

## GSI011 – Estrutura de Dados 2

Prof<sup>a</sup>: Christiane Regina Soares Brasil

# GSI011 – Estrutura de Dados 2 – Aula 6

- Algoritmos de ordenação podem ser divididos em:
  - **Básicos:** implementação mais simples, menos eficientes
    - *Bubble Sort, Selection Sort e Insertion Sort.*
  - **Sofisticados:** implementação não trivial, mais eficientes
    - *Merge Sort, Quick Sort.*

# GSI011 – Estrutura de Dados 2 – Aula 6

- Algoritmos Sofisticados de Ordenação: *Merge Sort*
  - Também conhecido como **ordenação por intercalação**
  - Algoritmo recursivo que usa estratégia de “divisão e conquista”

# GSI011 – Estrutura de Dados 2 – Aula 6

- Algoritmos de Ordenação Sofisticados: *Merge Sort*

```
void mergeSort(int* v, int in, int fim)
{
    int meio;
    if(in < fim )
    {
        meio = (in + fim)/2;
        mergeSort (v, in, meio);//metade da esquerda
        mergeSort(v, meio + 1, fim);//metade da direita
        //intercala as duas metades ordenadas
        merge(v, in, meio, fim);
    }
}
```

Utiliza “dividir para conquistar”:  
aplica em cada metade a ordenação e depois intercala os subvetores ordenados.

```
void merge(int* v, int in, int meio, int fim)
```

```
{
```

```
    int *aux, p1, p2, tam, i, j, k, f1, f2;
```

```
    f1 = f2 = 0;
```

```
    tam = fim - in + 1;
```

```
    p1 = in;
```

```
    p2 = meio + 1;
```

```
    aux = (int*) malloc(tam*sizeof(int));
```

```
    if(aux!=NULL)
```

```
    {
```

```
        for(i=0;i<tam;i++)
```

```
        {
```

```
            if(f1==0 && f2==0)//nenhum dos subvetores acabou
```

```
            { if(v[p1]<v[p2])
```

```
                { aux[i] = v[p1]; p1++;}
```

```
            else
```

```
                { aux[i] = v[p2]; p2++;}
```

```
            if(p1>meio) f1 = 1;//vetor acabou?
```

```
            if(p2>fim) f2 = 1;
```

```
        }
```

```
    }else//qual subvetor acabou?
```

```
    {
```

```
        if(f1 == 1)
```

```
        { aux[i] = v[p2]; p2++;}
```

```
        else
```

```
        { aux[i] = v[p1]; p1++;}
```

```
    }
```

```
    }//fim for
```

```
    for(j=0; k=in; j < tam; j++; k++)
```

```
        v[k] = aux[j];
```

```
    }
```

```
    free(aux);
```

```
}
```

23 4 67 -8 90 54 21

23 4 67 -8

in meio fim

in

meio

fim

90 54 21

```
void mergeSort(int* v, int in, int fim)
{
    int meio;
    if(in < fim )
    {
        meio = (in + fim)/2;
        mergeSort (v, in, meio);
        mergeSort(v, meio + 1, fim);
        merge(v, in, meio, fim);
    }
}
```

23 4 67 -8 90 54 21

23 4 67 -8

in meio fim

in

meio

fim

90 54 21

23 4

in fim  
meio

```
void mergeSort(int* v, int in, int fim)
{
    int meio;
    if(in < fim )
    {
        meio = (in + fim)/2;
        mergeSort (v, in, meio);
        mergeSort(v, meio + 1, fim);
        merge(v, in, meio, fim);
    }
}
```

23 4 67 -8 90 54 21

23 4 67 -8

in meio fim

in

meio

fim

90 54 21

23 4

in fim  
meio

p1 p2 p2

23

4

in in  
fim fim

tam = 2

f1 = 0

f2 = 0 1

4 23

aux

```
if(aux!=NULL)
{
    for(i=0;i<tam;i++)
    {
        if(f1==0 && f2==0)
        {
            if(v[p1]<v[p2])
            { aux[i] = v[p1]; p1++;}
            else
            { aux[i] = v[p2]; p2++;}
            if(p1>meio) f1 = 1;
            if(p2>fim) f2 = 1;
        }
    }
}
```

```
else//qual subvetor acabou?
{
    if(f1 == 1)
    { aux[i] = v[p2]; p2++;}
    else
    { aux[i] = v[p1]; p1++;}
}
} //fim for
for(j=0; k=in; j < tam; j++; k++)
    v[k] = aux[j];
}
free(aux);
}
```



23 4 67 -8 90 54 21

23 4 67 -8

in

meio

fim

90 54 21

in meio fim

23 4

67 -8

in fim  
meio

in fim  
meio

23

4

in in  
fim fim

tam = 2  
f1 = 0  
f2 = 0 1

4 23

```
if(aux!=NULL)
{
    for(i=0;i<tam;i++)
    {
        if(f1==0 && f2==0)
        {
            if(v[p1]<v[p2])
            { aux[i] = v[p1]; p1++;}
            else
            { aux[i] = v[p2]; p2++;}
            if(p1>meio) f1 = 1;
            if(p2>fim) f2 = 1;
        }
    }
}
```

```
else//qual subvetor acabou?
{
    if(f1 == 1)
    { aux[i] = v[p2]; p2++;}
    else
    { aux[i] = v[p1]; p1++;}
}
} //fim for
for(j=0; k=in; j < tam; j++; k++)
    v[k] = aux[j];
}
free(aux);
}
```

