Algoritmos de Ordenação

Indução Fraca (primeira alternativa):

Caso base: para n = 1 temos um conjunto ordenado de um único elemento.

Passo indutivo: assumimos que sabemos ordenar n-1 elementos e agora queremos ordenar n elementos (restando o n-enésimo elemento x para ser ordenado.

Conjunto não ordenado

Por hipótese de indução, sabemos ordenar n-1 elementos

O que falta fazer é colocar o elemento x na posição correta:

Esta indução dá origem ao algoritmo *Insertion Sort* (Inserção Direta ou Ordenação por Inserção).

Implementação Iterativa

```
static void insertionSort(int[] A) {
    int i, j, v;
    int fim = A.length;

    for (i = 1; i < fim; i++) {
        v = A[i];
        j = i;
        while ((j>0) && (A[j-1]>v)){
            A[j] = A[j - 1];
            j = j - 1;
        }
        A[j] = v;
    }
}
```

O primeiro laço (*for*) ocorre com i valendo de 1 até n-1 (ou seja, n-1 vezes). O laço interno/encadeado (*while*) ocorre enquanto j maior que zero (j inicia com i) e enquanto o valor da posição j-1 do arranjo for maior do que o valor do elemento atual.

Melhor caso (arranjo ordenado de forma crescente)

O teste do while ocorrerá apenas uma única vez a cada iteração de i:

melhor_caso(n) = n – 1

Pior caso (arranjo ordenado de forma decrescente)

A comparação A[j - 1] > v sempre retornará verdadeiro.

O total de operações será dado pela soma das iterações para cada valor de i:

para i=1 haverá 1 operação

para i=2 haverá 2 operações

...

para i = n-1 haverá n-1 operações

A soma de 1 + 2 + 3 ... + n-1 (soma de P.A.) resulta em (n-1+1)*(n-1)/2 = n*(n-1)/2

pior_caso = n*(n-1)/2 = $(n^2-n)/2$

Assim, esta implementação ϵ $O(n^2)$

Implementação Recursiva

```
static void insertionSortRec(int[] A, int
n) {
    if (n > 1){
        insertionSortRec(A, n-1);
        int temp;
        int i = n-1;
        while (i>0 && A[i-1] > A[i]) {
            temp = A[i];
            A[i] = A[i-1];
            A[i-1] = temp;
            i--;
        }
    }
}
```

Melhor caso (arranjo ordenado de forma crescente)

T(n) = 0 para n = 1

T(n) = 1*T(n-1) + 1 para n > 1

Se resolvermos a equação teremos:

T(n) = n - 1

Pior caso (arranjo ordenado de forma decrescente)

T(n) = 0 para n = 1

T(n) = 1*T(n-1) + n - 1 para n > 1

Se resolvermos a equação teremos:

$T(n) = (n^2 - n)/2$

Assim, esta implementação ϵ $O(n^2)$

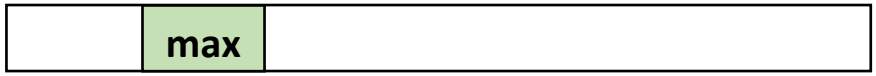
Indução Fraca (segunda alternativa):

Caso base: para n = 1 temos um conjunto ordenado de um único elemento.

Passo indutivo: encontramos o maior elemento do conjunto e assumimos que sabemos ordenar n-1 (excluindo-se o maior).

Conjunto não ordenado

Encontramos o maior



Trocamos o max de lugar com o último número do subconjunto que estamos analisando. Por hipótese de indução, sabemos ordenar n-1 elementos

Não falta fazer nada pois o max já está no lugar certo (todos os elementos à esquerda dele são menores do que ele).

Esta indução dá origem ao algoritmo *Selection Sort* (Ordenação por Seleção).

Implementação Iterativa

```
static void selectionSort(int[] A) {
    int fim, j, max, temp;
    int n = A.length;
    for (fim = n-1; fim>0; fim--) {
        max = fim;
        for (j = 0; j < fim; j++) {
            if (A[j] > A[max]) {
                max = j;
            }
        }
        if (fim != max){
            temp = A[fim];
            A[fim] = A[max];
            A[max] = temp;
        }
    }
}
```

O primeiro laço (*for*) ocorre com fim valendo de n-1 até n-1 (ou seja, n-1 vezes). O laço interno/encadeado (*for*) ocorre enquanto j menor do que fim (j inicia com 0).

