Estruturas de Dados - Ordenação

O problema de ordenação

- Entrada do problema: vetor arbitrário S
- Objetivo: ordenar os elementos de S
  - nesta disciplina, vamos considerar a ordenação em ordem crescente ($\forall i, \forall j, i < j \rightarrow S[i] \leq S[j]$)
  - obs: a ordenação decrescente tem um funcionamento parecido e, portanto, uma implementação parecida
- Como fazer?

O problema de ordenação

- Do inglês, *sort*: ordenar
- Há diferentes maneiras de se resolver esse problema
  - Bubble Sort
  - Selection Sort
  - Insertion Sort
  - Quick Sort
  - Merge Sort
  - Counting Sort
  - Radix Sort
  - ⋮
- Alguns métodos são mais eficientes do que outros
- Vamos apresentar alguns deles durante a disciplina

Bubble Sort

- Ideia para ordenar:
  - empurrar o maior dos n elementos para a n-ésima posição do vetor
  - após isso, empurrar o maior dos $n-1$ primeiros elementos para a $(n-1)$-ésima posição do vetor
  - após isso, empurrar o maior dos $n-2$ primeiros elementos para a $(n-2)$-ésima posição do vetor
  - $\vdots$
- Corretude: essa ideia funciona?
  - essa ideia resolve o problema de ordenação?

Bubble Sort

- A primeira iteração deve colocar o maior dos n elementos na n-ésima posição
  - Em um vetor ordenado, o maior elemento deve estar na última posição
- Logo, o n-ésimo elemento está na sua posição correta no vetor ordenado
- A partir disso, nosso problema passa a ser ordenar os $n - 1$ primeiros elementos
- Repetimos esse processo até que tenhamos apenas um elemento
  - vetor com um elemento: caso trivial, vetor já está ordenado

Bubble Sort: algoritmo

Algoritmo: BubbleSort(S, p, r)

Entrada: vetor S, índices p e r

Resultado: vetor S ordenado de p a $r-1$

```
1 enquanto r > p faça
2 |   m = p
3 |   para i = p + 1 até r − 1 faça
4 |   |   se S[i] > S[m] então
5 |   |   |   m = i
6 |   S[m] ↔ S[r − 1]
7 |   r = r − 1
```

Bubble Sort: complexidade

- Complexidade do Bubble Sort:
  - operação básica: comparação entre elementos do vetor
  - considere n o tamanho do vetor ($n = r - p$)
  - quantidade de comparações:

$$T(n) = (n-1) + (n-2) + ... + 3 + 2 + 1$$

$$T(n) = \sum_{i=n-1}^{1} i = \frac{(n-1+1)(n-1)}{2} = \frac{n^2 - n}{2}$$

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

Bubble Sort: algoritmo (outra versão)

Algoritmo: BubbleSort(S, p, r)

Entrada: vetor S, índices p e r

Resultado: vetor S ordenado de p a $r-1$

```
1 repita
2 |   trocado = false
3 |   para i = p até r − 2 faça
4 |   |   se S[i] > S[i + 1] então
5 |   |   |   trocado = true
6 |   |   |   S[i] ↔ S[i + 1]
7 enquanto trocado
```

Selection Sort

- Ideia para ordenar:
  - puxar o menor dos n elementos para a primeira posição do vetor
  - após isso, puxar o menor dos $n-1$ últimos elementos para a segunda posição do vetor
  - após isso, puxar o menor dos $n-2$ últimos elementos para a terceira posição do vetor
  - $\vdots$
- Corretude: essa ideia funciona?

Selection Sort

- A primeira iteração deve colocar o menor dos n elementos na primeira posição
  - Em um vetor ordenado, o menor elemento deve estar na primeira posição
- Logo, o primeiro elemento está na sua posição correta no vetor ordenado
- A partir disso, nosso problema passa a ser ordenar os $n - 1$ primeiros elementos
- Repetimos esse processo até que tenhamos apenas um elemento
  - vetor com um elemento: caso trivial, vetor já está ordenado

Selection Sort: algoritmo

Algoritmo: SelectionSort(S, p, r)

Entrada: vetor S, índices p e r

Resultado: vetor S ordenado de p a $r-1$

```
1 enquanto p < r faça
2 |   m = p
3 |   para i = p + 1 até r − 1 faça
4 |   |   se S[i] < S[m] então
5 |   |   |   m = i
6 |   S[m] ↔ S[p]
7 |   p = p + 1
```

Selection Sort: complexidade

- Complexidade do Selection Sort:
  - operação básica: comparação entre elementos do vetor
  - considere n o tamanho do vetor ($n = r - p$)
  - quantidade de comparações:

$$T(n) = (n-1) + (n-2) + ... + 3 + 2 + 1$$

$$T(n) = \sum_{i=1}^{n-1} i = \frac{(n-1+1)(n-1)}{2} = \frac{n^2 - n}{2}$$

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

Bubble Sort e Selection Sort

- Vantagens:
  - simples implementação
  - não necessitam de vetor auxiliar - utilizamos apenas uma variável auxiliar (economiza memória)
- Desvantagens:
  - lentos (complexidades $O(n^2)$)
    - obs: a segunda versão do Bubble Sort apresentada tem, no melhor caso, uma complexidade $O(n)$ se o vetor já estiver ordenado
  - sempre fazem a mesma quantidade de comparações, mesmo se o vetor estiver ordenado
  - não são estáveis
    - algoritmo estável: não muda a ordem relativa de elementos com valores iguais