**Merge Sort**

Introdução

O ***merge sort***, ou [ordenação](http://pt.wikipedia.org/wiki/Ordena%C3%A7%C3%A3o_(computa%C3%A7%C3%A3o)) por mistura, é um exemplo de [algoritmo de ordenação](http://pt.wikipedia.org/wiki/Algoritmo_de_ordena%C3%A7%C3%A3o) do tipo [dividir-para-conquistar](http://pt.wikipedia.org/wiki/Divis%C3%A3o_e_Conquista" \o "Divisão e Conquista).

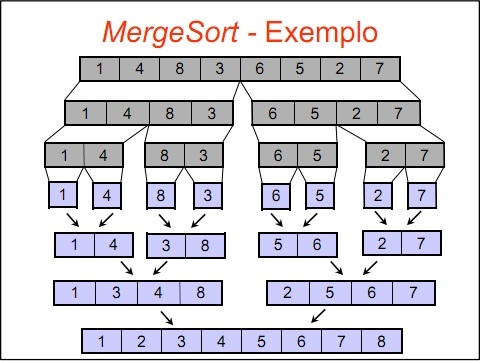
Sua ideia básica consiste em Dividir(o problema em vários sub-problemas e resolver esses sub-problemas através da recursividade) e Conquistar(após todos os sub-problemas terem sido resolvidos ocorre a conquista que é a união das resoluções dos sub-problemas).Como o algoritmo do Merge Sort usa a recursividade em alguns problemas esta técnica não é muito eficiente devido ao alto consumo de memória e tempo de execução.

Os três passos úteis dos algoritmos dividir-para-conquistar, ou *divide and conquer*, que se aplicam ao *merge sort* são:

1. Dividir: Dividir os dados em subsequências pequenas;
2. Conquistar: Classificar as duas metades recursivamente aplicando o *merge sort*;
3. Combinar: Juntar as duas metades em um único conjunto já classificado.

O Algoritmo e como funciona

Esse algoritmo de ordenação, consiste em quebrar todos os elementos pela metade, até com que eles fiquem todos separados, a partir daí, ordenando de par(es) em par(es)



Como vimos, ele pode parecer complicado no começo, mas na verdade ele é bem simples, apesar de ser bem mais complexo que o Bubble Sort.

Apesar desse algoritmo utilizar muitas funções recursivas, o que causa alto uso de memória, ele é considerado um algoritmo rápido.

Exemplo de código com método Marge Sort

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74 | inteiro [] sort(inteiro [] array)  inicio      se array.tamanho <= 1, então          retornar array      fim se        inteiro meio = array.tamanho/2      inteiro [meio] esq      inteiro [] dir        se array.tamanho for divisível por dois, então          dir = inteiro [meio];      fim se      senão          dir = inteiro [meio+1]        inteiro [array.tamanho] aux        para i de 0 até meio      faça          esq[i] = array[i]      fim para        int auxIndex = 0;        para i de meio até array.tamanho      faça          dir[auxIndex] = array[i]          auxIndex = auxIndex + 1      fim para        esq = sort(esq)      dir = sort(dir)        aux = mergesort(esq, dir)        retornar aux  fim        inteiro [] mergesort(inteiro [] esq, inteiro [] dir)  inicio      inteiro [esq.tamanho+dir.tamanho] aux        inteiro indexDir = 0, indexEsq=0, indexAux = 0        enquanto indexEsq < esq.tamnho ou indexDir < dir.tamanho      faça          se indexEsq < esq.tamnho e indexDir < dir.tamanho, então              se esq[indexEsq] <= dir[indexDir], então                  aux[indexAux] = esq[indexEsq]                  indexAux = indexAux + 1                  indexEsq = indexEsq + 1              fim se              senão                  aux[indexAux] = dir[indexDir]                  indexAux = indexAux + 1                  indexDir = indexDir + 1          fim se          senão se indexEsq < esq.tamanho, então              aux[indexAux] = esq[indexEsq]              indexAux = indexAux + 1              indexEsq = indexEsq + 1          fim se          senão se indexDir < dir.tamanho, então              aux[indexAux] = dir[indexDir]              indexAux = indexAux + 1              indexDir = indexDir + 1          fim se      fim enquanto        retornar aux  fim |

Vantagens

● Útil para ordenação externa;

● Pior caso: O(n log2 n);

● Aplicações com restrição de tempo;

● Fácil implementação.

Desvantagens

● Utiliza memória auxiliar;

● Alto consumo de memória.

Fontes

http://www.academia.edu

http://www.wikipedia.org