23 4 67 -8 90 54 21

23 4 67 -8

in meio fim

4 23

67 -8

in fim  
meio

67

-8

in in  
fim fim

90 54 21

23 4 67 -8 90 54 21

23 4 67 -8

in meio fim

4 23

67 -8

in fim  
meio

67

-8

in in  
fim fim

tam = 2

f1 = 0

f2 = 0 1

-8 67

in

meio

fim

90 54 21

```
if(aux!=NULL)
{
    for(i=0;i<tam;i++)
    {
        if(f1==0 && f2==0)
        {
            if(v[p1]<v[p2])
            { aux[i] = v[p1]; p1++;}
            else
            { aux[i] = v[p2]; p2++;}
            if(p1>meio) f1 = 1;
            if(p2>fim) f2 = 1;
        }
    }
}
```

```
else//qual subvetor acabou?
{
    if(f1 == 1)
    { aux[i] = v[p2]; p2++;}
    else
    { aux[i] = v[p1]; p1++;}
}
} //fim for
for(j=0; k=in; j < tam; j++; k++)
    v[k] = aux[j];
}
free(aux);
}
```

23 4 67 -8 90 54 21

23 4 67 -8

in meio fim

in

meio

fim

90 54 21

tam = 4  
f1 = 0 1  
f2 = 0

p1

p1 p1 p2 p2 p2

4 23 -8 67

in meio fim

-8 4 23 67

```
if(aux!=NULL)
{
    for(i=0;i<tam;i++)
    {
        if(f1==0 && f2==0)
        {
            if(v[p1]<v[p2])
            { aux[i] = v[p1]; p1++;}
            else
            { aux[i] = v[p2]; p2++;}
            if(p1>meio) f1 = 1;
            if(p2>fim) f2 = 1;
        }
    }
}
```

```
else//qual subvetor acabou?
{
    if(f1 == 1)
    { aux[i] = v[p2]; p2++;}
    else
    { aux[i] = v[p1]; p1++;}
}
} //fim for
for(j=0; k=in; j < tam; j++; k++)
    v[k] = aux[j];
}
free(aux);
}
```

23 4 67 -8 90 54 21

-8 4 23 67

in meio fim

in

meio

fim

90 54 21

90 54

in fim  
meio

23 4 67 -8 90 54 21

23 4 67 -8

in meio fim

in

meio

fim

90 54 21

```
if(aux!=NULL)
{
    for(i=0;i<tam;i++)
    {
        if(f1==0 && f2==0)
        {
            if(v[p1]<v[p2])
            { aux[i] = v[p1]; p1++;}
            else
            { aux[i] = v[p2]; p2++;}
            if(p1>meio) f1 = 1;
            if(p2>fim) f2 = 1;
        }
    }
}
```

```
else//qual subvetor acabou?
{
    if(f1 == 1)
    { aux[i] = v[p2]; p2++;}
    else
    { aux[i] = v[p1]; p1++;}
}
} //fim for
for(j=0; k=in; j < tam; j++; k++)
    v[k] = aux[j];
}
free(aux);
}
```

90 54

in fim  
meio

90 54

in in  
fim fim

tam = 2  
f1 = 0  
f2 = 0 1

54 90

23 4 67 -8 90 54 21

23 4 67 -8

in meio fim

in

meio

fim

90 54 21

```
for(i=0;i<tam;i++)
{
    if(f1==0 && f2==0)
    {
        if(v[p1]<v[p2])
        { aux[i] = v[p1]; p1++;}
        else
        { aux[i] = v[p2]; p2++;}
        if(p1>meio) f1 = 1;
        if(p2>fim) f2 = 1;
    }
}
```

```
else{//descobre qual vetor acabou
{
    if(f1 == 1)
    { aux[i] = v[p2]; p2++;}
    else
    { aux[i] = v[p1]; p1++;}
}
}
for(j=0; k=in; j < tam; j++; k++)
    v[k] = aux[j];
free(aux);
}
```

