

Algoritmos de ordenação - bubble, quick, insertion

Cleber Soares¹, Sthefany Oliveira ¹, Victor Souza¹

¹Faculdade de Computação e engenharia elétrica
Universidade Federal do Sul e Sudeste do Pará (UNIFESSPA)
68505-080 – Marabá – PA – Brasil

{cleber.soares, sthefany.oliveira, victoor.soouza}@unifesspa.edu.br

Abstract. *Ordering algorithms are computational methods for organizing and / or ordering a list of numbers or words according to their individualities, they can take care of each case that best suits their needs and then improve certain problems that are related to the retrieval of data in lists, that is, facilitate the search of information. This article presents its implementation, execution and comparative among three sorting algorithms that are: Insertion Sort, Bubble Sort, and Quick Sort. For the purpose of analyzing your results and checking the positives and negatives of each.*

Resumo. *Os algoritmos de ordenação são métodos computacionais para organizar e ou ordenar lista de números ou palavras, de acordo com suas individualidades, podem atender cada caso que se adeque melhor a sua necessidade para então melhorar certos problemas que estejam relacionados a recuperação de dados em listas, ou seja, facilitam a busca das informações. Este artigo apresenta a sua implementação, execução e comparativo entre três algoritmos de ordenação que são: Insertion Sort, Bubble Sort, e Quick Sort. Com a finalidade de analisar seus resultados e verificar os pontos positivos e negativos de cada um.*

Algoritmos

Bubble Sort

O Bubble Sort é o tipo mais antigo e mais simples usado para ordenações. Ele funciona comparando cada item da lista com o item do lado dele, e efetua a troca se o valor na posição que está sendo analisada for maior que o da posição após a dele. O algoritmo repete este processo até passar por todas as posições da lista. Isto faz com que os valores maiores “flutuem” para o final da lista, enquanto os valores menores “afundem” para o início da lista [da Silva Nascimento et al. 2016].

Tal algoritmo usa a estratégia de troca e seu pior caso $O(n^2)$ e melhor caso $O(n)$.

Algorithm 1 Bubble Sort

```
1: function BUBBLESORT( $A, n$ )
2:   for  $i = 0$  to  $i < (n - 1)$  do
3:     for  $j = 0$  to  $j < (n - i - 1)$  do
4:       if  $A[j] > A[j + 1]$  then
5:          $temp = A[j]$ 
6:          $A[j] = A[j + 1]$ 
7:          $A[j + 1] = temp$ 
8:       end if
9:     end for
10:  end for
11: end function
```

Insertion Sort

O algoritmo consiste em uma ordenação percorrendo o vetor da esquerda para direita, e conforme avança, vai ordenando os valores da sua esquerda. Possui um melhor caso em forma ordenada com complexidade de $O(n)$ e os casos medio e pior com complexidade de $O(n^2)$ [Santos and Tarôco].

Algorithm 2 Insertion Sort

```
1: function INSERTIONSORT( $A, p, r$ )
2:   for  $j = 2$  to  $n$  do
3:      $temp = A[j]$ 
4:      $i = j - 1$ 
5:     while  $(i > 0)$  and  $(A[i] > temp)$  do
6:        $A[i + 1] = A[i]$ 
7:        $i = i - 1$ 
8:     end while
9:      $A[i + 1] = temp$ 
10:  end for
11: end function
```

Quick Sort

Em primeiro momento iremos falar sobre a função partition que é de extrema importância para o Quick Sort.

A função partition elege um dos elementos da sequência como elemento pivô. Após essa eleição, cada elemento da sequência é comparado com o pivô e, se for menor que o pivô, deverá ficar à esquerda desse, se for maior que o pivô, deverá ficar à direita. Ao final dessa etapa, o elemento pivô fica na sua posição final da sequência ordenada [Prado 2005].

Essa função irá determinar o nível de balanceamento do quick sort que terá grande importância em seu desempenho.

Algorithm 3 Partition

```
1: function PARTITION( $A, p, r$ )
2:    $x = A[r]$ 
3:    $i = p - 1$ 
4:   for  $j = p$  to  $r - 1$  do
5:     if  $A[j] \leq x$  then
6:        $temp = A[i]$ 
7:        $A[i] = A[j]$ 
8:        $A[j] = temp$ 
9:     end if
10:  end for
11:   $temp = A[i + 1]$ 
12:   $A[i + 1] = A[r]$ 
13:   $A[r] = temp$ 
14:  return  $i + 1$ 
15: end function
```

O Quicksort é um algoritmo do tipo dividir para conquistar para ordenação cujo tempo de execução no pior caso é $O(n^2)$ sobre uma sequência de entrada de n números. O pior caso do Quicksort ocorre quando todos os particionamentos de todos os níveis de recursão ocorrem da pior maneira possível, ou seja, o elemento escolhido como pivô pela função particiona for sempre maior ou o menor elemento da sequência [Prado 2005].

O melhor caso do Quicksort ocorre quando todos os particionamentos em todos os níveis de recursão forem ótimos, ou seja, quando toda vez que a função particiona for chamada, ela divida a sequência exatamente ao meio ($\log_2 n$) [Prado 2005].

Algorithm 4 Quick Sort

```
1: function QUICKSORT( $A, p, r$ )
2:   if  $p < r$  then
3:      $q = PARTITION(A, p, r)$ 
4:      $QUICKSORT(A, p, q - 1)$ 
5:      $QUICKSORT(A, q + 1, r)$ 
6:   end if
7: end function
```

Referências

- da Silva Nascimento, J., Mozzaquatro, P. M., and Antoniazzi, R. L. (2016). Análise e comparação de algoritmos implementados em java. *Simpósio de Pesquisa e Desenvolvimento em Computação*, 1(1).
- Prado, J. A. S. (2005). Análise experimental do quicksort probabilístico com gerador de números pseudo-aleatórios penta-independente.
- Santos, E. and Tarôco, J. Uma comparação de algoritmos de ordenação baseados em comparação.