior caso de um algoritmo

Ordem de crescimento de funções

- Estamos interessados na taxa de crescimento ou ordem de crescimento de uma função que descreve o tempo de execução de um algoritmo
- Consideramos apenas o *maior* termo da fórmula e ignoramos constantes
- Dizemos assim que o algoritmo da busca tem tempo de execução de pior caso $O(n)$
- Um algoritmo é mais eficiente que outro se seu tempo de execução de pior caso tem ordem de crescimento menor
- Eficiência assintótica de um algoritmo: como a função que descreve seu tempo de execução cresce com o tamanho da entrada *no limite*

Ordem de crescimento de funções

- Estamos interessados na taxa de crescimento ou ordem de crescimento de uma função que descreve o tempo de execução de um algoritmo
- Consideramos apenas o *maior* termo da fórmula e ignoramos constantes
- Dizemos assim que o algoritmo da busca tem tempo de execução de pior caso $O(n)$
- Um algoritmo é mais eficiente que outro se seu tempo de execução de pior caso tem ordem de crescimento menor
- Eficiência assintótica de um algoritmo: como a função que descreve seu tempo de execução cresce com o tamanho da entrada *no limite*

Ordem de crescimento de funções

- Estamos interessados na taxa de crescimento ou ordem de crescimento de uma função que descreve o tempo de execução de um algoritmo
- Consideramos apenas o *maior* termo da fórmula e ignoramos constantes
- Dizemos assim que o algoritmo da busca tem tempo de execução de pior caso $O(n)$
- Um algoritmo é mais eficiente que outro se seu tempo de execução de pior caso tem ordem de crescimento menor
- Eficiência assintótica de um algoritmo: como a função que descreve seu tempo de execução cresce com o tamanho da entrada *no limite*

Ordem de crescimento de funções

- Estamos interessados na taxa de crescimento ou ordem de crescimento de uma função que descreve o tempo de execução de um algoritmo
- Consideramos apenas o *maior* termo da fórmula e ignoramos constantes
- Dizemos assim que o algoritmo da busca tem tempo de execução de pior caso $O(n)$
- Um algoritmo é mais eficiente que outro se seu tempo de execução de pior caso tem ordem de crescimento menor
- Eficiência assintótica de um algoritmo: como a função que descreve seu tempo de execução cresce com o tamanho da entrada *no limite*

Ordem de crescimento de funções

- Estamos interessados na taxa de crescimento ou ordem de crescimento de uma função que descreve o tempo de execução de um algoritmo
- Consideramos apenas o *maior* termo da fórmula e ignoramos constantes
- Dizemos assim que o algoritmo da busca tem tempo de execução de pior caso $O(n)$
- Um algoritmo é mais eficiente que outro se seu tempo de execução de pior caso tem ordem de crescimento menor
- Eficiência assintótica de um algoritmo: como a função que descreve seu tempo de execução cresce com o tamanho da entrada *no limite*

Ordem de crescimento de funções

Definição

Para uma dada função $g(n)$, denotamos por $O(g(n))$ o conjunto de funções

$$O(g(n)) = \{f(n) \colon \text{existem constantes positivas } c \text{ e } n_0 \text{ tais que } 0 \leqslant f(n) \leqslant cg(n) \text{ para todo } n \geqslant n_0\} .$$

- $f(n)$ pertence ao conjunto $O(g(n))$ se existe uma constante positiva c tal que $f(n)$ "não seja maior que" $cg(n)$, para n suficientemente grande
- Usamos a notação O para fornecer um limitante assintótico superior sobre uma função, dentro de um fator constante
- Usamos $f(n) = O(g(n))$ para $f(n) \in O(g(n))$

Ordem de crescimento de funções

Definição

Para uma dada função $g(n)$, denotamos por $O(g(n))$ o conjunto de funções

$$O(g(n)) = \{f(n) \colon \text{existem constantes positivas } c \text{ e } n_0 \text{ tais que } 0 \leqslant f(n) \leqslant cg(n) \text{ para todo } n \geqslant n_0\} .$$

- $f(n)$ pertence ao conjunto $O(g(n))$ se existe uma constante positiva c tal que $f(n)$ "não seja maior que" $cg(n)$, para n suficientemente grande
- Usamos a notação O para fornecer um limitante assintótico superior sobre uma função, dentro de um fator constante
- Usamos $f(n) = O(g(n))$ para $f(n) \in O(g(n))$

