UDF Centro Universitário

Métodos de Ordenação

Prof. Antonio PEREIRA

Cruzeiro do Sul Educacional

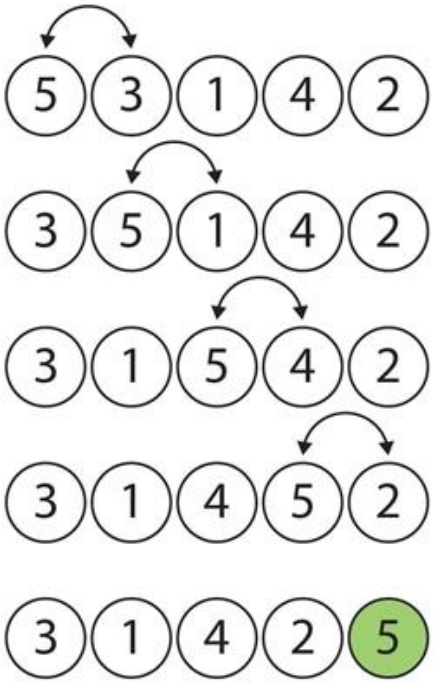
Métodos de Ordenação

- **Entrada: uma seqüência de n items < A_1, A_2, ... A_n>**
- **Saída: uma permutação (reordenação) <A_1',A_2', ... , A_n'> da seqüência de entrada, tal que $A_1' \leq A_2' \leq ... \leq A_n'$**

- **Ordenação Interna (ou em memória):** nesse método todos os elementos a serem ordenados estão na memória, por exemplo, um vetor de elementos. Podemos usar esses métodos quando os elementos cabem na memória.
- **Ordenação Externa:** nesse outro método, os elementos não cabem todos na memória, sendo necessário o uso de técnicas de swap para poder ordenar os elementos, por exemplo, um arquivo de 1GB.

Bubblesort (método da bolha)

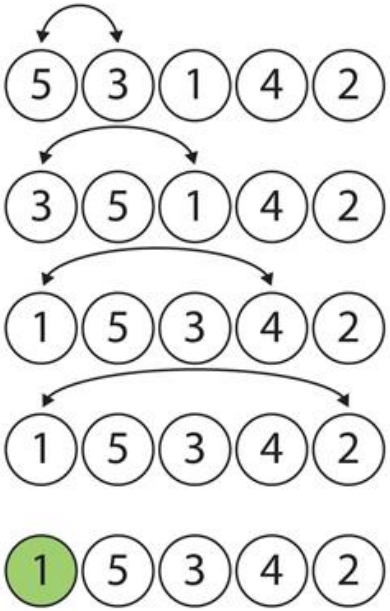
- O princípio do *Bubblesort* é a troca de valores entre posições consecutivas, fazendo com que os valores mais altos (ou mais baixos) “borbulhem” para o final do arranjo (daí o nome *Bubblesort*)




```
public static void bubbleSort(int vetor[]){
    int aux;
    int tam = vetor.length;
    for(int i=0; i < tam - 1; i++ ){
        for(int j=0; j < tam - 1 - i; j++){
            if(vetor[j] > vetor[j+1]){
                aux = vetor[j];
                vetor[j] = vetor[j+1];
                vetor[j+1] = aux;
            }
        }
    }
}
```

Selection Sort

- O método de ordenação por seleção (Seleção Direta) pode ser comparado à ordenação de cartas.
  - Imagine todas as cartas espalhadas na mesa e o jogador seleciona a menor de todas e a coloca em sua mão, assim até o final das cartas. Desse modo, no final do processo, as cartas estarão ordenadas.




```
public static void selectionSort(int vetor[]){
    int aux;
    int tam = vetor.length;
    for(int i=0; i < tam - 1; i++){
        for(int j = i + 1; j < tam; j++){
            if(vetor[j] < vetor [i]){
                aux = vetor[j];
                vetor[j] = vetor[i];
                vetor[i] = aux;
            }
        }
    }
}
```

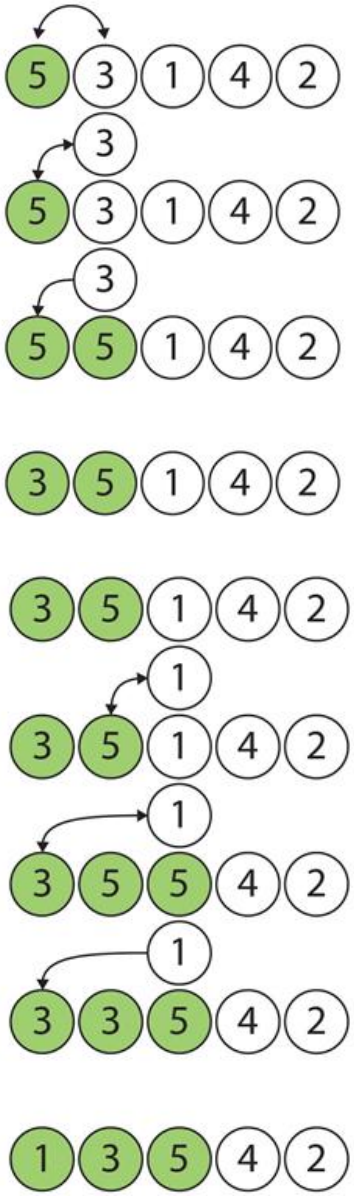
Insertion Sort

- O método de ordenação por inserção é bastante simples e eficiente para um pequeno número de dados a se ordenar.
- Consiste em ir inserindo os dados um a um já na posição correta onde deve ficar, sendo que no final da inserção do último elemento, todos os dados já estarão ordenados.
- A ideia é semelhante à ordenação de cartas de baralho na mão do jogador.
  - O jogador coloca uma carta por vez na mão, e a cada carta que insere, já a põe na ordem correta, sendo que ao final, todas as cartas estarão na ordem.

