

NOME: Julio Cesario de Paiva Leao
RA: 1916033
Engenharia de Software 2018/01
Estrutura de Dados

A04 - Exercícios - Ordenação de Dados - Parte 1

Dado o seguinte algoritmo de ordenação:

```
void sort(int *vet, int tam){
    int aux, i, j, mid = tam/2;

    while(mid > 0){
        i = mid;
        cont++;
        while(i < tam){
            aux = vet[i];
            j = i;
            cont++;
            while(j >= mid && aux < vet[j-mid]){
                vet[j] = vet[j-mid];
                j = j- mid;
                cont++;
            }
            vet[j] = aux;
            i++;
        }
        mid = mid/2;
    }
}
```

Temos as seguintes variáveis de inteiro:

Variáveis	Descrição
int *vet;	Ponteiro que pre aloca memoria com a função malloc através de um tamanho que o usuário informa, transformando-o assim num vetor.
int tam;	Tamanho do vetor informado pelo usuário.
int aux;	Variável auxiliar para o armazenamento temporário de variáveis que realizarão a troca.
int i;	Índice do vetor no qual sera percorrido o vetor.
int j;	Será o responsável por comparar as posições do vetor e identificar o maior e o menor valor.

<code>int mid = tam/2;</code>	<code>mid</code> recebe a metade do tamanho total do vetor, ou seja <code>tam</code> (informado pelo usuário) / 2.
-------------------------------	--

O que cada linha da função faz?

Linha	Descrição
<code>void sort(int *vet, int tam)</code>	Declaração de função sem retorno <i>sort</i> , recebendo como parâmetros um ponteiro de inteiro <i>*vet</i> e um inteiro <i>tam</i> (tamanho do vetor informado pelo usuário).
<code>int aux, i, j, mid = tam/2;</code>	Declaração de variáveis locais.
<code>while(mid > 0)</code>	Primeiro laço de repetição onde fara o seguinte teste: Enquanto <i>mid</i> (metade do vetor) for maior que <i>0</i> , repita.
<code>i = mid;</code>	Variável <i>i</i> recebe <i>mid</i> (posição central do vetor)
<code>while(i < tam){</code>	Segundo laço de repetição, onde testará se o índice <i>i</i> é menor que <i>tam</i> (tamanho total do vetor)
<code>aux = vet[i];</code>	Variável auxiliar (<i>aux</i>) recebe o valor do vetor na posição <i>i</i> , ou seja, o valor da posição central do vetor.
<code>j = i;</code>	Segundo índice, <i>j</i> , recebe o valor de <i>i</i> , onde <i>i</i> é a posição central do vetor.
<code>while(j >= mid && aux < vet[j-mid])</code>	Terceiro e último laço de repetição, onde finalmente sera feita a seleção dos valores. Será executado enquanto <i>j</i> (posição central do vetor) for maior ou igual que o próprio <i>mid</i> (meio do vetor) E a variável <i>aux</i> for menor que o vetor na posição <i>0</i> , uma vez que <i>j</i> recebeu o valor de <i>i</i> , e <i>i</i> continha o valor de <i>mid</i> , em outras palavras <i>j</i> == <i>mid</i> .
<code>vet[j] = vet[j-mid];</code>	Após entrar no laço o vetor na posição central [<i>j</i>] receberá vetor na posição 0 [<i>j-mid</i>].

<code>j = j- mid;</code>	O índice <i>j</i> receberá o valor <i>0</i> ;
<code>vet[j] = aux;</code>	E o vetor na primeira posição <i>vet[j]</i> receberá o valor antes armazenado à variável <i>aux</i> .
<code>i = i+1;</code>	Incrementa-se o índice <i>i</i> em mais <i>1</i> , para que o laço não entre em <i>loop infinito</i> , e prossiga com as operações
<code>mid = mid/2;</code>	Finalizando, a variável <i>mid</i> recebe metade dela mesma.