Ordem de crescimento de funções

Definição

Para uma dada função $g(n)$, denotamos por $O(g(n))$ o conjunto de funções

$$O(g(n)) = \{f(n) : \text{existem constantes positivas } c \text{ e } n_0 \text{ tais que } 0 \leqslant f(n) \leqslant cg(n) \text{ para todo } n \geqslant n_0\} .$$

- $f(n)$ pertence ao conjunto $O(g(n))$ se existe uma constante positiva c tal que $f(n)$ "não seja maior que" $cg(n)$, para n suficientemente grande
- Usamos a notação O para fornecer um limitante assintótico superior sobre uma função, dentro de um fator constante
- Usamos $f(n) = O(g(n))$ para $f(n) \in O(g(n))$

Ordem de crescimento de funções

Definição

Para uma dada função $g(n)$, denotamos por $O(g(n))$ o conjunto de funções

$$O(g(n)) = \{f(n) \colon \text{existem constantes positivas } c \text{ e } n_0 \text{ tais que } 0 \leqslant f(n) \leqslant cg(n) \text{ para todo } n \geqslant n_0\} .$$

- $f(n)$ pertence ao conjunto $O(g(n))$ se existe uma constante positiva c tal que $f(n)$ "não seja maior que" $cg(n)$, para n suficientemente grande
- Usamos a notação O para fornecer um limitante assintótico superior sobre uma função, dentro de um fator constante
- Usamos $f(n) = O(g(n))$ para $f(n) \in O(g(n))$

Será que $4n + 1 = O(n)$?

Existem constantes positivas c e n_0 tais que

$$4n + 1 \leqslant cn$$

para todo $n \geqslant n_0$. Tome $c = 5$ e então

$$\begin{aligned} 4n + 1 &\leqslant 5n \\ 1 &\leqslant n\,, \end{aligned}$$

ou seja, para $n_0 = 1$, a desigualdade $4n + 1 \leqslant 5n$ é satisfeita para todo $n \geqslant n_0$ e assim, $4n + 1 = O(n)$

Ordem de crescimento de funções

Será que $4n + 1 = O(n)$?

Existem constantes positivas c e n_0 tais que

$$4n + 1 \leqslant cn$$

para todo $n \geqslant n_0$. Tome $c = 5$ e então

$$\begin{aligned} 4n + 1 &\leqslant 5n \\ 1 &\leqslant n \,, \end{aligned}$$

ou seja, para $n_0 = 1$, a desigualdade $4n + 1 \leqslant 5n$ é satisfeita para todo $n \geqslant n_0$ e assim, $4n + 1 = O(n)$

Ordem de crescimento de funções

Será que $4n + 1 = O(n)$?

Existem constantes positivas c e n_0 tais que

$$4n + 1 \leqslant cn$$

para todo $n \geqslant n_0$. Tome $c = 5$ e então

$$\begin{aligned} 4n + 1 &\leqslant 5n \\ 1 &\leqslant n \, , \end{aligned}$$

ou seja, para $n_0 = 1$, a desigualdade $4n + 1 \leqslant 5n$ é satisfeita para todo $n \geqslant n_0$ e assim, $4n + 1 = O(n)$

Ordem de crescimento de funções

Será que $4n + 1 = O(n)$?

Existem constantes positivas c e n_0 tais que

$$4n + 1 \leqslant cn$$

para todo $n \geqslant n_0$. Tome $c = 5$ e então

$$\begin{aligned} 4n + 1 &\leqslant 5n \\ 1 &\leqslant n \,, \end{aligned}$$

ou seja, para $n_0 = 1$, a desigualdade $4n + 1 \leqslant 5n$ é satisfeita para todo $n \geqslant n_0$ e assim, $4n + 1 = O(n)$

Ordem de crescimento de funções

Será que $4n + 1 = O(n)$?

