Algoritmos e Estrutura de Dados II
(ESTCMP011)
2do PERIODO 2018

UEA
UNIVERSIDADE DO ESTADO DO AMAZONAS

Algoritmos de ordenação: insertion_sort

Prof. Luis Cuevas Rodríguez, PhD
E-mail: lcuevasrodriguez@gmail.com / lrodriguez@uea.edu.br
Celular: 9298154648

EST
Escola Superior de Tecnologia da UEA

NUCOMP
NÚCLEO DE COMPUTAÇÃO

Conteúdo

- Problema de ordenação
- Especificado o algoritmo de ordenação por:
  - inserção (insertion_sort)
- Analisar seu tempo de execução.

Problema de ordenação

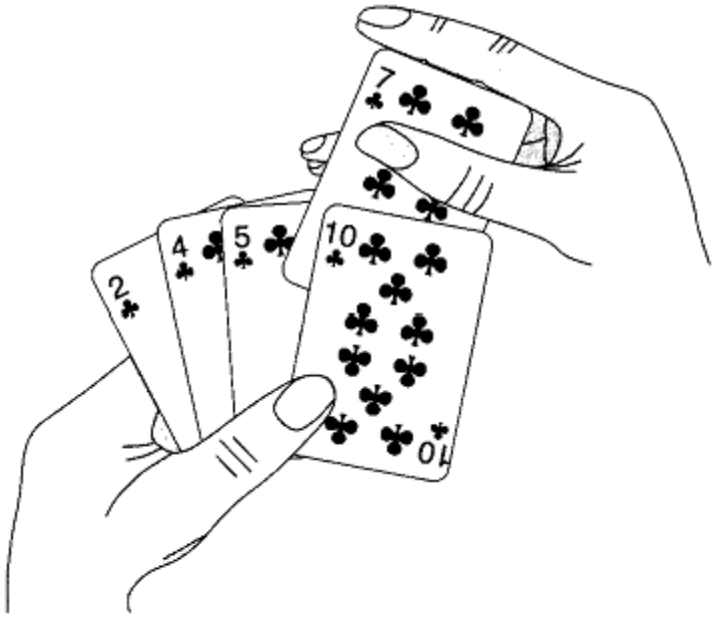
- Ordenar uma sequência de números em ordem crescente.:
  - Entrada: Uma sequência de *n* números $(a_1, a_2, ..., a_n)$ → chaves
  - Saída: Uma permutação (reordenação) $(a'_1, a'_2, . . . , a'_n)$ da sequência de entrada, tal que $a'_1 \leq a'_2 \leq . . . \leq a'_n$

Entrada	Saída
(31,41,59,26,41,58)	(26,31,41,41,58,59)

- Instância do problema de ordenação →Uma sequência de entrada do problema que satisfaz as restrições impostas no enunciado do problema necessária para se calcular uma solução para o problema.

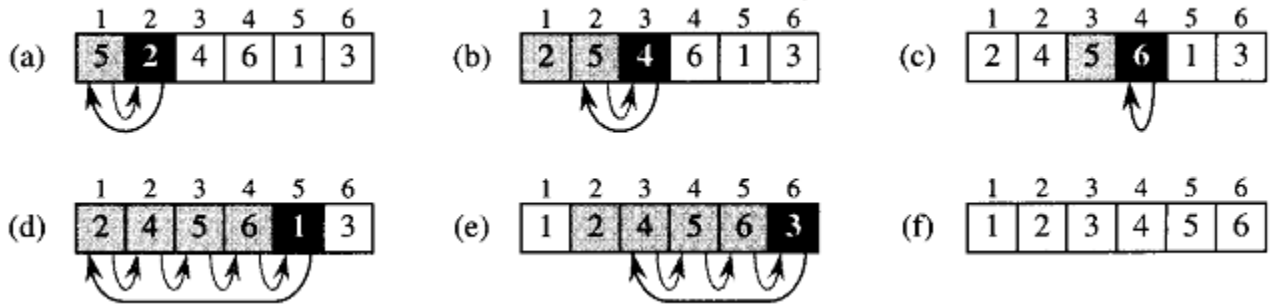
Problema de ordenação

- A ordenação é uma operação fundamental em ciência da computação
- um grande número de bons algoritmos de ordenação tem sido desenvolvido.
- O melhor algoritmo para uma determinada aplicação depende de:
  - número de itens a serem ordenados,
  - extensão em que os itens já estão ordenados de algum modo
  - possíveis restrições sobre os valores de itens
  - da espécie de dispositivo de armazenamento a ser usado: memória principal, discos ou fitas



