Classificação e Pesquisa de Dados

Cristiano Santos
cristiano.santos@amf.edu.br

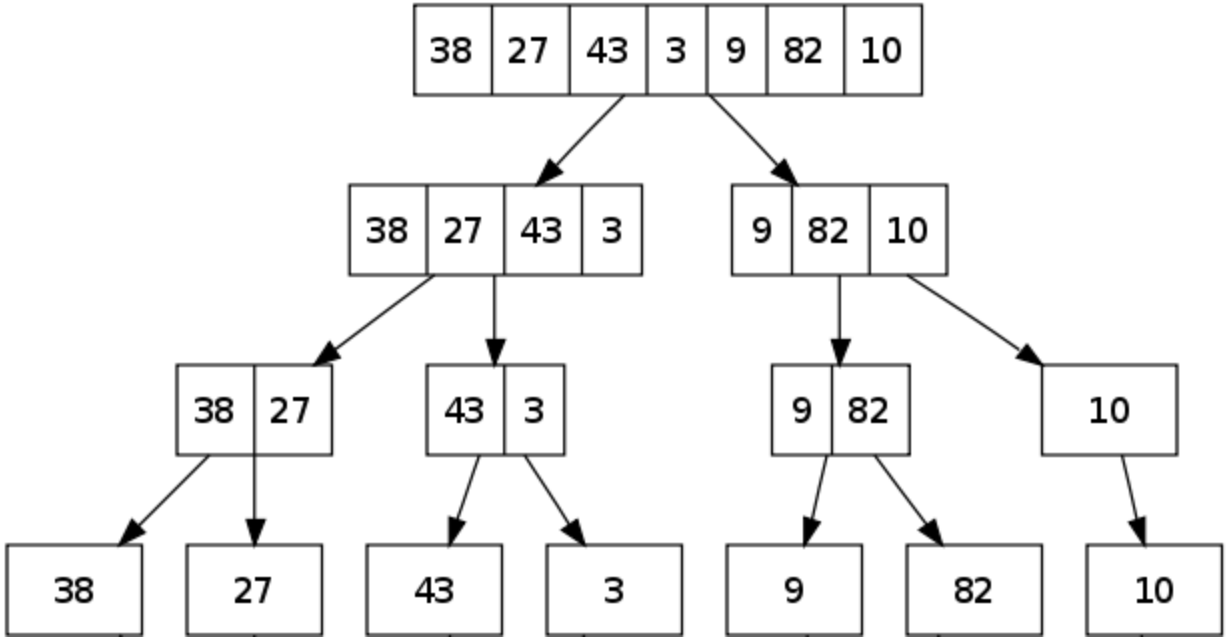
AULA 02

Merge Sort

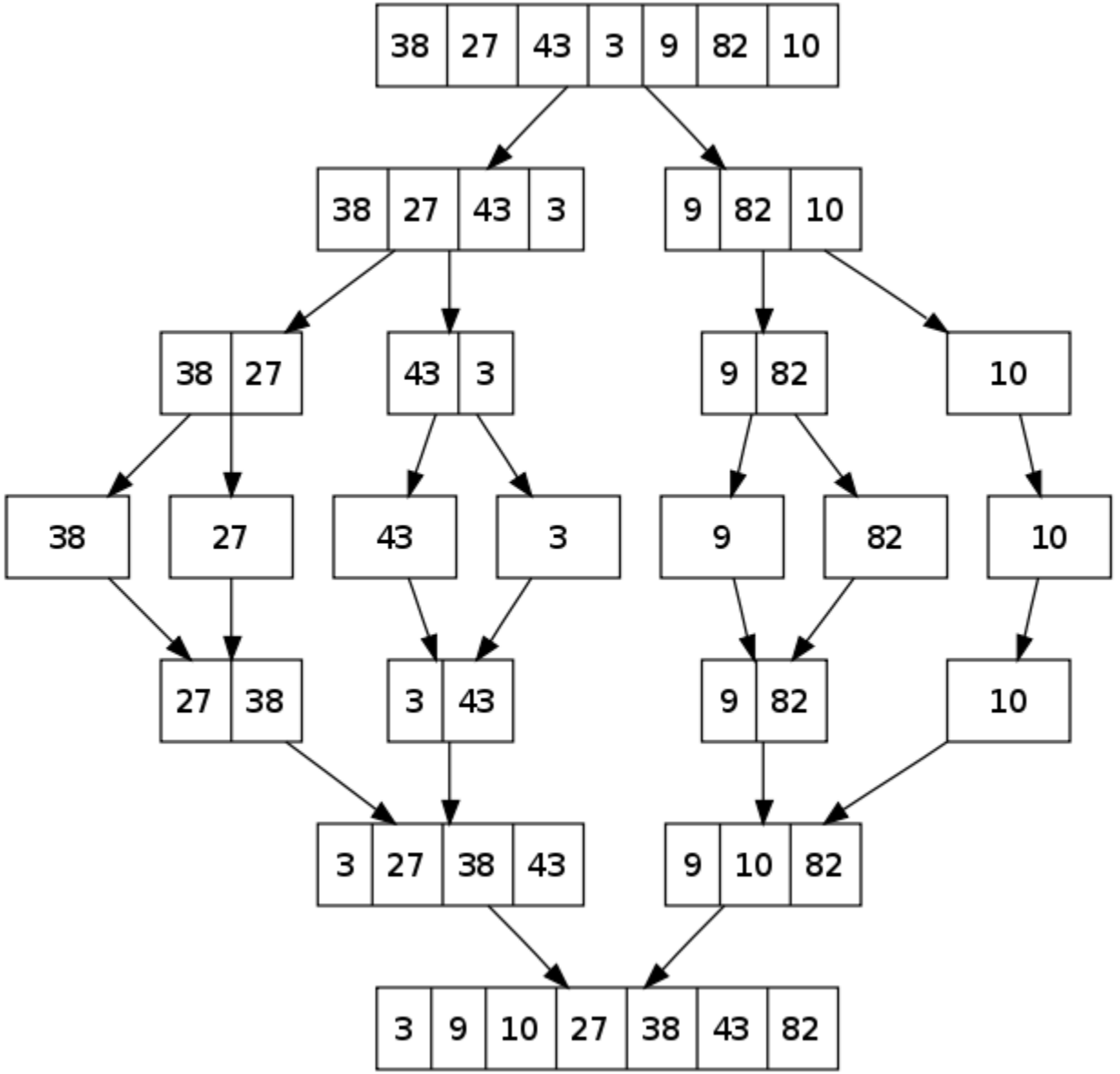
Merge Sort

- Merge Sort ou Ordenação por Mistura
  - Divide a lista em duas partes iguais (Dividir e Conquistar)
    - Continua dividindo (Recursivamente) até ter 1 elemento
    - Combina 2 conjuntos de forma a obter 1 conjunto maior e ordenado
    - Processo se repete até que exista apenas 1 conjunto.

Merge Sort



Merge Sort


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