Existem constantes positivas c e n_0 tais que

$$4n + 1 \leqslant cn$$

para todo $n \geqslant n_0$. Tome $c = 5$ e então

$$\begin{aligned} 4n + 1 &\leqslant 5n \\ 1 &\leqslant n\,, \end{aligned}$$

ou seja, para $n_0 = 1$, a desigualdade $4n + 1 \leqslant 5n$ é satisfeita para todo $n \geqslant n_0$ e assim, $4n + 1 = O(n)$

Análise da ordenação por trocas sucessivas

```
void trocas_sucessivas(int n, int v[MAX])
{
   int i, j, aux;

   for (i = n-1; i > 0; i--)
      for (j = 0; j < i; j++)
         if (v[j] > v[j+1]) {
            aux = v[j];
            v[j] = v[j+1];
            v[j+1] = aux;
         }
}
```

Análise da ordenação por trocas sucessivas

	Custo	***Vezes***
`void trocas_sucessivas(int n, int v[MAX])`	c_1	1
`{`	0	1
`   int i, j, aux;`	c_2	1
`   for (i = n-1; i > 0; i--)`	c_3	n
`      for (j = 0; j < i; j++)`	c_4	$\sum_{i=1}^{n-1}(i+1)$
`         if (v[j] > v[j+1]) {`	c_5	$\sum_{i=1}^{n-1} i$
`            aux = v[j];`	c_6	$\sum_{i=1}^{n-1} t_i$
`            v[j] = v[j+1];`	c_7	$\sum_{i=1}^{n-1} t_i$
`            v[j+1] = aux;`	c_8	$\sum_{i=1}^{n-1} t_i$
`         }`	0	$\sum_{i=1}^{n-1} i$
`}`	0	1

Análise da ordenação por trocas sucessivas

- Melhor caso: quando a sequência de entrada com n números inteiros é fornecida em ordem crescente ($t_i = 0$ para todo i)

$$T(n) = n + \sum_{i=1}^{n-1}(i+1) + \sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} t_i + \sum_{i=1}^{n-1} t_i + \sum_{i=1}^{n-1} t_i$$

$$= n + 2\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 + 3\sum_{i=1}^{n-1} t_i$$

$$= n + 2\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 + 3\sum_{i=1}^{n-1} 0$$

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

$$= n^2 + n - 1$$

Análise da ordenação por trocas sucessivas

- Melhor caso: quando a sequência de entrada com n números inteiros é fornecida em ordem crescente ($t_i = 0$ para todo i)

$$\begin{aligned}
T(n) &= n + \sum_{i=1}^{n-1}(i+1) + \sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} t_i + \sum_{i=1}^{n-1} t_i + \sum_{i=1}^{n-1} t_i \\
&= n + 2\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 + 3\sum_{i=1}^{n-1} t_i \\
&= n + 2\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 + 3\sum_{i=1}^{n-1} 0 \\
&= n + 2\,\frac{n(n-1)}{2} + n - 1 \\
&= n^2 + n - 1
\end{aligned}$$

Análise da ordenação por trocas sucessivas

- Melhor caso: quando a sequência de entrada com n números inteiros é fornecida em ordem crescente ($t_i = 0$ para todo i)

$$
\begin{aligned}
T(n) &= n + \sum_{i=1}^{n-1}(i+1) + \sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} t_i + \sum_{i=1}^{n-1} t_i + \sum_{i=1}^{n-1} t_i \\
&= n + 2\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 + 3\sum_{i=1}^{n-1} t_i \\
&= n + 2\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 + 3\sum_{i=1}^{n-1} 0 \\
&= n + 2\,\frac{n(n-1)}{2} + n - 1 \\
&= n^2 + n - 1
\end{aligned}
$$

Análise da ordenação por trocas sucessivas

- Melhor caso: quando a sequência de entrada com n números inteiros é fornecida em ordem crescente ($t_i = 0$ para todo i)

$$
\begin{aligned}
T(n) &= n + \sum_{i=1}^{n-1}(i+1) + \sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} t_i + \sum_{i=1}^{n-1} t_i + \sum_{i=1}^{n-1} t_i \\
&= n + 2\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 + 3\sum_{i=1}^{n-1} t_i \\
&= n + 2\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 + 3\sum_{i=1}^{n-1} 0 \\
&= n + 2\,\frac{n(n-1)}{2} + n - 1 \\
&= n^2 + n - 1
\end{aligned}
$$

Análise da ordenação por trocas sucessivas

- Melhor caso: quando a sequência de entrada com n números inteiros é fornecida em ordem crescente ($t_i = 0$ para todo i)

$$
\begin{aligned}
T(n) &= n + \sum_{i=1}^{n-1}(i+1) + \sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} t_i + \sum_{i=1}^{n-1} t_i + \sum_{i=1}^{n-1} t_i \\
&= n + 2\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 + 3\sum_{i=1}^{n-1} t_i \\
&= n + 2\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 + 3\sum_{i=1}^{n-1} 0 \\
&= n + 2\,\frac{n(n-1)}{2} + n - 1 \\
&= n^2 + n - 1
\end{aligned}
$$