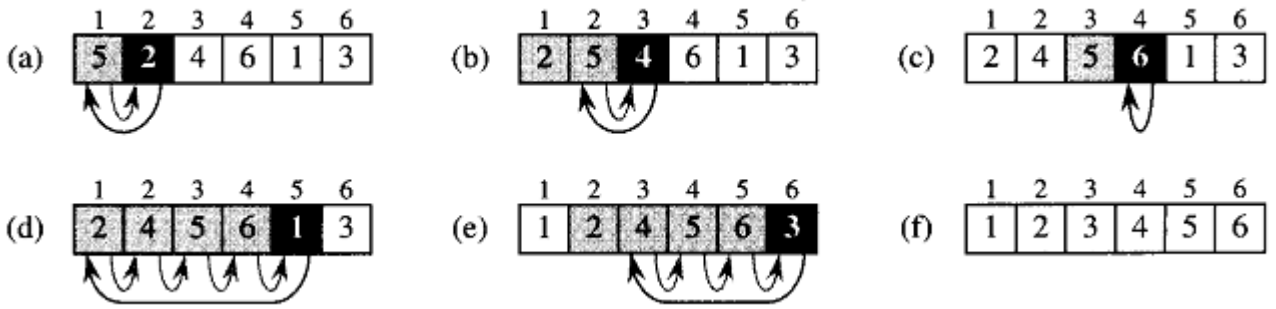
Ordenação por **inserção**

- Algoritmo eficiente para ordenar um número pequeno de elementos.
- Funciona da maneira como muitas pessoas ordenam as cartas em um jogo de bridge ou pôquer.
  - Inicia com a mão esquerda vazia e as cartas viradas com a face para baixo na mesa.
  - Remover uma carta de cada vez da mesa, inserindo-a na posição correta na mão esquerda.
  - Para encontrar a posição correta de uma carta, compara-se a cada uma das cartas que já estão na mão, da direita para a esquerda



Ordenação por inserção

```
1 for j ← 2 to comprimento[A]
2       do chave ← A[j]
3          ▷ Inserir A[j] na seqüência ordenada A[1..j – 1].
4          i ← j – 1
5          while i > 0 e A[i] > chave
6             do A[i + 1] ← A[i]
7                i ← i – 1
8          A[i + 1] ← chave
```

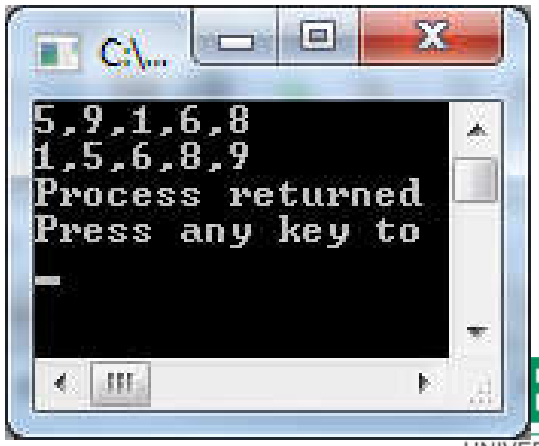


Exercícios

- Ilustre a operação de INSERTION-SORT nos vetores
  - A = (31, 41, 59, 26, 41, 58).
  - B = (5, 1, 9, 6, 8).

