Mergesort

Marcelo K. Albertini

28 de Novembro de 2013

Quicksort

- ordenação para tipos primitivos em java
- C qsort

Mergesort

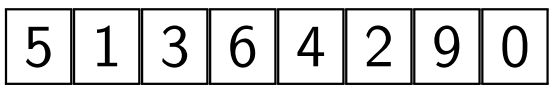
- ordenação de objetos em java
- ordenação estável em C++
- ordenação estavel em python
- autor: Von Neumann: EDVAC um dos primeiros computadores de propósito geral

Mergesort

Ideia

1. dividir vetor em 2 metades
2. recursivamente ordenar cada metade
3. mesclar – merge – as duas metades ordenadas

entrada



Mergesort

Ideia

1. dividir vetor em 2 metades
2. recursivamente ordenar cada metade
3. mesclar – merge – as duas metades ordenadas

entrada	5	1	3	6	4	2	9	0
ordena esquerda	1	3	5	5	4	2	9	0

Mergesort

Ideia

1. dividir vetor em 2 metades
2. recursivamente ordenar cada metade
3. mesclar – merge – as duas metades ordenadas

entrada	5	1	3	6	4	2	9	0
ordena esquerda	1	3	5	5	4	2	9	0
ordena direita	1	3	5	6	0	2	4	9

Mergesort

Ideia

1. dividir vetor em 2 metades
2. recursivamente ordenar cada metade
3. mesclar – merge – as duas metades ordenadas

entrada	5	1	3	6	4	2	9	0
ordena esquerda	1	3	5	5	4	2	9	0
ordena direita	1	3	5	6	0	2	4	9
antes do merge	1	3	5	6	0	2	4	9

Mergesort

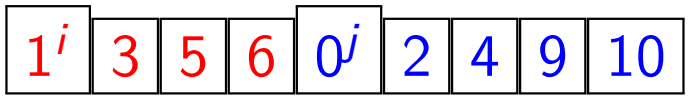
Ideia

1. dividir vetor em 2 metades
2. recursivamente ordenar cada metade
3. mesclar – merge – as duas metades ordenadas

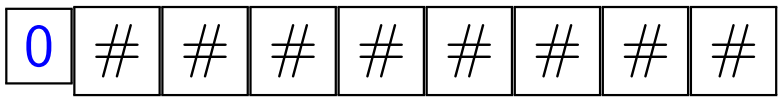
entrada	5	1	3	6	4	2	9	0
ordena esquerda	1	3	5	5	4	2	9	0
ordena direita	1	3	5	6	0	2	4	9
antes do merge	1	3	5	6	0	2	4	9
ordenado	0	1	2	3	4	5	6	9