Análise da ordenação por trocas sucessivas

- Melhor caso: $T(n) = n^2 + n - 1 = O(n^2)$
- Devemos encontrar duas constantes positivas c e n_0 tais que

$$n^2 + n - 1 \leqslant cn^2 , \quad \text{para todo } n \geqslant n_0$$

- Então, escolhendo $c = 2$ temos:

$$n^2 + n - 1 \leqslant 2n^2$$
$$n - 1 \leqslant n^2$$

ou seja,

$$n^2 - n + 1 \geqslant 0 .$$

- A inequação $n^2 - n + 1 \geqslant 0$ é sempre válida para todo $n \geqslant 1$

Análise da ordenação por trocas sucessivas

- Melhor caso: $T(n) = n^2 + n - 1 = O(n^2)$
- Devemos encontrar duas constantes positivas c e n_0 tais que

$$n^2 + n - 1 \leqslant cn^2 \,, \quad \text{para todo } n \geqslant n_0$$

- Então, escolhendo $c = 2$ temos:

$$n^2 + n - 1 \leqslant 2n^2$$
$$n - 1 \leqslant n^2$$

ou seja,

$$n^2 - n + 1 \geqslant 0 \,.$$

- A inequação $n^2 - n + 1 \geqslant 0$ é sempre válida para todo $n \geqslant 1$

Análise da ordenação por trocas sucessivas

- Melhor caso: $T(n) = n^2 + n - 1 = O(n^2)$
- Devemos encontrar duas constantes positivas c e n_0 tais que

$$n^2 + n - 1 \leqslant cn^2 , \quad \text{para todo } n \geqslant n_0$$

- Então, escolhendo $c = 2$ temos:

$$n^2 + n - 1 \leqslant 2n^2$$

$$n - 1 \leqslant n^2$$

ou seja,

$$n^2 - n + 1 \geqslant 0 .$$

- A inequação $n^2 - n + 1 \geqslant 0$ é sempre válida para todo $n \geqslant 1$

Análise da ordenação por trocas sucessivas

- Melhor caso: $T(n) = n^2 + n - 1 = O(n^2)$
- Devemos encontrar duas constantes positivas c e n_0 tais que

$$n^2 + n - 1 \leqslant cn^2 \,, \quad \text{para todo } n \geqslant n_0$$

- Então, escolhendo $c = 2$ temos:

$$\begin{aligned} n^2 + n - 1 &\leqslant 2n^2 \\ n - 1 &\leqslant n^2 \end{aligned}$$

ou seja,

$$n^2 - n + 1 \geqslant 0 \,.$$

- A inequação $n^2 - n + 1 \geqslant 0$ é sempre válida para todo $n \geqslant 1$

Análise da ordenação por trocas sucessivas

- Melhor caso: $T(n) = n^2 + n - 1 = O(n^2)$
- Devemos encontrar duas constantes positivas c e n_0 tais que

$$n^2 + n - 1 \leqslant cn^2 \,, \quad \text{para todo } n \geqslant n_0$$

- Então, escolhendo $c = 2$ temos:

$$\begin{aligned} n^2 + n - 1 &\leqslant 2n^2 \\ n - 1 &\leqslant n^2 \end{aligned}$$

ou seja,

$$n^2 - n + 1 \geqslant 0 \,.$$

- A inequação $n^2 - n + 1 \geqslant 0$ é sempre válida para todo $n \geqslant 1$

Análise da ordenação por trocas sucessivas

- Melhor caso: $T(n) = n^2 + n - 1 = O(n^2)$
- Devemos encontrar duas constantes positivas c e n_0 tais que

$$n^2 + n - 1 \leqslant cn^2 \,, \quad \text{para todo } n \geqslant n_0$$

- Então, escolhendo $c = 2$ temos:

$$\begin{aligned} n^2 + n - 1 &\leqslant 2n^2 \\ n - 1 &\leqslant n^2 \end{aligned}$$

ou seja,

$$n^2 - n + 1 \geqslant 0 \,.$$

- A inequação $n^2 - n + 1 \geqslant 0$ é sempre válida para todo $n \geqslant 1$