Não há melhor ou pior caso em relação à comparação de valores de elementos do arranjo.
O número de operações será dado pela soma das operações de comparação dentro do *if* presente no laço *for* interno:
para fim=n-1 haverá n-1 operações
para fim=n-2 haverá n-2 operações
...
para fim = 2 haverá 1 operação
A soma de n-1 + n-2 + ... + 2 + 1 (soma de P.A.) resulta em (n-1+1)*(n-1)/2 = n*(n-1)/2

operações(n) = n*(n-1)/2 = $(n^2-n)/2$

Assim, esta implementação $\epsilon\ \Theta(n^2)$

Implementação Recursiva

```
static void selectionSortRec(int[] A, int
n) {
    if (n > 1){
        int max = 0;
        for (int i = 1; i < n; i++) {
            if (A[i] > A[max]) {
                max = i;
            }
        }
        if (n-1 != max){
            int temp = A[n-1];
            A[n-1] = A[max];
            A[max] = temp;
        }
        selectionSortRec(A, n-1);
    }
}
```

Não há melhor ou pior caso em relação à comparação de valores de elementos do arranjo.

$T(n) = 0$ para $n = 1$

$T(n) = 1*T(n-1) + n-1$ para $n > 1$

Se resolvermos a equação teremos:

$T(n) = (n^2 - n)/2$

Assim, esta implementação $\epsilon\ \Theta(n^2)$

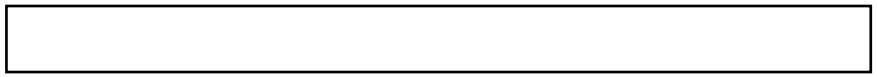
Indução Fraca (terceira alternativa):

Caso base: para n = 1 temos um conjunto ordenado de um único elemento.

Passo indutivo: encontramos o maior elemento do conjunto, mas durante a busca por esse elemento, a cada comparação entre elementos vizinhos, colocamos o menor dos dois a esquerda.

Assumimos que sabemos ordenar n-1 (excluindo-se o maior).

Conjunto não ordenado:



Encontramos o maior (com sucessivas trocas entre vizinhos):

Por hipótese de indução, sabemos ordenar n-1 elementos

Não falta fazer nada pois o max já está no lugar certo (todos os elementos à esquerda dele são menores do que ele).

Esta indução dá origem ao algoritmo *Bubble Sort* (Método Bolha).

Implementação Iterativa

```
static void bubbleSort(int[] A) {
    int temp;
    int n = A.length;
    for (int i = n - 1; i > 0; i--) {
        for (int j=1; j <= i; j++) {
            if (A[j - 1] > A[j]) {
                temp = A[j - 1];
                A[j - 1] = A[j];
                A[j] = temp;
            }
        }
    }
}
```

O primeiro laço (*for*) ocorre com i valendo de n-1 até n-1 (ou seja, n-1 vezes). O laço interno/encadeado (*for*) ocorre enquanto j menor ou igual a i (j inicia com 1).

Não há melhor ou pior caso em relação à comparação de valores de elementos do arranjo.
O número de operações será dado pela soma das operações de comparação dentro do *if* presente no laço *for* interno:
para i=n-1 haverá n-1 operações
para i=n-2 haverá n-2 operações
...
para i = 2 haverá 1 operação
A soma de n-1 + n-2 + ... + 2 + 1 (soma de P.A.) resulta em (n-1+1)*(n-1)/2 = n*(n-1)/2

operações(n) = n*(n-1)/2 = $(n^2-n)/2$

Assim, esta implementação ϵ $\Theta(n^2)$

Implementação Recursiva

```
static void bubbleSortRec(int[] A, int n)
{
    if (n > 1){
        int temp;
        for (int i = 1; i < n; i++) {
            if (A[i-1] > A[i] ) {
                temp = A[i-1];
                A[i-1] = A[i];
                A[i] = temp;
            }
        }
        bubbleSortRec(A, n-1);
    }
}
```

Não há melhor ou pior caso em relação à comparação de valores de elementos do arranjo.

$T(n) = 0$ para $n = 1$

$T(n) = 1*T(n-1) + n-1$ para $n > 1$

Se resolvermos a equação teremos:

$T(n) = (n^2 - n)/2$

Assim, esta implementação $\epsilon\ \Theta(n^2)$