Demonstração gráfica:

Usemos um vetor :

```
int v = {10, 3, 19, 7, 11, 38}, tam = 6;
```

posições(tam)	0	1	2	3	4	5
Valores v[6]	10	3	19	7	11	38

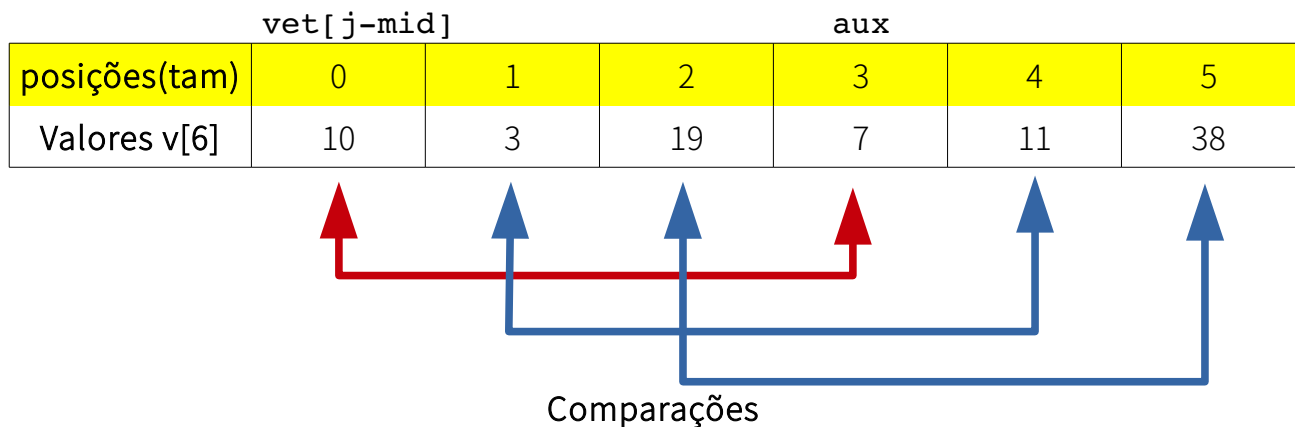
mid = tam / 2

mid == 3

```

i = mid
aux = vet[i]
aux == 7
j = i

```



1º teste

Aux < vet[j-mid]

Ou

7 < 10 (Como 7 é menor que 10 então entra no laço)

Então

vet[j] = vet[j-mid]

Ou

vet[3] = vet[0] (realiza a troca trazendo o valor da posição 0, valor 10, para a posição 3)

Ou

Vet[3] = 10

j = j - mid

Ou

J = 3 - 3 == 0 (subtrai mid de j fazendo ele receber o valor 0 fazendo sair do laço)

vet[j] = aux

Ou

Vet[0] = 7 (vetor na posição inicial recebe o valor antes armazenado em aux, 7)