Merge Sort

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

}
```

Merge Sort

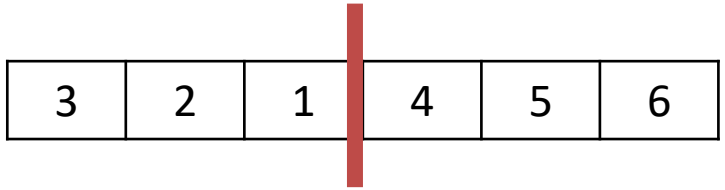
```
void mergeSort(int *v, int inicio, int fim){
        int meio;
        if (inicio < fim){
                meio = floor((inicio + fim)/2);
                mergeSort(v, inicio, meio);
                mergeSort(v, meio+1, fim);
                merge(v, inicio, meio, fim)

}
```

Divisão

Conquista

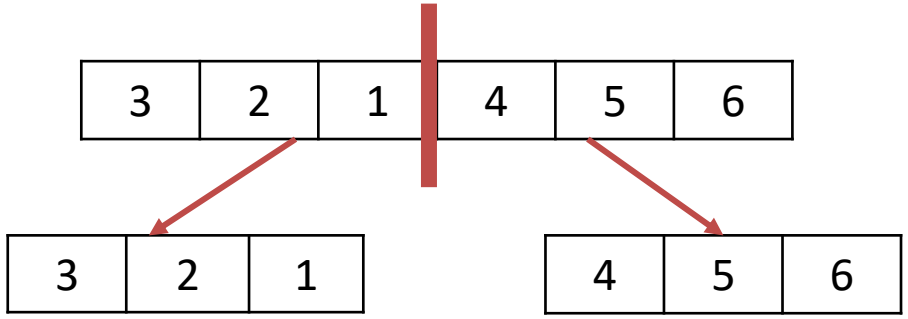
Merge Sort

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


}
```



