Estruturas de Dados II

Quick Sort

Prof. Bruno Azevedo

Instituto Federal de São Paulo

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus Catanduva

Quick Sort

- Vamos conhecer mais um algoritmo para ordenação de elementos: o **Quick Sort**.
- Este provavelmente é algoritmo de ordenação mais utilizado por desenvolvedores.
- Entretanto, seu tempo de execução no pior caso é de $\Theta(n^2)$.
- Isso ocorre porque na prática ele é excepcionalmente eficiente. Seu tempo de execução **esperado** é $\Theta(n \log n)$ quando todos os números são distintos e as constantes ocultas na notação assintótica são pequenas.

Quick Sort

- Além disso, ele também ordena *in place*, ao contrário do Merge Sort.
- O pior caso ocorre apenas para instâncias bem específicas.
- O Quick Sort, assim como o Merge Sort, aplica a estratégia de **Divisão e Conquista**.

Quick Sort

- Mas hoje faremos algo diferente...
- Como sabem, vocês terão que aprender tudo sozinhos no mercado de trabalho. Vocês não terão o professor Bruno para ajudá-los.

Quick Sort

- Portanto, esse último algoritmo de ordenação que aprenderemos em ED2 não será detalhado por mim, mas por vocês.
- Estudem e compreendam o funcionamento do algoritmo Quick Sort (não é complicado) e me enviem a explicação por e-mail. Grupos de até duas pessoas.
- Enviem até o fim da aula!
- Não se preocupem com a análise de complexidade, já que esta cai na mesma situação do Merge Sort, envolvendo conhecimentos externos ao curso de vocês.

Quick Sort

- Vou ajudar vocês. Segue o pseudo-código do algoritmo Quick Sort.

```
QuickSort(A, p, r)
  if p < r
    q = Partition(A, p, r)
    QuickSort(A, p, q - 1)
    QuickSort(A, q + 1, r)
```

```
Partition(A, p, r)
  x = A[r]
  i = p - 1
  for j = p to r - 1
      if A[j] <= x
          i = i + 1
          Troca A[i] com A[j]
  Troca A[i + 1] com A[r]
  return i + 1
```

Exercícios

- Imagino que façam a tarefa anterior rapidamente, portanto, seguem mais alguns exercícios para vocês praticarem o conhecimento obtido.

1. Aplique manualmente o Quick Sort no vetor V = {2, 8, 7, 1, 3, 5, 6, 4} e exiba todos os passos intermediários.
2. Aplique manualmente o Quick Sort no vetor V = {13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11} e exiba todos os passos intermediários.
3. Qual é o tempo de execução do Quick Sort em um vetor de comprimento *n* que já está ordenado em ordem crescente? E se o vetor conter todos os elementos iguais?
4. O que determina a ocorrência do pior caso na execução do Quick Sort? Quais eventos durante sua execução podem causar um desempenho "degradado" do algoritmo?
5. Existe uma versão randomizada do Quick Sort. Como ela funciona e qual vantagem ela oferece comparada a versão clássica?