Operação Merge

copiar menor valor do auxiliar

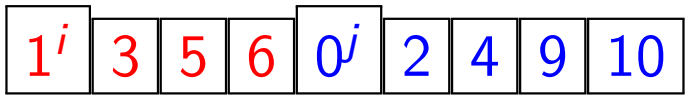


no vetor ordenado

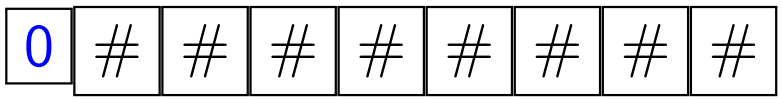


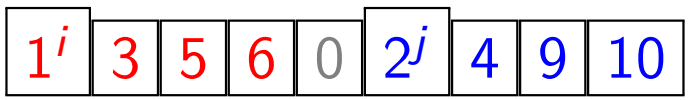
Operação Merge

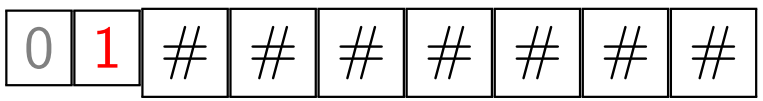
copiar menor valor do auxiliar



no vetor ordenado

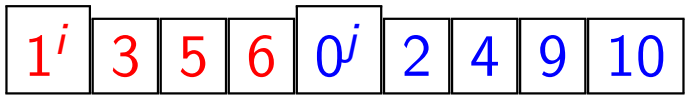




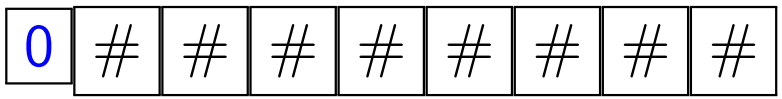


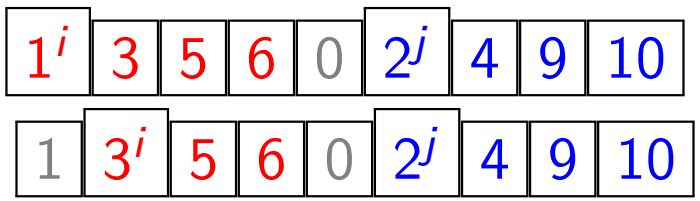
Operação Merge

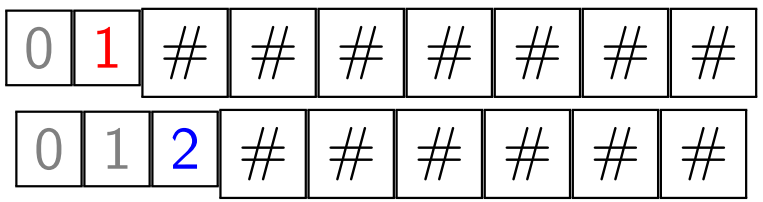
copiar menor valor do auxiliar



no vetor ordenado

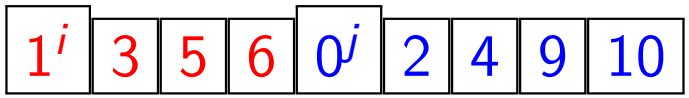




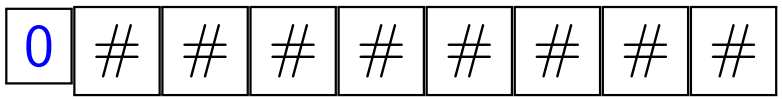


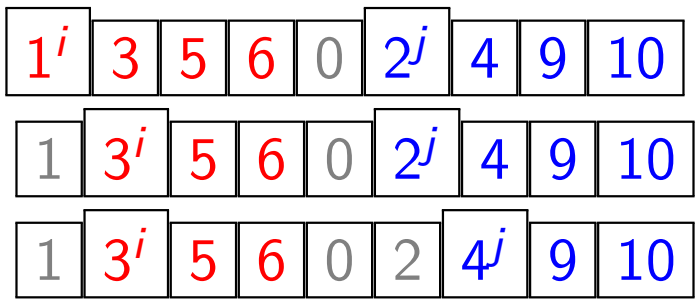
Operação Merge

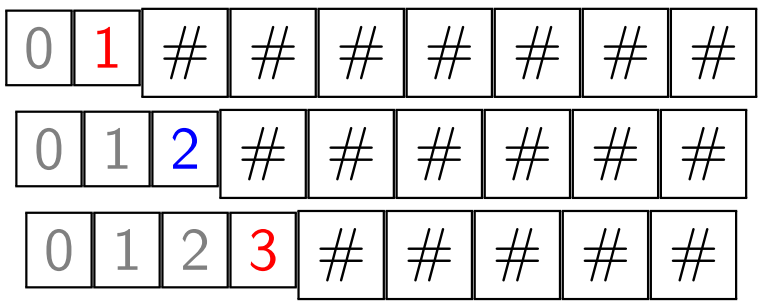
copiar menor valor do auxiliar



no vetor ordenado