Merge Sort

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

}
```

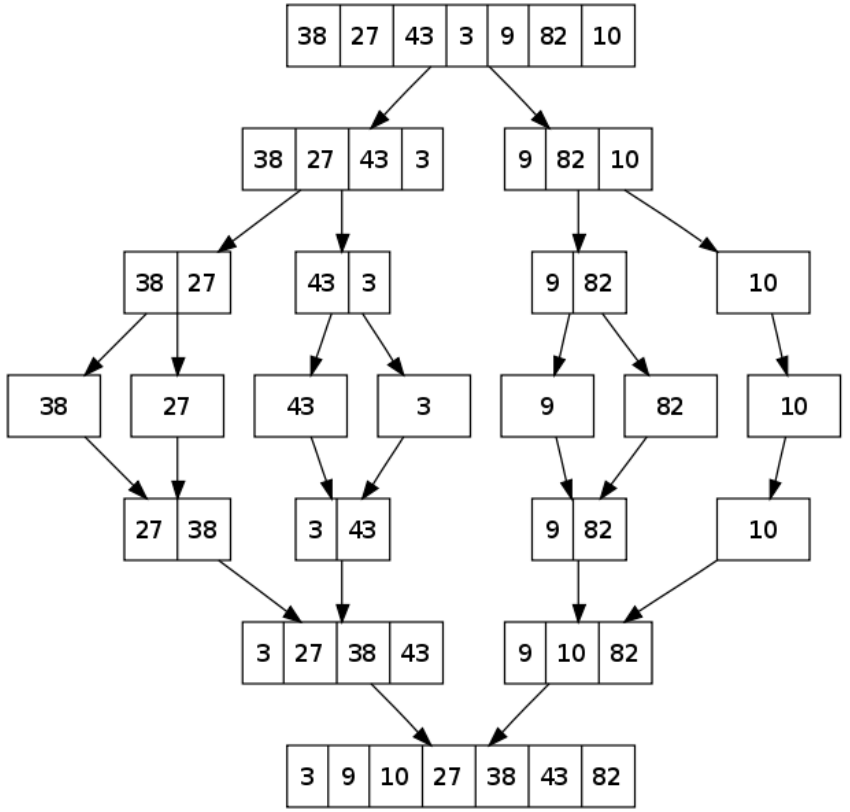


Merge Sort

```
void merge (int *v, int ini, int meio, int fim) {
        int *temp, p1, p2, tam, i,  j, k;
        int fim1 = 0, fim2 = 0;
        tam = fim – ini +1;
        p1 = ini;
        p2 = meio + 1;
        temp = (int *) malloc(tam * sizeof(int));
        if(temp != NULL){
                for(i = 0; i < tam; i++){
                        if(!fim1 && !fim2){
                                if(v[p1] < v[p2])
                                        temp[i] = v[p1++];
                                else
                                        temp[i] = v[p2++];

                                if(p1>meio) fim1 = 1;
                                if(p2>fim)    fim2 = 1;
                        }else{
                                if (!fim1)
                                        temp[i] = v[p1++];
                                else
                                        temp[i] = v[p2++];
                        }
                }
                for(j = 0, k = ini; j < tam; j++, k++)
                        v[k] = temp[j];
        }
        free(temp);
```

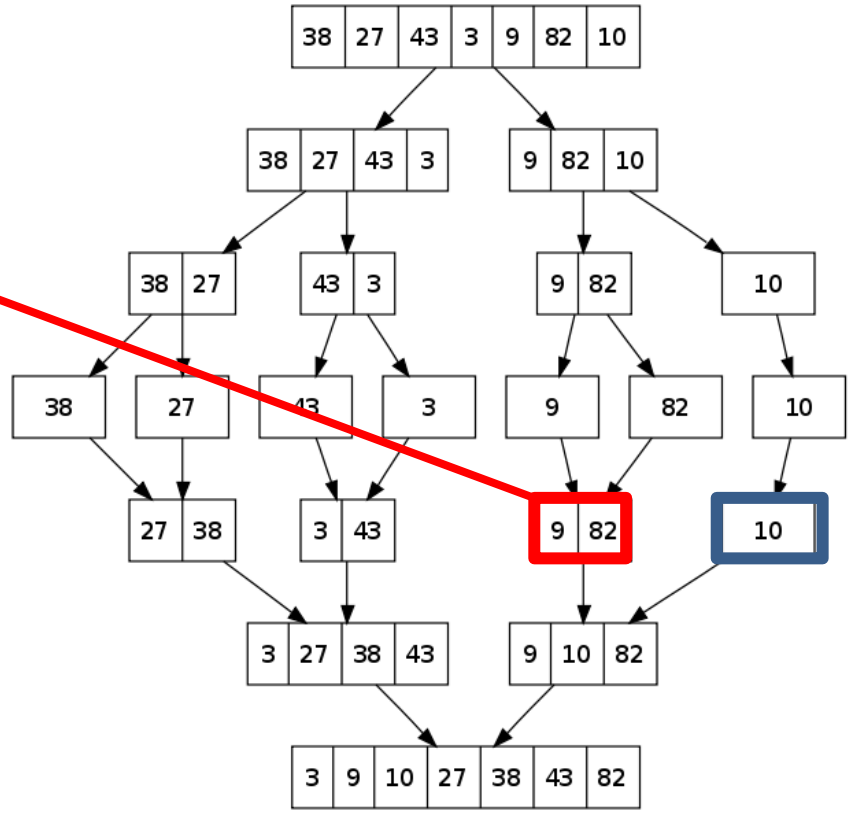
Merge Sort

