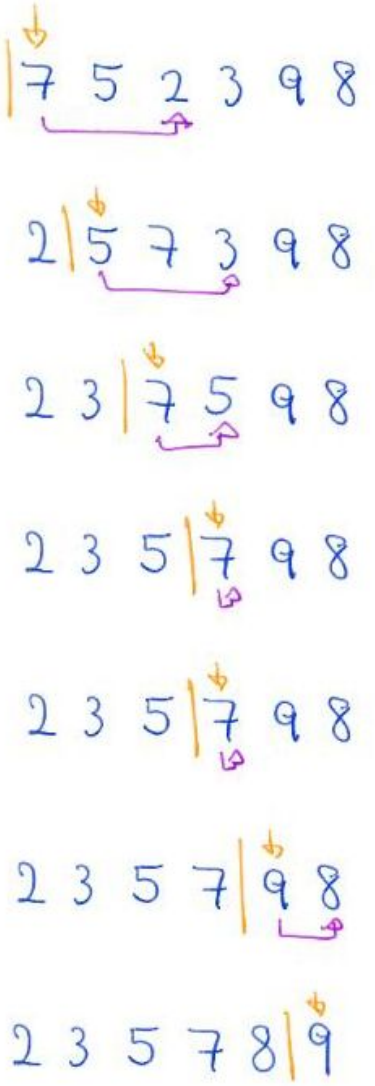
AED1 - Aula 08
Ordenação por seleção (selectionsort) e por transposição (bubblesort)

Selectionsort (ordenação por seleção)

Ideia e exemplo:

- varre o vetor do início ao fim e em cada iteração
    - busca o mínimo do subvetor restante e o coloca na posição corrente.
- Como exemplo, considere o vetor 7 5 2 3 9 8



Código:

```
void selectionSort(int v[], int n)
{
   int i, j, ind_min, aux;
   for (i = 0; i < n - 1; i++)
   {
       ind_min = i;
```

```
        for (j = i + 1; j < n; j++)
            if (v[j] < v[ind_min])
                ind_min = j;
        aux = v[i];
        v[i] = v[ind_min];
        v[ind_min] = aux;
    }
}
```

Invariante e corretude:

- os invariantes do laço externo, que valem no início de cada iteração são:
    - o vetor é uma permutação do original,
    - v[0 .. i - 1] está ordenado,
    - v[i - 1] <= v[k], para i <= k < n.
- invariantes do laço interno:
    - v[ind_min] <= v[i .. j - 1]
- demonstrar que esses invariantes estão corretos:
    - verificando que eles valem antes da primeira iteração
    - e que valem de uma iteração pra outra.
- verificar que, no final do laço, os invariantes implicam a corretude do algoritmo.

Eficiência de tempo:

- em qualquer caso o número de operações realizadas pelo algoritmo é da ordem de n(n-1)/2 ~= n^2/2 = O(n^2).
    - isso porque em cada iteração do laço externo precisamos percorrer todo o subvetor restante para encontrar o próximo mínimo.
- Bônus:
    - observe que, se soubéssemos encontrar o mínimo do subvetor restante sem precisar percorrê-lo totalmente, trocar tal mínimo com o elemento da posição corrente leva tempo constante.
        - essa ideia geraria um algoritmo mais eficiente?
        - esse algoritmo funciona? Isto é, ele está correto?
    - note também que podemos propor uma variante deste algoritmo que
        - varre o vetor do fim para o começo e
        - seleciona o máximo do subvetor restante a cada iteração.

Estabilidade:

- ordenação não é estável.
    - isso porque as trocas do mínimo com a posição corrente podem levar à inversão da ordem relativa entre elementos iguais.
    - considere o vetor [2 2 1 3 4 5 6 7].