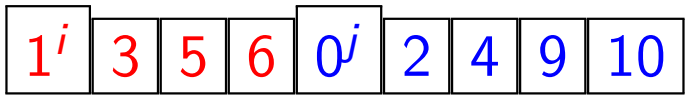


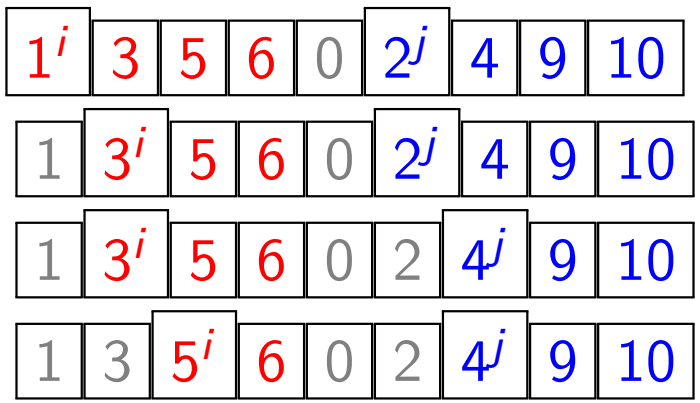




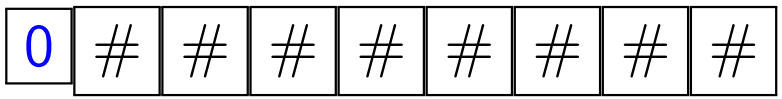
Operação Merge

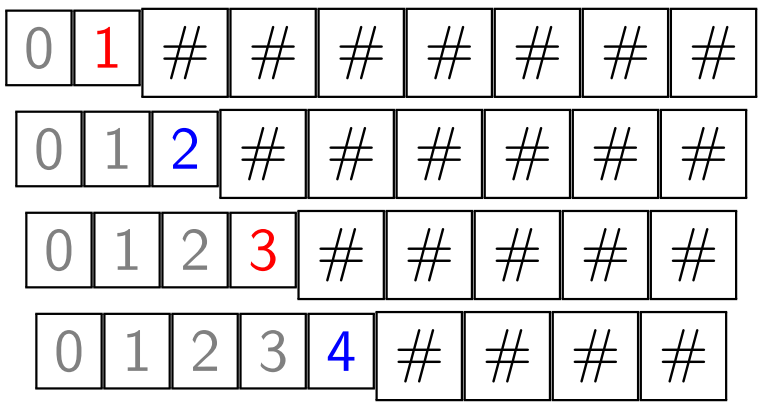
copiar menor valor do auxiliar





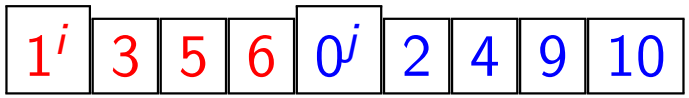
no vetor ordenado

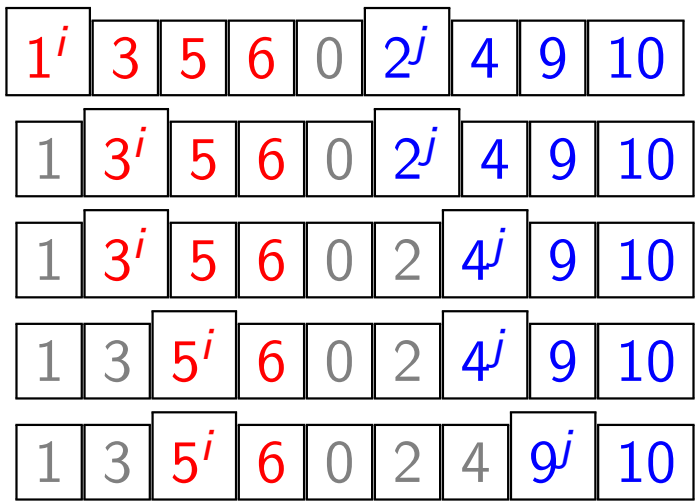




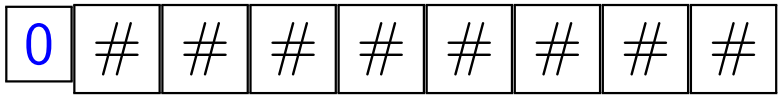
Operação Merge

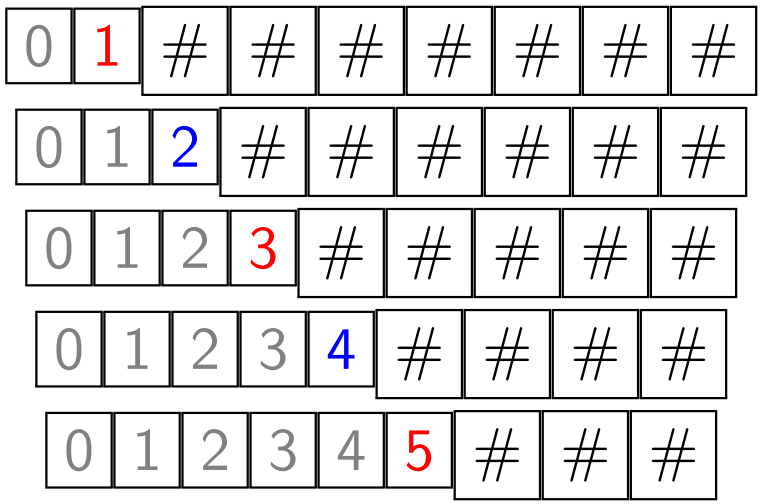
copiar menor valor do auxiliar



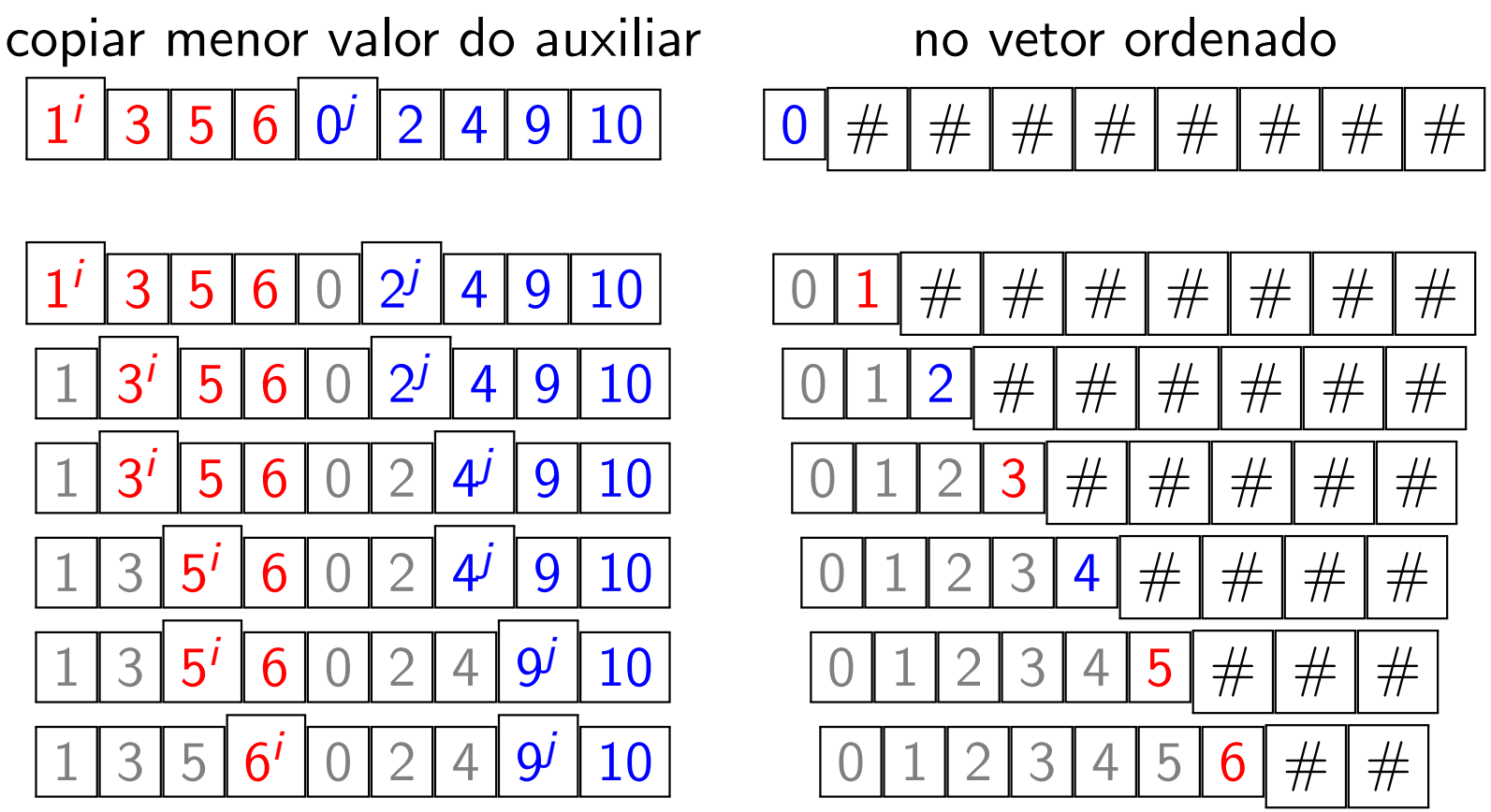


no vetor ordenado



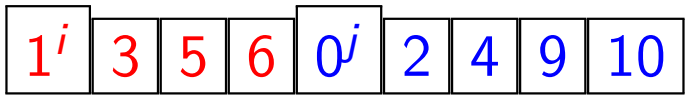


Operação Merge

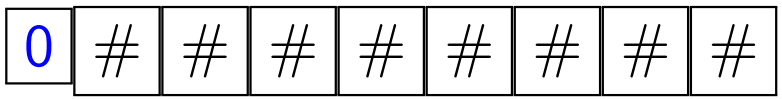


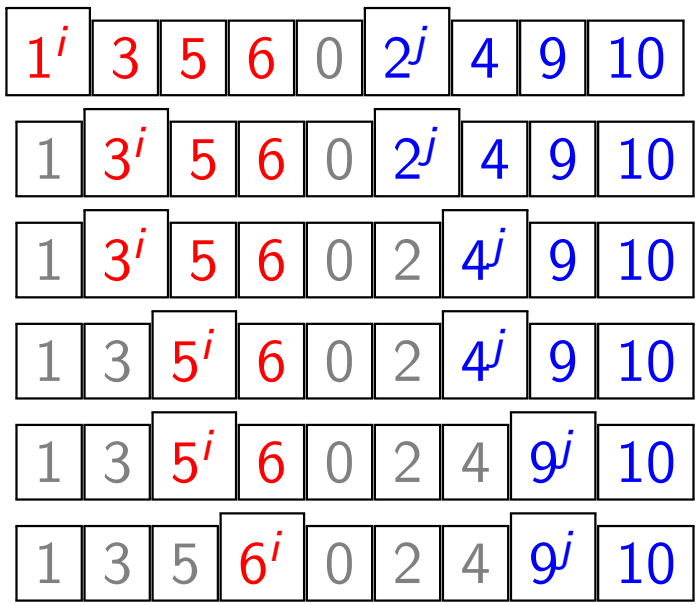
Operação Merge

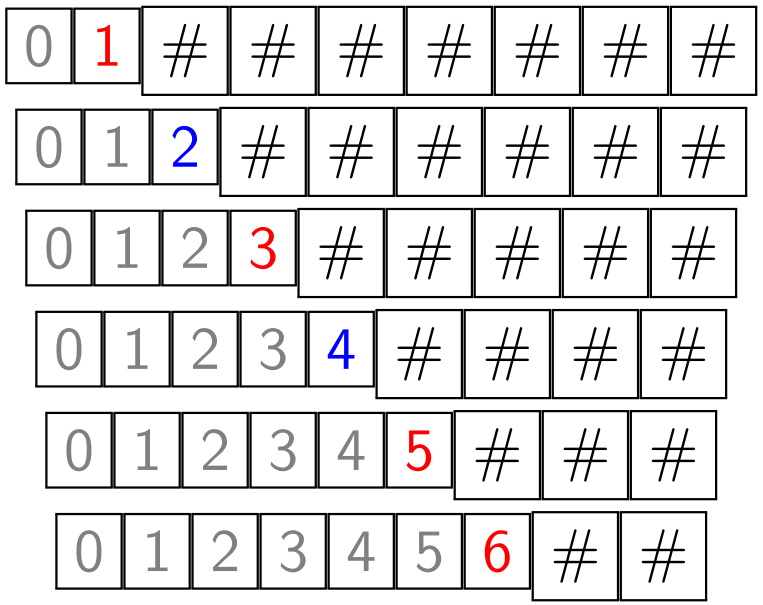
copiar menor valor do auxiliar



no vetor ordenado





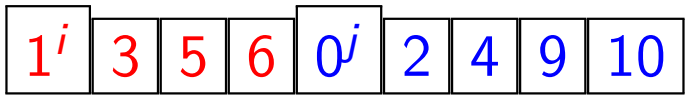