```
void merge (int *v, int ini, int meio, int fim) {
        int *temp, p1, p2, tam, i,  j, k;
        int fim1 = 0, fim2 = 0;
        tam = fim – ini +1;
        p1 = ini;
        p2 = meio + 1;
        temp = (int *) malloc(tam * sizeof(int));
        if(temp != NULL){
                for(i = 0; i < tam; i++){
                        if(!fim1 && !fim2){
                                if(v[p1] < v[p2])
                                        temp[i] = v[p1++];
                                else
                                        temp[i] = v[p2++];

                                if(p1>meio) fim1 = 1;
                                if(p2>fim)    fim2 = 1;
                        }else{
                                if (!fim1)
                                        temp[i] = v[p1++];
                                else
                                        temp[i] = v[p2++];
                        }
                }
                for(j = 0, k = ini; j < tam; j++, k++)
                        v[k] = temp[j];
        }
        free(temp);
```



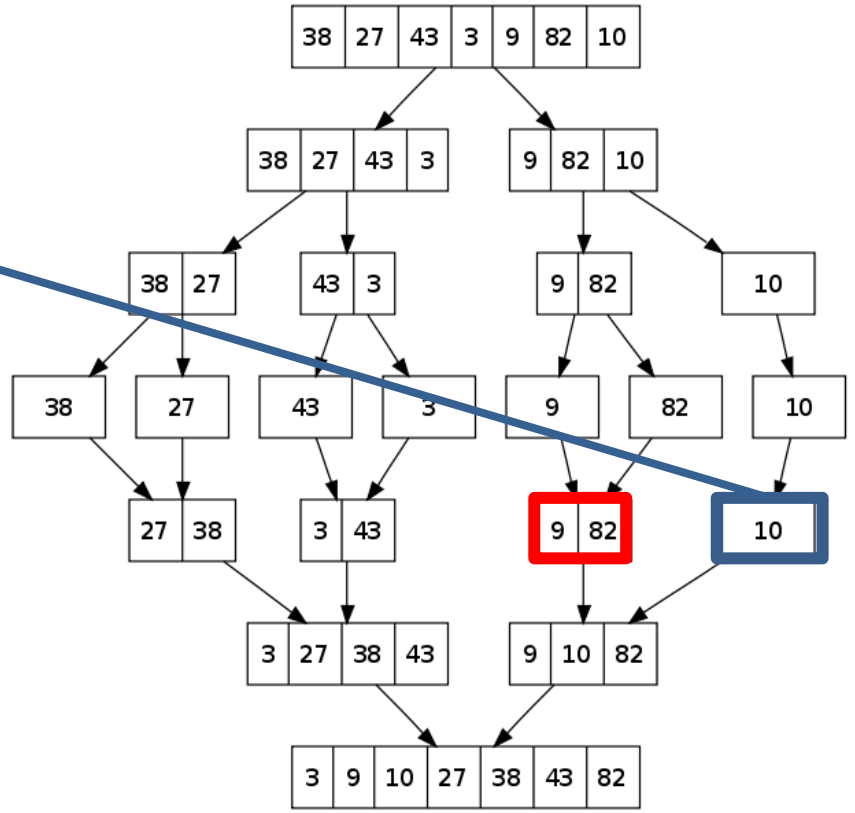
Merge Sort

```
void merge (int *v, int ini, int meio, int fim) {
        int *temp, p1, p2, tam, i,  j, k;
        int fim1 = 0, fim2 = 0;
        tam = fim – ini +1;
        p1 = ini;
        p2 = meio + 1;
        temp = (int *) malloc(tam * sizeof(int));
        if(temp != NULL){
                for(i = 0; i < tam; i++){
                        if(!fim1 && !fim2){
                                if(v[p1] < v[p2])
                                        temp[i] = v[p1++];
                                else
                                        temp[i] = v[p2++];