Análise da ordenação por trocas sucessivas

- Assim, escolhendo $c = 2$ e $n_0 = 1$, temos que

$$n^2 + n - 1 \leqslant cn^2$$

  para todo $n \geqslant n_0$, onde $c = 2$ e $n_0 = 1$
- Portanto, $T(n) = O(n^2)$

Análise da ordenação por trocas sucessivas

- Assim, escolhendo $c = 2$ e $n_0 = 1$, temos que

$$n^2 + n - 1 \leqslant cn^2$$

  para todo $n \geqslant n_0$, onde $c = 2$ e $n_0 = 1$
- Portanto, $T(n) = O(n^2)$

Análise da ordenação por trocas sucessivas

- Pior caso: quando a sequência de entrada com n números inteiros é fornecida em ordem decrescente ($t_i = i$ para todo i)

$$T(n) = n + \sum_{i=1}^{n-1}(i+1) + \sum_{i=1}^{n-1} i + 3\sum_{i=1}^{n-1} t_i$$

$$= n + \sum_{i=1}^{n-1}(i+1) + \sum_{i=1}^{n-1} i + 3\sum_{i=1}^{n-1} i$$

$$= n + 5\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1$$

$$= n + 5\,\frac{n(n-1)}{2} + n - 1$$

$$= \frac{5}{2}n^2 - \frac{5}{2}n + 2n - 1 = \frac{5}{2}n^2 - \frac{1}{2}n - 1$$

Análise da ordenação por trocas sucessivas

- Pior caso: quando a sequência de entrada com n números inteiros é fornecida em ordem decrescente ($t_i = i$ para todo i)

$$
\begin{aligned}
T(n) &= n + \sum_{i=1}^{n-1}(i+1) + \sum_{i=1}^{n-1} i + 3\sum_{i=1}^{n-1} t_i \\
&= n + \sum_{i=1}^{n-1}(i+1) + \sum_{i=1}^{n-1} i + 3\sum_{i=1}^{n-1} i \\
&= n + 5\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 \\
&= n + 5\,\frac{n(n-1)}{2} + n - 1 \\
&= \frac{5}{2}\,n^2 - \frac{5}{2}\,n + 2n - 1 = \frac{5}{2}\,n^2 - \frac{1}{2}\,n - 1
\end{aligned}
$$

Análise da ordenação por trocas sucessivas

- Pior caso: quando a sequência de entrada com n números inteiros é fornecida em ordem decrescente ($t_i = i$ para todo i)

$$
\begin{aligned}
T(n) &= n + \sum_{i=1}^{n-1}(i+1) + \sum_{i=1}^{n-1} i + 3\sum_{i=1}^{n-1} t_i \\
&= n + \sum_{i=1}^{n-1}(i+1) + \sum_{i=1}^{n-1} i + 3\sum_{i=1}^{n-1} i \\
&= n + 5\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 \\
&= n + 5\,\frac{n(n-1)}{2} + n - 1 \\
&= \frac{5}{2}n^2 - \frac{5}{2}n + 2n - 1 = \frac{5}{2}n^2 - \frac{1}{2}n - 1
\end{aligned}
$$

Análise da ordenação por trocas sucessivas

- Pior caso: quando a sequência de entrada com n números inteiros é fornecida em ordem decrescente ($t_i = i$ para todo i)

$$\begin{aligned}
T(n) &= n + \sum_{i=1}^{n-1}(i+1) + \sum_{i=1}^{n-1} i + 3\sum_{i=1}^{n-1} t_i \\
&= n + \sum_{i=1}^{n-1}(i+1) + \sum_{i=1}^{n-1} i + 3\sum_{i=1}^{n-1} i \\
&= n + 5\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 \\
&= n + 5\,\frac{n(n-1)}{2} + n - 1 \\
&= \frac{5}{2}n^2 - \frac{5}{2}n + 2n - 1 = \frac{5}{2}n^2 - \frac{1}{2}n - 1
\end{aligned}$$

Análise da ordenação por trocas sucessivas

- Pior caso: quando a sequência de entrada com n números inteiros é fornecida em ordem decrescente ($t_i = i$ para todo i)

$$
\begin{aligned}
T(n) &= n + \sum_{i=1}^{n-1}(i+1) + \sum_{i=1}^{n-1} i + 3\sum_{i=1}^{n-1} t_i \\
&= n + \sum_{i=1}^{n-1}(i+1) + \sum_{i=1}^{n-1} i + 3\sum_{i=1}^{n-1} i \\
&= n + 5\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 \\
&= n + 5\,\frac{n(n-1)}{2} + n - 1 \\
&= \frac{5}{2}\,n^2 - \frac{5}{2}\,n + 2n - 1 = \frac{5}{2}\,n^2 - \frac{1}{2}\,n - 1
\end{aligned}
$$

