Nome: Lucas Alves da Costa.

Disciplina: Estrutura de dados I

Data: 13/12/2017

MergeSort.

Criado por Von Neumann em 1945 o ***mergesort*** se baseia no conceito de (merge = junção) de duas sequencias ordenadas. Primeiramente ele subdivide a sequência original na metade e ordena recursivamente essas sequencias. Por fim faz a junção das duas sequências ordenadas para compor uma sequência maior ordenada.

Exemplo; (1,3,5,7,9) + (0,2,4,6,8) *merge* →(0,1,2,3,4,5,6,7,8,9)

A relação de recorrência do ***mergesort*** corresponde a;

$$T(n) = 2 \cdot T(n/2) + O(n) \in \Theta(nlgn)$$

- *Estável: [v]*
- *In-place: [x]*

Esse esquema de ordenação por mistura consiste basicamente em dividir para conquistar, ou seja, dividir o problema em vários subproblemas e resolve-los recursivamente. Como o mergesort usa a recursividade, há um alto consumo de memória a tempo de execução, tornado se eficiente em muitos casos.

Esse algoritmo usa três passos uteis:

I. **Dividir**: Calcula o ponto médio do sub-arranjo, o que demora um tempo constante $\boldsymbol{\Theta(1)}$**;**

II. **Conquistar**: Recursivamente resolve dois subproblemas, cada um de tamanho ***n/2***, o que contribui com; $\boldsymbol{2 \cdot T(n/2)}$ para o tempo de execução;

III. **Combinar:** Unir os sub-arranjos em um único conjunto ordenado, que leva o tempo $\boldsymbol{\Theta(n)}$**;**
Sua complexidade de tempo é; $\boldsymbol{\Theta(nlgn)}$ e sua complexidade de espaço é; $\boldsymbol{\Theta(n)}$**.**

Em comparação com o Quick sort o mergesort tem a mesma complexidade, mas se comparado com algoritmos básicos de comparação e troca, o Mergesort é mais rápido e eficiente quando aplicado em grandes quantidades de dados.

Suas desvantagens são:

- Utiliza recursividade nas funções.
- Tem gasto extra de memória pois cria uma cópia do vetor a cada recursão.

Exemplo;

Um vetor de números inteiros desordenados: esse vetor pode ser facilmente dividido em dois, ficando desta forma mais fácil de ser ordenado.

Vetor original.

1	0	6	8	4	-2	21	3	5	2

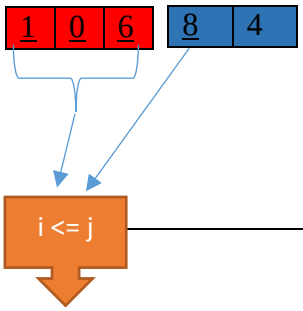
Vetor dividido em duas partes.

1	0	6	8	4

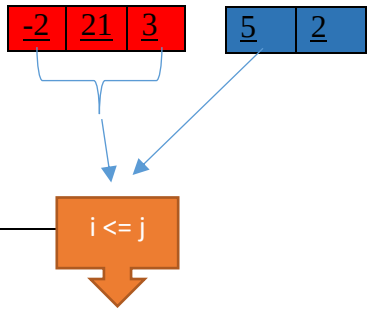
-2	21	3	5	2

Agora cada parte é subdividida em duas;

Exemplo A:



Exemplo B:



Vetor principal.

1	0	6	-2	5	2	8	4	21	3

Para facilitar, vamos nomear o vetor da esquerda de RED e o vetor da direita de BLUE. Note que no **exemplo (A)** todos os valores do vetor RED foram comparados com apenas o primeiro valor ***(8)*** que está no vetor BLUE, usamos a condição **`(i <= j)`** onde **`i`** é um elemento qualquer do vetor RED e **`j`** é outro elemento qualquer que está sendo comparado. Já no **exemplo (B)** descobrimos que o segundo valor ***(21)*** do vetor RED é maior do que o primeiro elemento ***(5)*** do vetor BLUE, já que o mesmo é menor envio esse elemento para o vetor principal, a partir daí os próximos elementos do vetor RED vão ser comparados com o próximo elemento ***(2)*** do vetor BLUE, como estamos procurando sempre os menores valores não importa em qual dos vetores o valor está, o foco é sempre copiar este valor para o vetor principal.

Em seguida foram copiados os valores restantes que não atenderam o requisito de comparação para o vetor principal, logo após, o vetor será novamente dividido e subdividido recursivamente até que o mesmo esteja totalmente ordenado, com esse exemplo fica mais fácil compreender a ideia de dividir para conquistar pois o foco aqui é dividir o vetor em pequenas partes para facilitar o processo de ordenação.