                                if(p1>meio) fim1 = 1;
                                if(p2>fim)    fim2 = 1;
                        }else{
                                if (!fim1)
                                        temp[i] = v[p1++];
                                else
                                        temp[i] = v[p2++];
                        }
                }
                for(j = 0, k = ini; j < tam; j++, k++)
                        v[k] = temp[j];
        }
        free(temp);
```



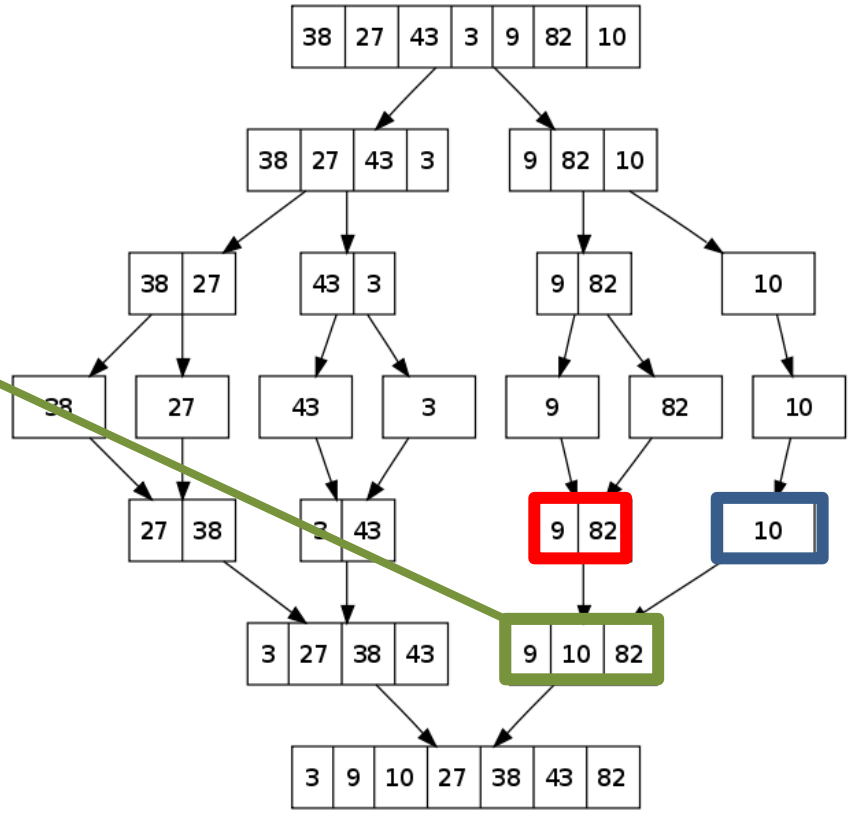
Merge Sort

```
void merge (int *v, int ini, int meio, int fim) {
        int *temp, p1, p2, tam, i,  j, k;
        int fim1 = 0, fim2 = 0;
        tam = fim – ini +1;
        p1 = ini;
        p2 = meio + 1;
        temp = (int *) malloc(tam * sizeof(int));
        if(temp != NULL){
                for(i = 0; i < tam; i++){
                        if(!fim1 && !fim2){
                                if(v[p1] < v[p2])
                                        temp[i] = v[p1++];
                                else
                                        temp[i] = v[p2++];

                                if(p1>meio) fim1 = 1;
                                if(p2>fim)    fim2 = 1;
                        }else{
                                if (!fim1)
                                        temp[i] = v[p1++];
                                else
                                        temp[i] = v[p2++];
                        }
                }
                for(j = 0, k = ini; j < tam; j++, k++)
                        v[k] = temp[j];
        }
        free(temp);
```



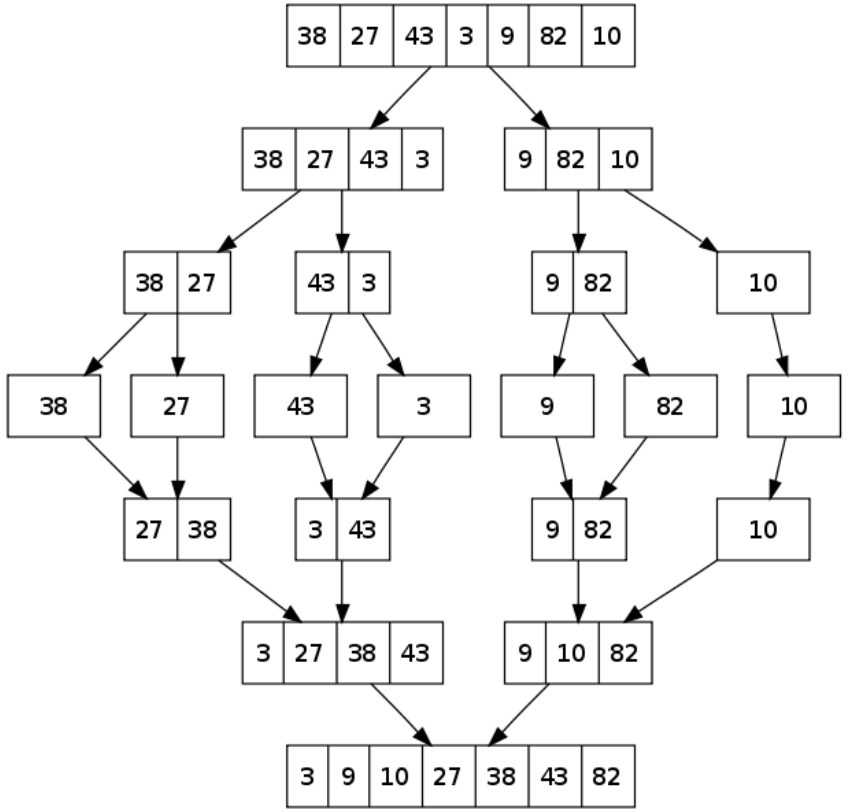
Merge Sort