Insertion Sort

- A principal característica desse método consiste em ordenarmos nosso arranjo utilizando um subarranjo ordenado localizado em seu início, e a cada novo passo, acrescentamos a este subarranjo mais um elemento, até que atingimos o último elemento do arranjo fazendo assim com que ele se torne ordenado.

Insertion Sort



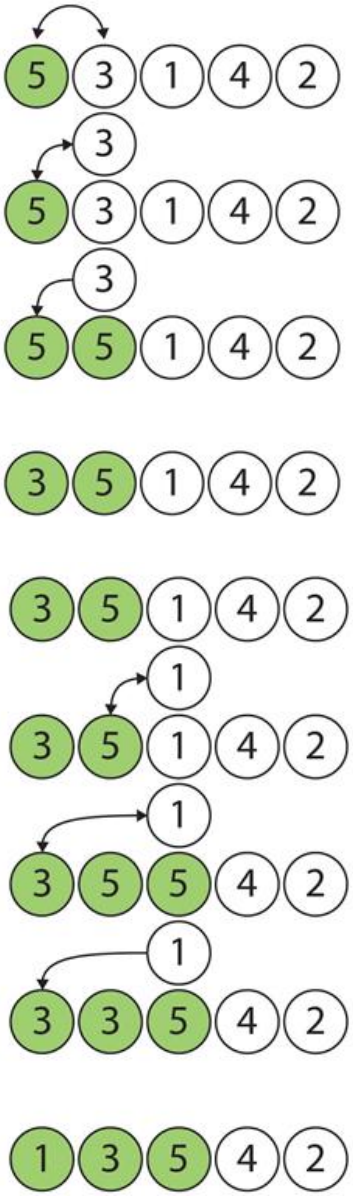
Inicialmente consideramos o primeiro elemento do arranjo como se ele estivesse ordenado, ele será considerado o nosso subarranjo ordenado inicial.

Após copiá-lo, devemos percorrer o subarranjo a partir do último elemento para o primeiro. Assim poderemos encontrar a posição correta da nossa variável auxiliar dentro do subarranjo.

Vamos comparar a cópia com os valores no subvetor ordenado. A cada vez que o elemento for menor que o elemento a ser comparado, deve-se copiar o elemento para a direita.

A cada nova interação, o elemento é inserido na posição cor reta.

Insertion Sort




```
public static void insertSort(int vetor[]){
        int aux;
        int tam = vetor.length;
        int j;
        for(int i=1; i < tam; i++){
                aux = vetor[i];
                j = i - 1;
                while(j >= 0 && aux < vetor[j]){
                        vetor[j+1] = vetor[j];
                        j--;
                }
                vetor[j+1] = aux;
        }
}
```

Quicksort

- Seu algoritmo consiste em dividir o problema em problemas menores, utilizando um algoritmo recursivo para a ordenação dos elementos.
- Um dos aspectos interessantes da ordenação por segmentação é que ela ordena as coisas quase da mesma maneira como fazem as pessoas.
- Primeiro, ela cria grandes “pilhas”, e depois, as ordena em pilhas cada vez menores, terminando finalmente com um array inteiramente ordenado.

Quicksort

- O algoritmo de ordenação por segmentação começa estimando um valor de intervalo médio para o array.
- Se o array for composto de números de 1 a 10, o ponto médio poderia ser 5 ou 6 (elemento pivô).
  - O valor exato do ponto médio não é fundamental. O algoritmo vai trabalhar com um ponto médio de qualquer valor. Entretanto, quanto mais próximo o ponto médio estimado estiver do ponto médio real do array, mais rápida será a ordenação.
- A rotina calcula um ponto médio considerando o primeiro e o último elemento da parte do array que está sendo ordenada.
- Depois que a rotina seleciona um ponto médio, ela coloca todos os elementos menores que o ponto médio na parte inferior do array e todos os elementos maiores na parte superior. Em seguida, o algoritmo se repete na parte inferior do array e na parte superior, sucessivamente, até que o sub-array tenha apenas um elemento.

Quicksort

```
public static void quickSort(int vetor[]){
    quickSort(vetor, 0, vetor.length - 1);
}
public static void quickSort(int vetor[], int i, int s){
    int e=i, d=s;
    int item = vetor[((e+d)/2)];
    while(e <= d){
        while(vetor[e] < item) e++;
        while(vetor[d] > item) d--;
        if(e <= d){
            int aux;    // Variável auxiliar para as trocas
            aux = vetor[e];
            vetor[e] = vetor[d];
            vetor[d] = aux;
            d--;
            e++;
        }
    }
    if(d - i > 0) quickSort(vetor, i, d);
    if(s - e > 0) quickSort(vetor, e, s);
}
```