parte esquerda acabou

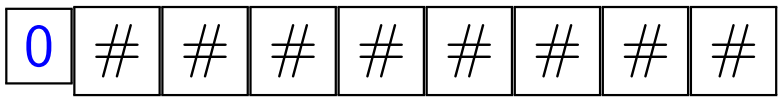
, copiar os valores restantes

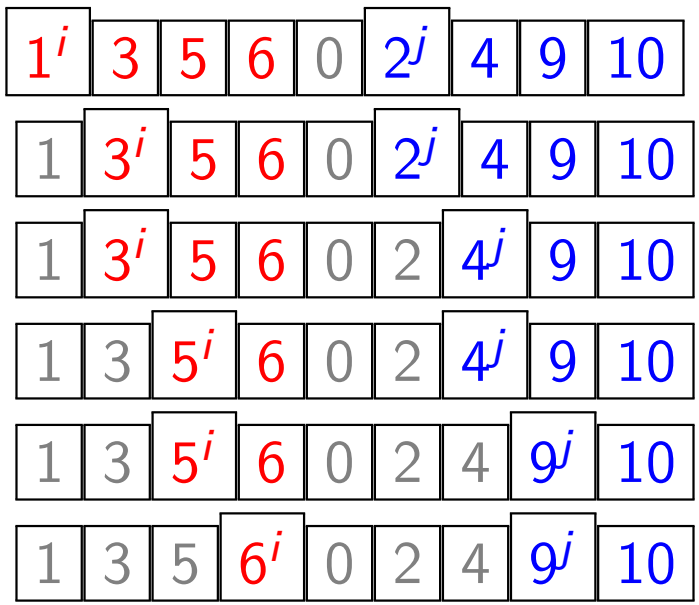
Operação Merge

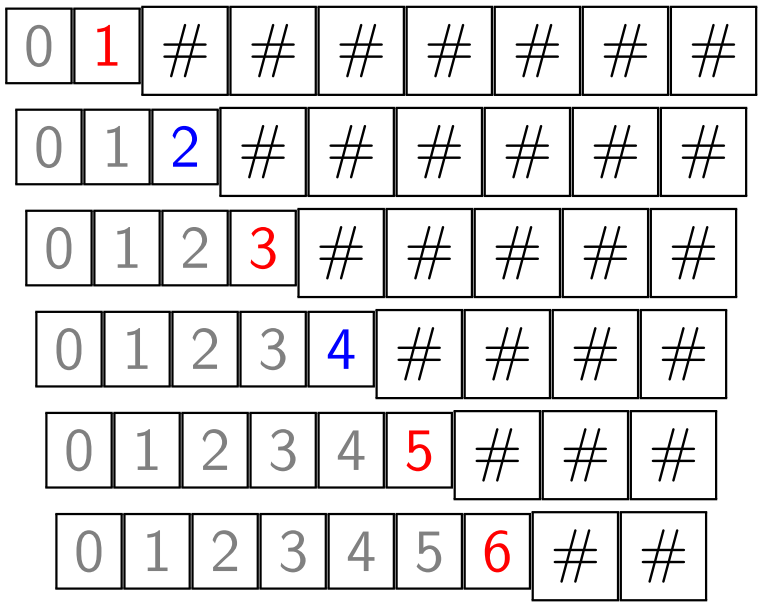
copiar menor valor do auxiliar



no vetor ordenado







parte esquerda acabou

, copiar os valores restantes

Implementação

```
1  // v: vetor sendo ordenado, aux: vetor auxiliar
2  // inf: posicao inferior sendo trabalhada
3  // med: posicao mediana, sup: posicao superior
4  merge(int[] v, int[] aux, int inf, int med, int sup) {
5    for (int k = inf; k <= sup; k++)
6      aux[k] = v[k]; // copia dos valores para auxiliar
7
8    int i = inf, j = med+1;
9    for (int k = inf; k <= sup; k++) {
10     // se subvetor da direita terminou
11     if       (i > med)      v[k] = aux[j++];