```
void merge (int *v, int ini, int meio, int fim) {
        int *temp, p1, p2, tam, i,  j, k;
        int fim1 = 0, fim2 = 0;
        tam = fim – ini +1;
        p1 = ini;
        p2 = meio + 1;
        temp = (int *) malloc(tam * sizeof(int));
        if(temp != NULL){
                for(i = 0; i < tam; i++){
                        if(!fim1 && !fim2){
                                if(v[p1] < v[p2])
                                        temp[i] = v[p1++];
                                else
                                        temp[i] = v[p2++];

                                if(p1>meio) fim1 = 1;
                                if(p2>fim)    fim2 = 1;
                        }else{
                                if (!fim1)
                                        temp[i] = v[p1++];
                                else
                                        temp[i] = v[p2++];
                        }
                }
                for(j = 0, k = ini; j < tam; j++, k++)
                        v[k] = temp[j];
        }
        free(temp);
```



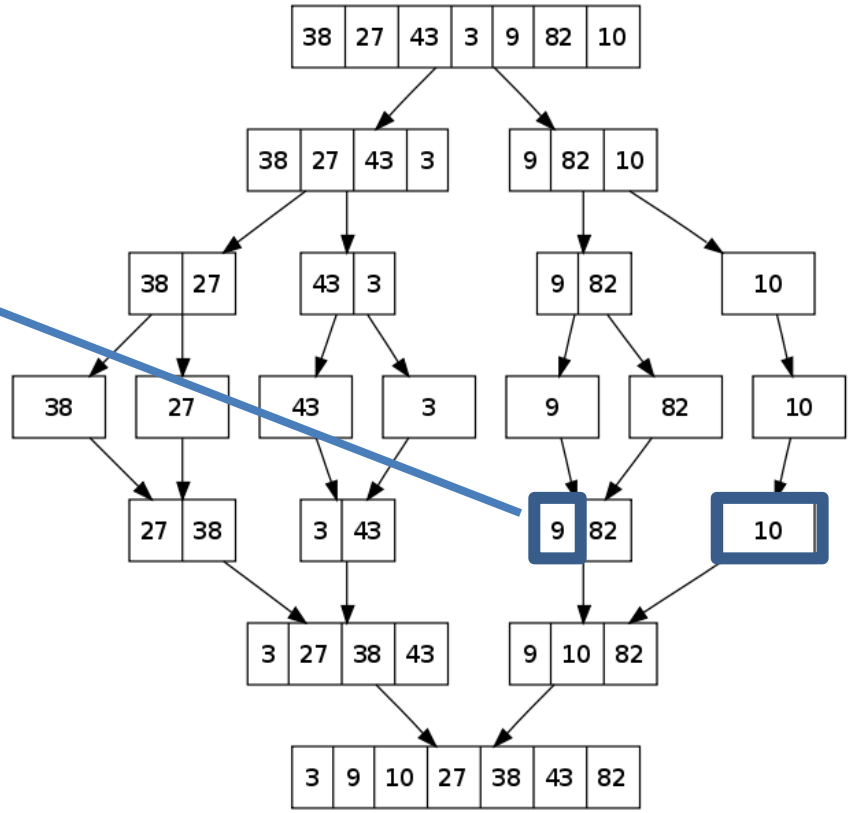
Merge Sort

```
void merge (int *v, int ini, int meio, int fim) {
        int *temp, p1, p2, tam, i,  j, k;
        int fim1 = 0, fim2 = 0;
        tam = fim – ini +1;
        p1 = ini;
        p2 = meio + 1;
        temp = (int *) malloc(tam * sizeof(int));
        if(temp != NULL){
                for(i = 0; i < tam; i++){
                        if(!fim1 && !fim2){
                                if(v[p1] < v[p2])
                                        temp[i] = v[p1++];
                                else
                                        temp[i] = v[p2++];