i++

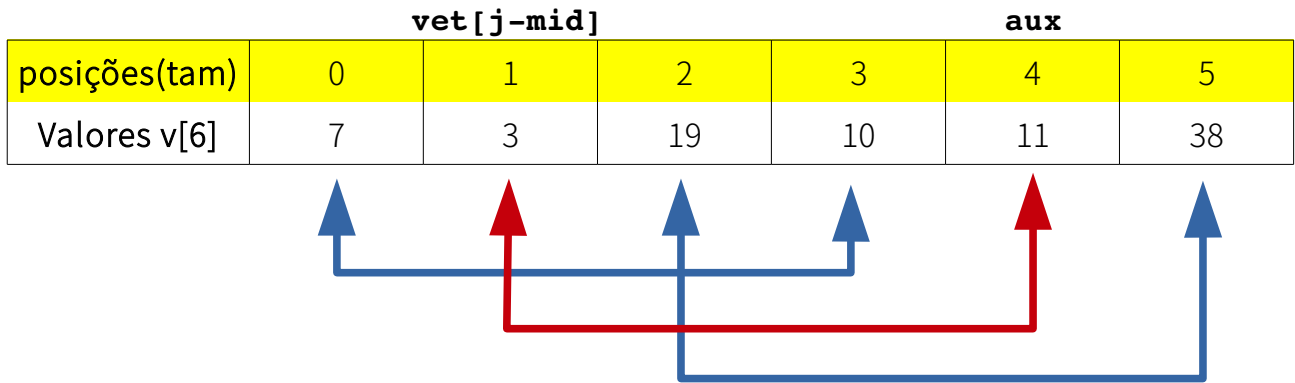
Ou

I == 4 (próxima posição do vetor onde fará uma nova comparação)

```

aux = vet[i]
aux == 11
j = i

```



2º teste

```
Aux < vet[j-mid]
```

Ou

11 < 3 (Como 11 NÃO é menor que 3 então não entra no laço e não precisará fazer a troca)

```
vet[j] = aux
```

Ou

Vet[4] = 11 (vetor na posição 4 recebe o valor antes armazenado em aux, 11)

```
i++
```

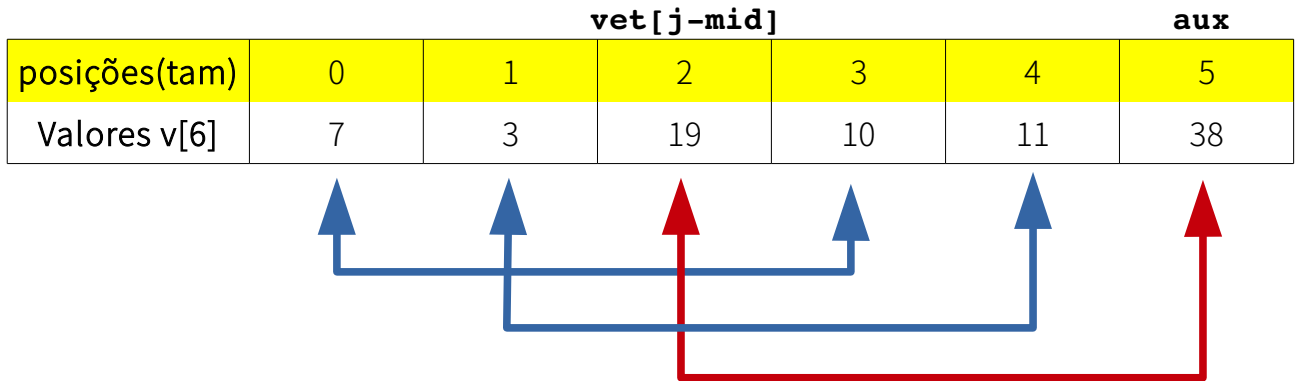
Ou

I == 5 (próxima posição do vetor onde fará uma nova comparação)

```

aux = vet[i]
aux == 38
j = i

```



3º teste

```
Aux < vet[j-mid]
```

Ou

38 < 19 (Como 38 **NÃO** é menor que 19 então não entra no laço e não precisará fazer a troca)

```
vet[j] = aux
```

Ou

Vet[5] = 38 (vetor na posição 4 recebe o valor antes armazenado em aux, 38)


```
i++
```

Ou


i == 6 (como i agora é igual a tam, sai do segundo laço)

mid = tam / 2
mid == 1


posições(tam)	0	1	2	3	4	5
Valores v[6]	7	3	19	10	11	38




posições(tam)	0	1	2	3	4	5
Valores v[6]	3	7	19	10	11	38




posições(tam)	0	1	2	3	4	5
Valores v[6]	3	7	19	10	11	38



posições(tam)	0	1	2	3	4	5
Valores v[6]	3	7	10	19	11	38



posições(tam)	0	1	2	3	4	5
Valores v[6]	3	7	10	11	19	38



Vetor Organizado!!!

Enquanto que com o mesmo vetor o método *Bubblesort* realiza **36** comparações, esse método realiza apenas **14** comparações.