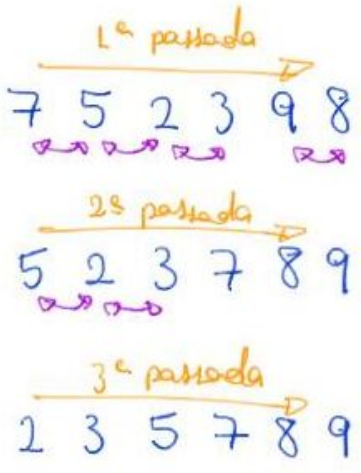
Eficiência de espaço:

- ordenação é in place, pois só usa estruturas auxiliares (e portanto memória) de tamanho constante em relação à entrada.

Bubblesort (ordenação por transposição)

Ideia e exemplo:

- Varre o vetor diversas vezes, invertendo pares adjacentes fora de ordem.
- Como exemplo, considere o vetor 7 5 2 3 9 8



Códigos:

```c
void bubbleSort1(int v[], int n)
{
   int i, aux, mudou = 1;
   while (mudou == 1)
   {
       mudou = 0;
       for (i = 1; i < n; i++)
           if (v[i - 1] > v[i])
           {
               aux = v[i - 1];
               v[i - 1] = v[i];
               v[i] = aux;
               mudou = 1;
           }
   }
}
```

- Por que esse algoritmo para?
    - porque um vetor em que todo par adjacente respeita v[i - 1] <= v[i] por definição está ordenado.

```c
void bubbleSort2(int v[], int n)
{
   int j, i, aux;
   for (j = 0; j < n; j++)
```

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

- Quando esse algoritmo para, o vetor está ordenado?
    - sim, pois existem no máximo (n escolhe 2) = n(n - 1)/2 pares invertidos e depois de n passagens desinvertendo pares todos estão em ordem.
    - Mais precisamente, note que uma passagem leva o maior elemento para a última posição do vetor, a segunda passagem leva o segundo maior para a penúltima posição, e assim por diante. Portanto, depois de n passagens todos os elementos estão ordenados.
    - Disso decorre a melhoria do próximo algoritmo, já que o laço interno não precisa passar pelas últimas posições que já contém os maiores elementos em ordem crescente.

```
void bubbleSort3(int v[], int n)
{
   int j, i, aux;
   for (j = 0; j < n; j++)
        for (i = 1; i < n - j; i++)
            if (v[i - 1] > v[i])
            {
                aux = v[i - 1];
                v[i - 1] = v[i];
                v[i] = aux;
            }
}
```

- Será que podemos fazer ainda melhor?
    - observe que todos os elementos depois da posição da última inversão já estão ordenados (caso contrário teria ocorrido alguma inversão entre eles)
    - além disso, eles correspondem aos maiores elementos do vetor, como observado anteriormente.
    - portanto, a passagem subsequente do laço interno não precisa passar pelas posições após a última inversão da iteração anterior.
    - essa melhoria é implementada no próximo algoritmo.

```
void bubbleSort4(int v[], int n)
{
   int j, i, aux, ut, l;
   l = n;
   for (j = 0; j < n; j++)
   {
```

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

Invariante e corretude:

- os principais invariantes que valem no início de cada iteração são:
    - o vetor é uma permutação do original,
    - v[l .. n - 1] está ordenado,
    - v[l] >= v[k], para 0 <= k < l.
- demonstrar que esses invariantes estão corretos:
    - verificando que eles valem antes da primeira iteração
    - e que valem de uma iteração pra outra.
- verificar que, no final do laço, os invariantes implicam a corretude do algoritmo.

Eficiência de tempo:

- no melhor caso é O(n).
    - Ex.: vetor está ordenado ou tem apenas adjacentes fora de ordem.
- no pior caso é O(n^2).
    - Ex.: vetor está ordenado exceto pelo menor estar na última posição.

Estabilidade:

- ordenação é estável.
    - por que?
- O que acontece se trocarmos "v[i-1] > v[i]" por "v[i-1] >= v[i]"? Continua ordenando? Continua estável? Sempre termina?

Eficiência de espaço:

- ordenação é in place, pois só usa estruturas auxiliares (e portanto memória) de tamanho constante em relação à entrada.

Animação:

- Visualization and Comparison of Sorting Algorithms - www.youtube.com/watch?v=ZZuD6iUe3Pc