# Algoritmos de Ordenação - Insertion Sort

SCC0201 - Introdução à Ciência de Computação II

Clausius G. Reis  Leandro A. Amaral  Tiago S. Nazaré
Vanessa Q. Marinho

Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo

21 de junho de 2015

# Agenda

# Agenda

# Introdução

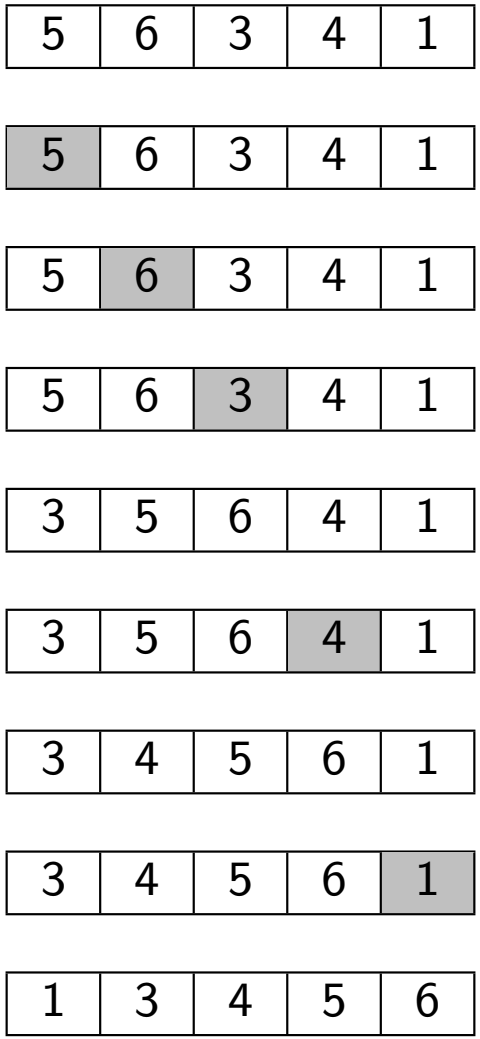
É um dos algoritmos mais rápidos para ordenar **vetores pequenos**. Porém, por sua natureza quadrática, é inviável para vetores maiores.

Método bastante utilizado por jogadores de cartas para ordená-las em suas mãos.

**Ideia Básica:**
A cada passo, a partir de $i = 2$, o $i$-ésimo elemento do vetor é selecionado e inserido no lugar apropriado (considerando-se apenas os elementos anteriores a ele).

# Agenda

# Exemplo - Ordem Crescente



# Agenda

# Insertion Sort - Código

```
void insertion_sort(int *a, int size){
        int i, j, value;

        for (i=1; i<size; ++i){
                value = a[i];
                j=i-1;

                while(j >= 0 && a[j] > value){
                        a[j+1] = a[j];
                        j = j-1;
                }

                a[j+1] = value;
        }
}
```

# Agenda

# Análise de Complexidade - Melhor Caso ($\mathcal{O}(n)$)

O melhor caso acontece quando o vetor já está ordenado.

Nesse caso, o tempo de execução é linear. Durante cada iteração, o elemento do vetor de entrada é comparado apenas com o elemento mais a direita da parte já ordenada do array. Portanto: Número de comparações: $(1 + 1 + ... + 1) =$ n-1 $= \mathcal{O}(n)$

# Análise de Complexidade - Pior Caso e Caso Médio

**Pior Caso** - $\mathcal{O}(n^2)$
O pior caso acontece quando a entrada é um array ordenado em ordem inversa.
Nesse caso, cada iteração do algoritmo irá comparar o elemento atual com todos os anteriores.
Número de comparações: $(1 + 2 + ... + \text{n-1}) = \text{n(n-1)}/2 = \mathcal{O}(n^2)$

**Caso Médio** - $\mathcal{O}(n^2)$
O Caso Médio também é quadrático. **Em casa:** Olhar a demonstração do cálculo para o caso médio disponível no link abaixo da Universidade de Auckland.

---

http://130.216.33.163/compsci220s1c/lectures/2014S1C/Part1/220-08.pdf

# Vantagens

Para vetores já ordenados, o algoritmo descobre a um custo $\mathcal{O}(n)$ que cada item já está no seu lugar.

