Mergesort: ordenação por intercalação

Fabio Lubacheski
fabio.aglubacheski@sp.senac.br

MergeSort

A ordenação por intercalação (*merge*) utiliza a estratégia **recursiva** de **divisão-e-conquista** para ordenar um vetor `v[0..n-1]`,

de tal modo que tenhamos `v[0]`≤`. . .`≤`v[n-1]`. considere que **`n`** é o numero de elementos de `v[]`.

- **Divisão**: se `v[]` tem zero ou 1 elemento, `v[]` já está ordenado. Caso contrário, particionamos `v[]` o vetor ao meio, gerando dois subvetores, cada um com `n/2` elementos.
- **Recursão**: recursivamente, aplica-se a estratégia para os dois subvetores.
- **Conquista**: faz-se a **intercalação** dos subvetores ordenados, produzindo a versão ordenada do vetor `v[]`.

Simulação do MergeSort

https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

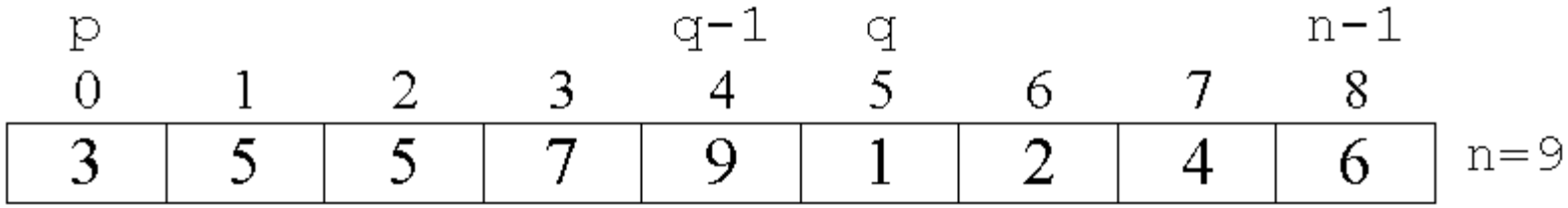
Antes de mais nada, vamos definir a Intercalação.

A **intercalação** é o processo utilizado para construir um vetor ordenado, de tamanho *n*+*m*, a partir de dois vetores já ordenados de tamanhos *n* e *m.*

Escreva uma função que receba dois vetores (A[] e B[]), com *n* e *m* elementos, respectivamente. Os vetores estão ordenados em ordem crescente, a função aloca um vetor C[], exatamente com soma dos tamanhos de A e B, e intercala os elementos de A[] e B[] em C[], de forma que o vetor C[] fique em ordem crescente. A função deve ter deve ter no máximo ***n+m*** **passos**, ou seja, a soma dos tamanho dos vetores.

Intercalação de subvetores ordenados

Agora podemos resolver o problema de intercalar os subvetores ordenados, ou seja: dados subvetores crescentes `v[p..q-1]` e `v[q..n-1]`, rearranjar `v[p..n-1]` em ordem crescente.

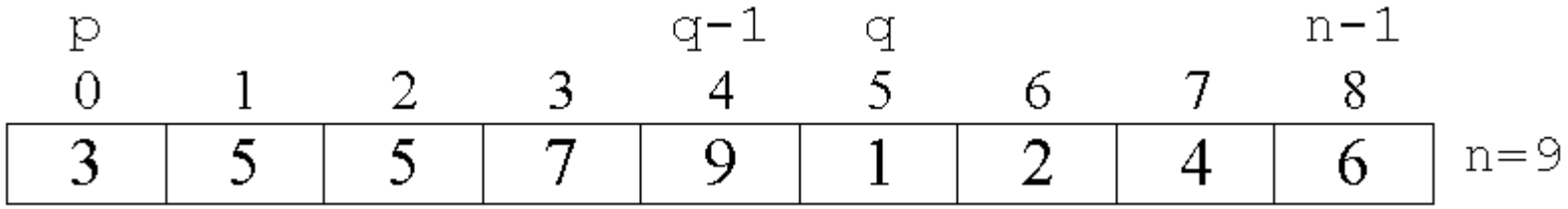


O problema parece fácil, mas não é trivial. Será preciso usar um vetor auxiliar, digamos `w`, do mesmo tipo e mesmo tamanho que `v[p..n-1]`, pois os vetores a serem intercalados estão no mesmo vetor, sendo somente regiões diferentes do vetor `v[]`. O processo de intercalação deve ser realizado em **n passos**, onde **n** é o tamanho do vetor.

Intercalação de vetores ordenados

A declaração da função intercalação poderia ser conforme abaixo:

```
void intercala(int v[], int p, int q, int n)
```



Para a configuração do vetor acima a chamada da função poderia ser: `intercala(v,0,5,9);`

Mergesort

Agora podemos usar a função intercala discutida anteriormente para escrever a função **MergeSort**

```
void MergeSort(int v[], int p, int n)
```

A função **MergeSort** é recursiva e reduz o tamanho do vetor a cada chamada. A função recebe no inicio um vetor com n elementos de p até n-1, ou seja, v[p..n-1], a função rearranja o vetor em ordem crescente, a primeira chamada da função MergeSort seria:

```
int v[]={9,8,7,6,5,4,3,2,1};
MergeSort(v, 0, v.length);
```

Mergesort

A função **MergeSort** só faz as chamadas recursivas

se `p<n-1`, pois se `p` igual a `n-1` (=base da recursão) teremos somente um elemento no vetor e não é preciso fazer nada.

Caso `p<n-1` a função MergeSort divide o vetor em duas partes `v[p..q-1]` e `v[q..n-1]`, para isso o calculo de q é dado por:

```
q=(p + n)/2 #divisão inteira
```

Em seguida é chamado recursivamente para esquerda `MergeSort(v,p,q)` e para direita `MergeSort(v,q,n)`, ao no final intercala intercala(v,p,q,n) o resultado dos dois vetores `v[p..q-1] e v[q..n-1]`.

Exercícios

1) Execute o algoritmo (Diagrama de Execução) MergeSort com a entrada `v[]={7,8,3,5,4}`

   Para testar o algoritmo passo a passo use o site abaixo:

   https://cscircles.cemc.uwaterloo.ca/java_visualize/#mode=display

2) Escreva uma versão iterativa para o método de ordenação MergeSort.

Fim