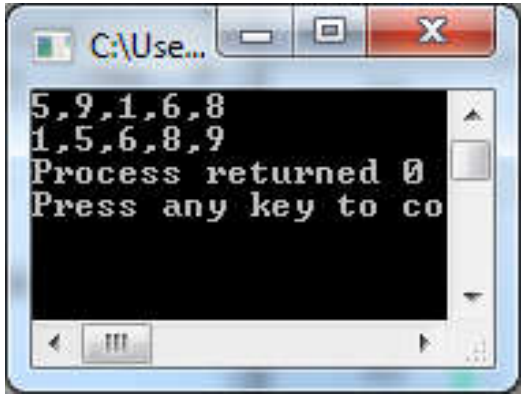
Implementação 1

```
#define TAM 5
void imprime_vetor(int l[TAM]);

int main()
{
     int lista[TAM]={5,9,1,6,8};
     int temp, i, j;
     imprime_vetor(lista);
     for (i=0; i<TAM; i++){
          temp = lista[i];
          j = i - 1;
          while ((lista[j] > temp) && (j >= 0) ){
               lista[j+1] = lista[j];
               lista[j] = temp;
               j--;
          }
     }
     cout << endl;
     imprime_vetor(lista);
     return 0;
}
```



Implementação 1

```
#define TAM 5
void imprime_vetor(int l[TAM]);

int main()
{
    int lista[TAM]={5,9,1,6,8};
    int temp, i, j;
    imprime_vetor(lista);
    ordena_insercao(lista);
    cout << endl;
    imprime_vetor(lista);
    return 0;
}
```




```
void ordena_insercao(int *l){
    int i,j, temp;
    for (i=0; i<TAM; i++){
        temp = *(l+i);
        j = i - 1;
        while ((*(l+j) > temp) && (j >= 0) ){
            *(l+j+1) = *(l+j);
            *(l+j) = temp;
            j--;
        }
    }
}
```

Exercícios

- Reescreva o procedimento INSERTION-SORT para ordenar em ordem não crescente, em vez da ordem não decrescente.

Análise do algoritmo

- O tempo de duração de um algoritmo cresce com o tamanho da entrada; assim, é tradicional descrever o tempo de execução de um programa como uma função do tamanho de sua entrada.
  - **tamanho da entrada**: número de itens na entrada
  - **tempo de execução**: o número de operações (cada execução da *i-ésima* linha leva um tempo c_i, onde c_i é uma constante)

Análise do algoritmo

```
int main()
{
    int lista[TAM]={5,9,1,6,8};
    int temp, i, j;
    imprime_vetor(lista);
    for (i=0; i<TAM; i++){
        temp = lista[i];
        j = i - 1;
        while ((lista[j] > temp) && (j >= 0) ){
            lista[j+1] = lista[j];
            lista[j] = temp;
            j--;
        }
    }
    cout << endl;
    imprime_vetor(lista);
    return 0;
}
```

c_1	1	
c_2	1	
c_3	1	
c_4	n+1	
c_5	n	
c_6	n	
c_7	$\sum j$	
c_8	$\sum j-1$	
c_9	$\sum j-1$	
c_{10}	$\sum j-1$	
c_{11}	1	
c_{12}	1	
c_{13}	1	

Análise de algoritmos

- Usando padrão de complexidade (crescimento) . Constantes e elementos de ordem inferior são descartados
  - Melhor caso: $an+b = \Theta(n)$
  - Pior caso: $an^2+n+b = \Theta(n^2)$

Bibliografía

- CORMEN, T.H., LEISERSON, C.E., RIVEST, R.L., STEIN, C. Algoritmos: Teoria e Prática. Tradução da 3a. edição. Rio de Janeiro: Elsevier, 2012.
- SZWARCFITER, J, L., MARKENZON, L. Estruturas de Dados e seus Algoritmos. 2a edição. Rio de Janeiro: LTC, 2010.
- ZIVIANI, N. Projeto de Algoritmos com Implementação em Pascal e C. 3a edição. São Paulo: Cengage Learning, 2010.
- STROUSTRUP, B. Programming: Principles and Practice Using C++. 2nd Edition. Addison-Wesley Professional. 2014.
- B. Stroustrup. The C++ Programming Language, 4th Edition, 2013.
- Feofiloff, P. Algoritmos em Linguagem C. Elsevier. 2009.
- AHO, A. V. et al. Data Structure and Algorithms. Readings, Addison-Wesley.
- WIRTH, N. Algoritmos e Estruturas de Dados. Rio de Janeiro: Ed. Prentice Hall do Brasil.
- KNUTH, D. E. The Art of Computer Programming. Vol. 1, Addison-Wesley, Reading, Mass.