                                if(p1>meio) fim1 = 1;
                                if(p2>fim)    fim2 = 1;
                        }else{
                                if (!fim1)
                                        temp[i] = v[p1++];
                                else
                                        temp[i] = v[p2++];
                        }
                }
                for(j = 0, k = ini; j < tam; j++, k++)
                        v[k] = temp[j];
        }
        free(temp);
```



Merge Sort

```
void merge (int *v, int ini, int meio, int fim) {
        int *temp, p1, p2, tam, i,  j, k;
        int fim1 = 0, fim2 = 0;
        tam = fim – ini +1;
        p1 = ini;
        p2 = meio + 1;
        temp = (int *) malloc(tam * sizeof(int));
        if(temp != NULL){
                for(i = 0; i < tam; i++){
                        if(!fim1 && !fim2){
                                if(v[p1] < v[p2])
                                        temp[i] = v[p1++];
                                else
                                        temp[i] = v[p2++];

                                if(p1>meio) fim1 = 1;
                                if(p2>fim)    fim2 = 1;
                        }else{
                                if (!fim1)
                                        temp[i] = v[p1++];
                                else
                                        temp[i] = v[p2++];
                        }
                }
                for(j = 0, k = ini; j < tam; j++, k++)
                        v[k] = temp[j];
        }
        free(temp);
```



Merge Sort

```
void merge (int *v, int ini, int meio, int fim) {
        int *temp, p1, p2, tam, i,  j, k;
        int fim1 = 0, fim2 = 0;
        tam = fim – ini +1;
        p1 = ini;
        p2 = meio + 1;
        temp = (int *) malloc(tam * sizeof(int));
        if(temp != NULL){
                for(i = 0; i < tam; i++){
                        if(!fim1 && !fim2){
                                if(v[p1] < v[p2])
                                        temp[i] = v[p1++];
                                else
                                        temp[i] = v[p2++];