54 90

21

in  
fim

23 4 67 -8 90 54 21

23 4 67 -8

in meio fim

in

meio

fim

90 54 21

p1

```
if(aux!=NULL)
{
    for(i=0;i<tam;i++)
    {
        if(f1==0 && f2==0)
        {
            if(v[p1]<v[p2])
            { aux[i] = v[p1]; p1++;}
            else
            { aux[i] = v[p2]; p2++;}
            if(p1>meio) f1 = 1;
            if(p2>fim) f2 = 1;
        }
    }
}
```

```
else//qual subvetor acabou?
{
    if(f1 == 1)
    { aux[i] = v[p2]; p2++;}
    else
    { aux[i] = v[p1]; p1++;}
}
} //fim for
for(j=0; k=in; j < tam; j++; k++)
    v[k] = aux[j];
}
free(aux);
}
```

tam = 3  
f1 = 0  
f2 = 0 1

p1 p2 p2

54 90 21

in meio fim

21 54 90



23	4	67	-8	90	54	21
----	---	----	----	----	----	----

in

meio

fim

tam = 7

f1 = 0 1

f2 = 0

p1

p1 p1 p1 p1 p2 p2 p2

-8	4	23	67	21	54	90
----	---	----	----	----	----	----

in

meio

fim

-8	4	21	23	54	67	90
----	---	----	----	----	----	----

- Algoritmos de Ordenação Sofisticados: *Merge Sort*
  - Complexidade
    - $O(n \log n)$ : em todos os casos
    - Não depende da ordem inicial do vetor

- Estável
- Gasto com memória extra (vetor auxiliar)

# GSI011 – Estrutura de Dados 2 – Aula 6

- Algoritmos de Ordenação Sofisticados: *Quick Sort*

```
void quickSort(int* v, int in, int fim)
{
    int pospivo;
    if(in < fim)
    {
        pospivo = particiona(v, in, fim);
        quickSort(v, in, pospivo - 1); //lado esquerdo do pivô
        quickSort(v, pospivo + 1, fim); //lado direito do pivô
    }
}
```

Também conhecido como algoritmo de partição. Usa estratégia de “divisão e conquista”.

# GSI011 – Estrutura de Dados 2 – Aula 6

```
void particiona(int* v, int in, int fim)
{
    int esq, dir, pivo, aux;
    esq = in;  dir = fim;
    pivo = v[in];
    while(esq < dir )
    {
        while(v[esq] <= pivo && esq<=fim) esq++;
        while(v[dir] > pivo && dir >= in) dir--;
        if (esq < dir)
        { aux = v[esq];
          v[esq] = v[dir];
          v[dir] = aux;
        }
    }
    v[in] = v[dir];
    v[dir] = pivo;
    return dir; }
```

23 4 67 -8 90 54 21

esq esq esq

dir

pivo = 23

23 4 21 -8 90 54 67

esq esq esq dir dir  
dir dir

-8 4 21 23 90 54 67

```
void particiona(int* v, int in, int fim)
{
    int esq, dir, pivo, aux;
    esq = in; dir = fim;
    pivo = v[in];
    while(esq < dir )
    {
        while(v[esq] <= pivo
            && esq<=fim) esq++;
        while(v[dir] > pivo
            && dir >= in) dir--;
        if (esq < dir)
        { aux = v[esq];
          v[esq] = v[dir];
          v[dir] = aux;
        }
    }
    v[in] = v[dir];
    v[dir] = pivo;
    return dir; }
```

23 4 67 -8 90 54 21

pivo = -8

-8 4 21 23 90 54 67

esq esq dir

dir dir

pivo = 4

-8 4 21

esq esq

dir dir

-8 4 21

```
void particiona(int* v, int in, int fim)
{
    int esq, dir, pivo, aux;
    esq = in; dir = fim;
    pivo = v[in];
    while(esq < dir )
    {
        while(v[esq] <= pivo
            && esq<=fim) esq++;
        while(v[dir] > pivo
            && dir >= in) dir--;
        if (esq < dir)
        { aux = v[esq];
          v[esq] = v[dir];
          v[dir] = aux;
        }
    }
    v[in] = v[dir];
    v[dir] = pivo;
    return dir; }
```

```

void particiona(int* v, int in, int fim)
{
    int esq, dir, pivo, aux;
    esq = in;  dir = fim;
    pivo = v[in];
    while(esq < dir )
    {
        while(v[esq] <= pivo
            && esq<=fim) esq++;
        while(v[dir] > pivo
            && dir >= in) dir--;
        if (esq < dir)
        { aux = v[esq];
          v[esq] = v[dir];
          v[dir] = aux;
        }
    }
    v[in] = v[dir];
    v[dir] = pivo;
    return dir; }

```

23 4 67 -8 90 54 21

pivo = 90

-8 4 21 23 90 54 67

esq esq esq esq  
dir

67 54 90

esq esq esq  
dir

pivo = 67

54 67 90

23 4 67 -8 90 54 21

esq esq esq

dir

pivo = 23

23 4 21 -8 90 54 67

esq esq esq dir dir

dir dir

pivo = -8

-8 4 21 23 90 54 67

esq esq dir dir esq esq esq

dir esq

dir

-8 4 21

esq esq  
dir dir

pivo = 90

67 54 90

esq esq  
dir

pivo = 4

-8 4 21

23

pivo = 67

54 67 90

-8 4 21 23 54 67 90



- Algoritmos de Ordenação Sofisticados: *Quick Sort*
  - Complexidade
    - $O(n \log n)$ , melhor caso e caso médio.
    - $O(n^2)$ , pior caso: pivo escolhido fica sempre nas extremidades.

- Não é estável
- *Insertion Sort* é melhor quando o vetor já está ordenado ( $O(n)$ )