Logo, o método da inserção é o método a ser utilizado quando o vetor está "quase" ordenado.

É também um bom método quando se deseja adicionar uns poucos itens a um vetor já ordenado: neste caso, o custo é linear.

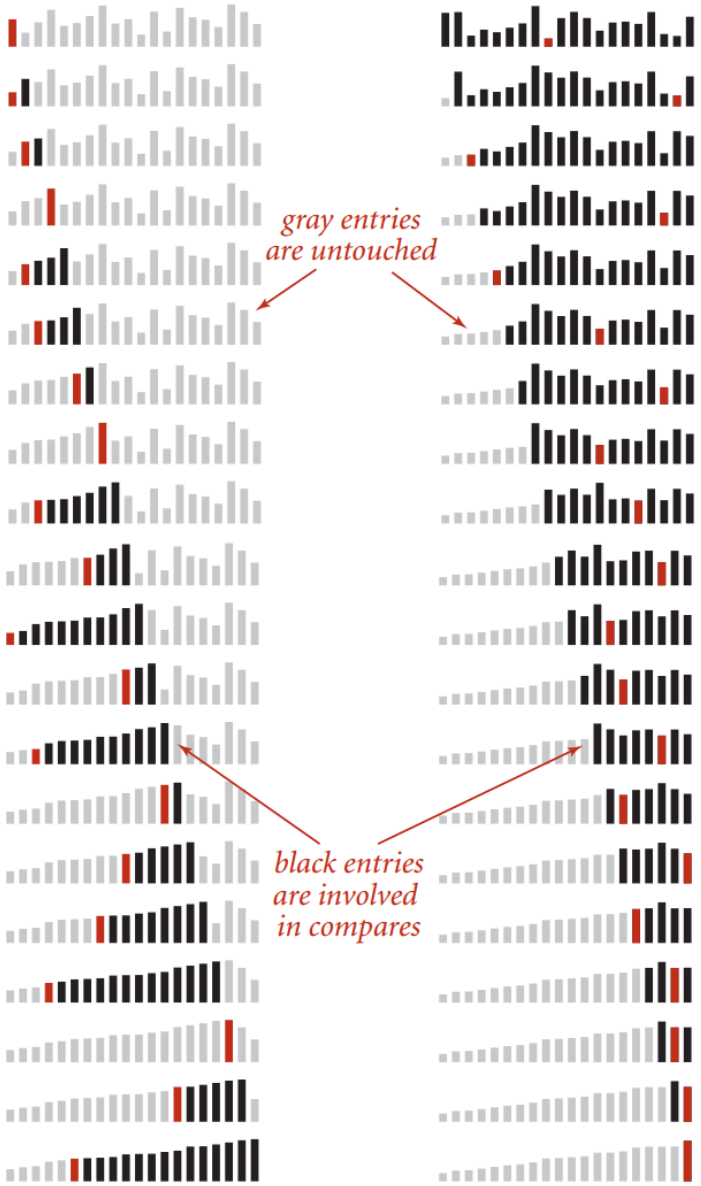
# Comparação

A figura a seguir apresenta a comparação entre os algoritmos *Selection Sort* e *Insertion Sort*.

Um conjunto de barras são ordenadas de acordo com suas alturas.

É possível ver que o algoritmo *Insertion Sort* não acessa elementos a direita do elemento selecionado em cada iteração (apenas olha para trás) e o *Selection Sort* não acessa elementos a esquerda do elemento de cada iteração (apenas olha para frente).

Outro fato visualmente claro é que o *Selection Sort* realiza um número maior de comparações.


gray entries
are untouched
black entries
are involved
in compares
insertion sort
selection sort

# Exercício

Implementar (em C) o algoritmo de Insertion Sort de maneira recursiva. **Dica:** Considere o fim do vetor como variável.

# Resposta

```
void insertion_sort_rec(int *a, int end, int size){

        int j, value;
        value = a[end];
        j=end-1;
        while(j >= 0 && a[j] > value){
                a[j+1] = a[j];
                --j;
        }
        a[j+1] = value;

        if(end < (size -1))
                insertion_sort_rec(a,++end,size);
}
void main() {
    insertion_sort_rec(a, 1, size);
}
```