Análise da ordenação por trocas sucessivas

- Pior caso: $T(n) = (5/2)n^2 - (1/2)n - 1 = O(n^2)$
- Devemos encontrar duas constantes positivas c e n_0 tais que

$$\frac{5}{2}\,n^2 - \frac{1}{2}\,n - 1 \leqslant cn^2\,, \quad \text{para todo } n \geqslant n_0$$

- Então, escolhendo $c = 5/2$ temos:

$$\begin{aligned} \frac{5}{2}\,n^2 - \frac{1}{2}\,n - 1 &\leqslant \frac{5}{2}\,n^2 \\ -\frac{1}{2}\,n - 1 &\leqslant 0 \end{aligned}$$

ou seja,

$$\frac{1}{2}\,n + 1 \geqslant 0$$

Análise da ordenação por trocas sucessivas

- Pior caso: $T(n) = (5/2)n^2 - (1/2)n - 1 = O(n^2)$
- Devemos encontrar duas constantes positivas c e n_0 tais que

$$\frac{5}{2}\,n^2 - \frac{1}{2}\,n - 1 \leqslant cn^2\,, \quad \text{para todo } n \geqslant n_0$$

- Então, escolhendo $c = 5/2$ temos:

$$\frac{5}{2}\,n^2 - \frac{1}{2}\,n - 1 \leqslant \frac{5}{2}\,n^2$$
$$-\frac{1}{2}\,n - 1 \leqslant 0$$

ou seja,

$$\frac{1}{2}\,n + 1 \geqslant 0$$

Análise da ordenação por trocas sucessivas

- Pior caso: $T(n) = (5/2)n^2 - (1/2)n - 1 = O(n^2)$
- Devemos encontrar duas constantes positivas c e n_0 tais que

$$\frac{5}{2}\,n^2 - \frac{1}{2}\,n - 1 \leqslant cn^2\,, \quad \text{para todo } n \geqslant n_0$$

- Então, escolhendo $c = 5/2$ temos:

$$\frac{5}{2}\,n^2 - \frac{1}{2}\,n - 1 \leqslant \frac{5}{2}\,n^2$$

$$-\frac{1}{2}\,n - 1 \leqslant 0$$

ou seja,

$$\frac{1}{2}\,n + 1 \geqslant 0$$

Análise da ordenação por trocas sucessivas

- Pior caso: $T(n) = (5/2)n^2 - (1/2)n - 1 = O(n^2)$
- Devemos encontrar duas constantes positivas c e n_0 tais que

$$\frac{5}{2}\,n^2 - \frac{1}{2}\,n - 1 \leqslant cn^2\,, \quad \text{para todo } n \geqslant n_0$$

- Então, escolhendo $c = 5/2$ temos:

$$\begin{aligned} \frac{5}{2}\,n^2 - \frac{1}{2}\,n - 1 &\leqslant \frac{5}{2}\,n^2 \\ -\frac{1}{2}\,n - 1 &\leqslant 0 \end{aligned}$$

ou seja,

$$\frac{1}{2}\,n + 1 \geqslant 0$$

Análise da ordenação por trocas sucessivas

- Pior caso: $T(n) = (5/2)n^2 - (1/2)n - 1 = O(n^2)$
- Devemos encontrar duas constantes positivas c e n_0 tais que

$$\frac{5}{2}\,n^2 - \frac{1}{2}\,n - 1 \leqslant cn^2\,, \quad \text{para todo } n \geqslant n_0$$

- Então, escolhendo $c = 5/2$ temos:

$$\begin{aligned} \frac{5}{2}\,n^2 - \frac{1}{2}\,n - 1 &\leqslant \frac{5}{2}\,n^2 \\ -\frac{1}{2}\,n - 1 &\leqslant 0 \end{aligned}$$

ou seja,

$$\frac{1}{2}\,n + 1 \geqslant 0$$

Análise da ordenação por trocas sucessivas

- A inequação $(1/2)n + 1 \geqslant 0$ é sempre válida para todo $n \geqslant 1$
- Assim, escolhendo $c = 5/2$ e $n_0 = 1$, temos que

$$\frac{5}{2}n^2 - \frac{1}{2}n - 1 \leqslant cn^2$$

para todo $n \geqslant n_0$, onde $c = 5/2$ e $n_0 = 1$
- Portanto, $T(n) = O(n^2)$