                                if(p1>meio) fim1 = 1;
                                if(p2>fim)    fim2 = 1;
                        }else{
                                if (!fim1)
                                        temp[i] = v[p1++];
                                else
                                        temp[i] = v[p2++];
                        }
                }
                for(j = 0, k = ini; j < tam; j++, k++)
                        v[k] = temp[j];
        }
        free(temp);
```

Combina ordenando

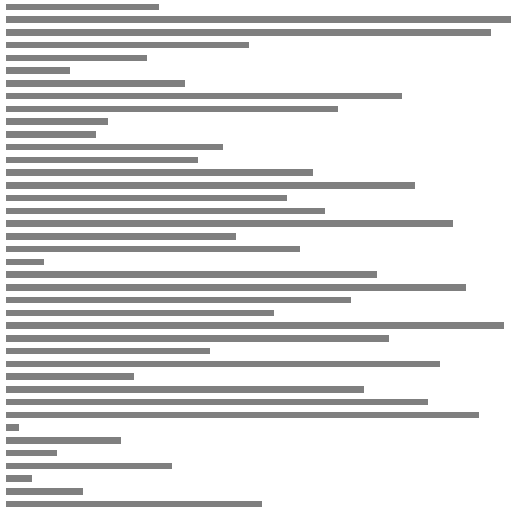
Vetor acabou?

Copia sobra

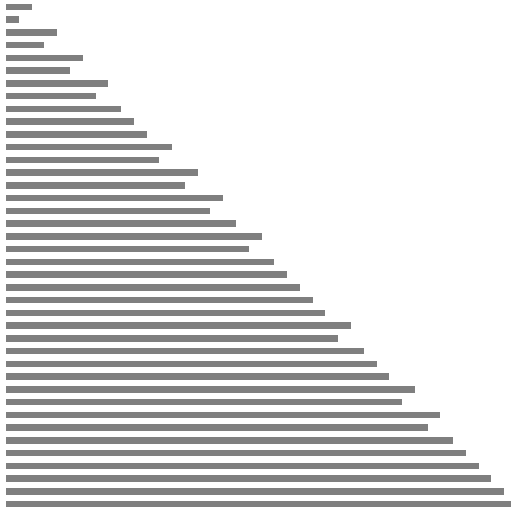
Copia do auxiliar para o original

Estudo de Casos

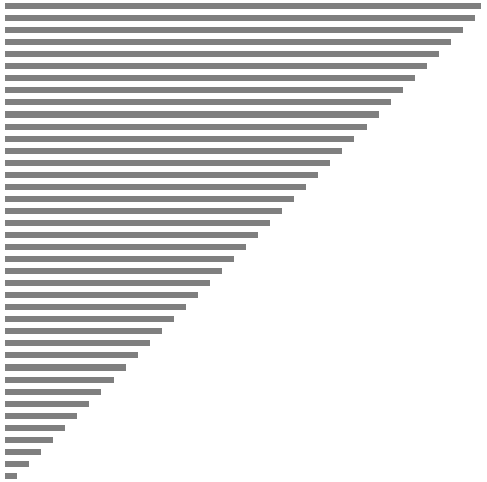
Aleatório



Quase ordenado



Inverso



Valores repetidos

Características

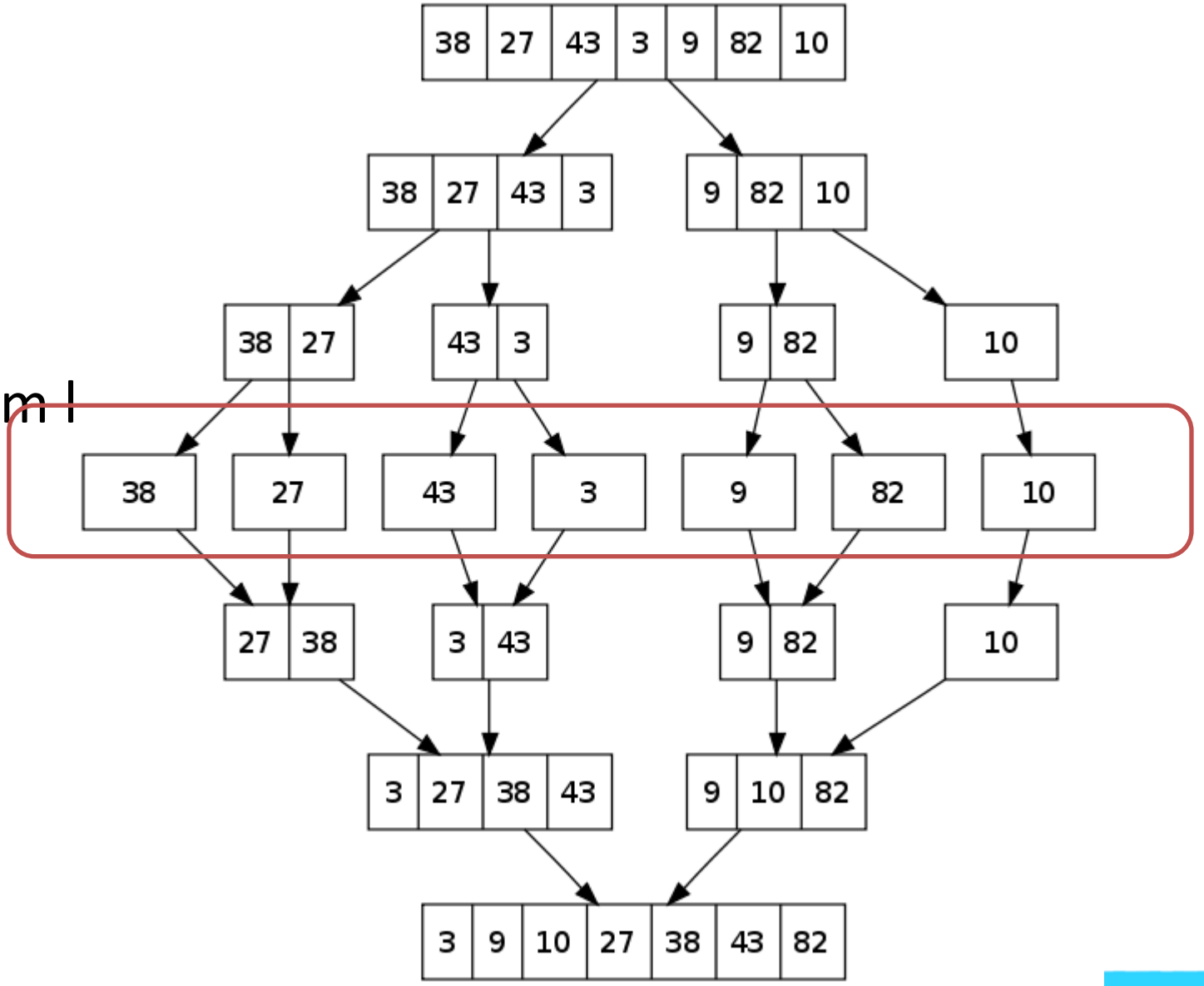
- Estável

Características

- Estável
- Memória
  - O(n) se implementado com listas ligadas

Características

- Estável
- Memória
  - O(n) se implementado com l



Características

- Estável
- Memória
  - O(n) se implementado com listas ligadas
- Computação
  - $O(n \cdot \log n)$
- Não é adaptativo
- Não precisa de acesso aleatório aos dados

Leitura para a prova

livro “Algorithms” de Cormen et al. sobre fundamentos.