Análise da ordenação por trocas sucessivas

- A inequação $(1/2)n + 1 \geqslant 0$ é sempre válida para todo $n \geqslant 1$
- Assim, escolhendo $c = 5/2$ e $n_0 = 1$, temos que

$$\frac{5}{2}n^2 - \frac{1}{2}n - 1 \leqslant cn^2$$

para todo $n \geqslant n_0$, onde $c = 5/2$ e $n_0 = 1$

- Portanto, $T(n) = O(n^2)$

Análise da ordenação por trocas sucessivas

- A inequação $(1/2)n + 1 \geqslant 0$ é sempre válida para todo $n \geqslant 1$
- Assim, escolhendo $c = 5/2$ e $n_0 = 1$, temos que

$$\frac{5}{2}n^2 - \frac{1}{2}n - 1 \leqslant cn^2$$

  para todo $n \geqslant n_0$, onde $c = 5/2$ e $n_0 = 1$
- Portanto, $T(n) = O(n^2)$

Moral da história

- Algoritmos diferentes para resolver um mesmo problema em geral diferem dramaticamente em eficiência
- Essas diferenças podem ser muito mais significativas que a diferença de tempos de execução em um supercomputador e em um computador pessoal

Moral da história

- Algoritmos diferentes para resolver um mesmo problema em geral diferem dramaticamente em eficiência
- Essas diferenças podem ser muito mais significativas que a diferença de tempos de execução em um supercomputador e em um computador pessoal

Moral da história

Problema da ordenação

- Algoritmo A
  - Tempo de execução de pior caso $O(n^2)$
  - Supercomputador: 100 milhões de operações por segundo
  - Programador: hacker, codificação com tempo de execução $2n^2$
- Algoritmo B
  - Tempo de execução de pior caso $O(n \log_2 n)$
  - Computador pessoal: 1 milhão de operações por segundo
  - Programador: mediano, codificação com tempo de execução $50n \log_2 n$
- Ordenar 1 milhão de números ($n = 10^6$)

Moral da história

Problema da ordenação

- Algoritmo A
  - Tempo de execução de pior caso $O(n^2)$
  - Supercomputador: 100 milhões de operações por segundo
  - Programador: hacker, codificação com tempo de execução $2n^2$
- Algoritmo B
  - Tempo de execução de pior caso $O(n \log_2 n)$
  - Computador pessoal: 1 milhão de operações por segundo
  - Programador: mediano, codificação com tempo de execução $50n \log_2 n$
- Ordenar 1 milhão de números ($n = 10^6$)

Moral da história

Problema da ordenação

- Algoritmo A
  - Tempo de execução de pior caso $O(n^2)$
  - Supercomputador: 100 milhões de operações por segundo
  - Programador: hacker, codificação com tempo de execução $2n^2$
- Algoritmo B
  - Tempo de execução de pior caso $O(n \log_2 n)$
  - Computador pessoal: 1 milhão de operações por segundo
  - Programador: mediano, codificação com tempo de execução $50n \log_2 n$
- Ordenar 1 milhão de números ($n = 10^6$)

Moral da história

Problema da ordenação

- Algoritmo A
  - Tempo de execução de pior caso $O(n^2)$
  - Supercomputador: 100 milhões de operações por segundo
  - Programador: hacker, codificação com tempo de execução $2n^2$
- Algoritmo B
  - Tempo de execução de pior caso $O(n \log_2 n)$
  - Computador pessoal: 1 milhão de operações por segundo
  - Programador: mediano, codificação com tempo de execução $50n \log_2 n$
- Ordenar 1 milhão de números ($n = 10^6$)

Moral da história

Problema da ordenação

- Algoritmo A

$$\frac{2 \cdot (10^6)^2 \text{ operações}}{10^8 \text{ operações/segundo}} = 20.000 \text{ segundos} \approx 5{,}56 \text{ horas}$$

- Algoritmo B

$$\frac{50 \cdot 10^6 \log_2 10^6 \text{ operações}}{10^6 \text{ operações/segundo}} \approx 1.000 \text{ segundos} \approx 16{,}67 \text{ minutos}$$

Moral da história

Problema da ordenação

- Algoritmo A

$$\frac{2 \cdot (10^6)^2 \text{ operações}}{10^8 \text{ operações/segundo}} = 20.000 \text{ segundos} \approx 5{,}56 \text{ horas}$$

- Algoritmo B

$$\frac{50 \cdot 10^6 \log_2 10^6 \text{ operações}}{10^6 \text{ operações/segundo}} \approx 1.000 \text{ segundos} \approx 16{,}67 \text{ minutos}$$

Exercícios

1. Qual o menor valor de n tal que um programa com tempo de execução $100n^2$ é mais rápido que um programa cujo tempo de execução é 2^n, supondo que os programas foram implementados no mesmo computador?
2. Suponha que estamos comparando as implementações dos métodos de ordenação por trocas sucessivas e por intercalação em um mesmo computador. Para entradas de tamanho n, o método das trocas sucessivas gasta $8n^2$ passos enquanto que o método da intercalação gasta $64n \log_2 n$ passos. Para quais valores de n o método das trocas sucessivas é melhor que o método da intercalação?
3. Expresse a função $n^3/1000 - 100n^2 - 100n + 3$ na notação O

Exercícios

4. Para cada função $f(n)$ e tempo t na tabela abaixo determine o maior tamanho n de um problema que pode ser resolvido em tempo t, considerando que o programa soluciona o problema em $f(n)$ microssegundos.

	1 segundo	1 minuto	1 hora	1 dia	1 mês	1 ano	1 século
$\log_2 n$							
$\sqrt{n}$							
n							
$n \log_2 n$							
n^2							
n^3							
2^n							
$n!$							

Exercícios

5. É verdade que $2^{n+1} = O(2^n)$? E é verdade que $2^{2n} = O(2^n)$?
6. Suponha que você tenha algoritmos com os cinco tempos de execução listados abaixo. Quão mais lento cada um dos algoritmos fica quando você (*i*) duplica o tamanho da entrada, ou (*ii*) incrementa em uma unidade o tamanho da entrada?
   (a) n^2
   (b) n^3
   (c) $100n^2$
   (d) $n \log_2 n$
   (e) 2^n

Exercícios

7. Suponha que você tenha algoritmos com os cinco tempos de execução listados abaixo. Quão mais lento cada um dos algoritmos fica quando você (*i*) duplica o tamanho da entrada, ou (*ii*) incrementa em uma unidade o tamanho da entrada?
   (a) n^2
   (b) n^3
   (c) $100n^2$
   (d) $n \log_2 n$
   (e) 2^n

8. Rearranje a seguinte lista de funções em ordem crescente de taxa de crescimento. Isto é, se a função $g(n)$ sucede imediatamente a função $f(n)$ na sua lista, então é verdade que $f(n) = O(g(n))$.

$$
\begin{aligned}
f_1(n) &= n^{2.5} \\
f_2(n) &= \sqrt{2n} \\
f_3(n) &= n + 10 \\
f_4(n) &= 10^n \\
f_5(n) &= 100^n \\
f_6(n) &= n^2 \log_2 n
\end{aligned}
$$

Exercícios

9. Considere o problema de computar o valor de um polinômio em um ponto. Dados n coeficientes $a_0, a_1, \ldots, a_{n-1}$ e um número real x, queremos computar $\sum_{i=0}^{n-1} a_i x^i$.
   (a) Escreva um programa simples com tempo de execução de pior caso $O(n^2)$ para solucionar este problema.
   (b) Escreva um programa com tempo de execução de pior caso $O(n)$ para solucionar este problema usando o método chamado de regra de Horner para reescrever o polinômio:

$$\sum_{i=1}^{n-1} a_i x^i = (\cdots(a_{n-1}x + a_{n-2})x + \cdots + a_1)x + a_0 \ .$$

Exercícios

10. Seja $A[0..n-1]$ um vetor de n números inteiros distintos dois a dois. Se $i < j$ e $A[i] > A[j]$ então o par (i, j) é chamado uma **inversão** de A.
    (a) Liste as cinco inversões do vetor $A = \langle 2, 3, 8, 6, 1 \rangle$.
    (b) Qual vetor com elementos no conjunto $\{1, 2, \ldots, n\}$ tem a maior quantidade de inversões? Quantas são?
    (c) Escreva um programa que determine o número de inversões em qualquer permutação de n elementos em tempo de execução de pior caso $O(n \log_2 n)$.