12     // se subvetor da esquerda terminou
13     else if (j > sup)       v[k] = aux[i++];
14     else // senão, compara e copia o menor valor
15     if (aux[j] < aux[i])   v[k] = aux[j++];
16     else                   v[k] = aux[i++];
17   }
18 }
```

Mergesort

```
1  mergesort(int[] v) {
2      int aux[] = new int[v.length];
3      sort(v, aux, 0, v.length-1);
4  }
5
6  sort(int[] v, int[] aux, int inf, int sup) {
7    if (sup <= inf) return;
8    int med = inf + (sup - inf)/2;
9
10   sort(v, aux, inf, med);
11   sort(v, aux, med+1, sup);
12   merge(v, aux, inf, med, sup);
13 }
```

Mergesort recursivo

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

Mergesort recursivo

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5

Mergesort recursivo

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5

Mergesort recursivo

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5

Mergesort recursivo

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5

Mergesort recursivo

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5

Mergesort recursivo

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5

Mergesort recursivo

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5

Mergesort recursivo

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5

Mergesort recursivo

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	4	7	1	3	5

Mergesort recursivo

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	4	7	1	3	5

0	2	6	8	9	4	7	1	3	5

Mergesort recursivo

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	4	7	1	3	5

0	2	6	8	9	4	7	1	3	5
0	2	6	8	9	1	4	7	3	5

Mergesort recursivo

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	4	7	1	3	5

0	2	6	8	9	4	7	1	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5

Mergesort recursivo

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	4	7	1	3	5

0	2	6	8	9	4	7	1	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5

Mergesort recursivo

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	4	7	1	3	5

0	2	6	8	9	4	7	1	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5

Mergesort recursivo

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	4	7	1	3	5

0	2	6	8	9	4	7	1	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	3	4	5	7

Mergesort recursivo

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	4	7	1	3	5

0	2	6	8	9	4	7	1	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	3	4	5	7
0	2	6	8	9	1	3	4	5	7

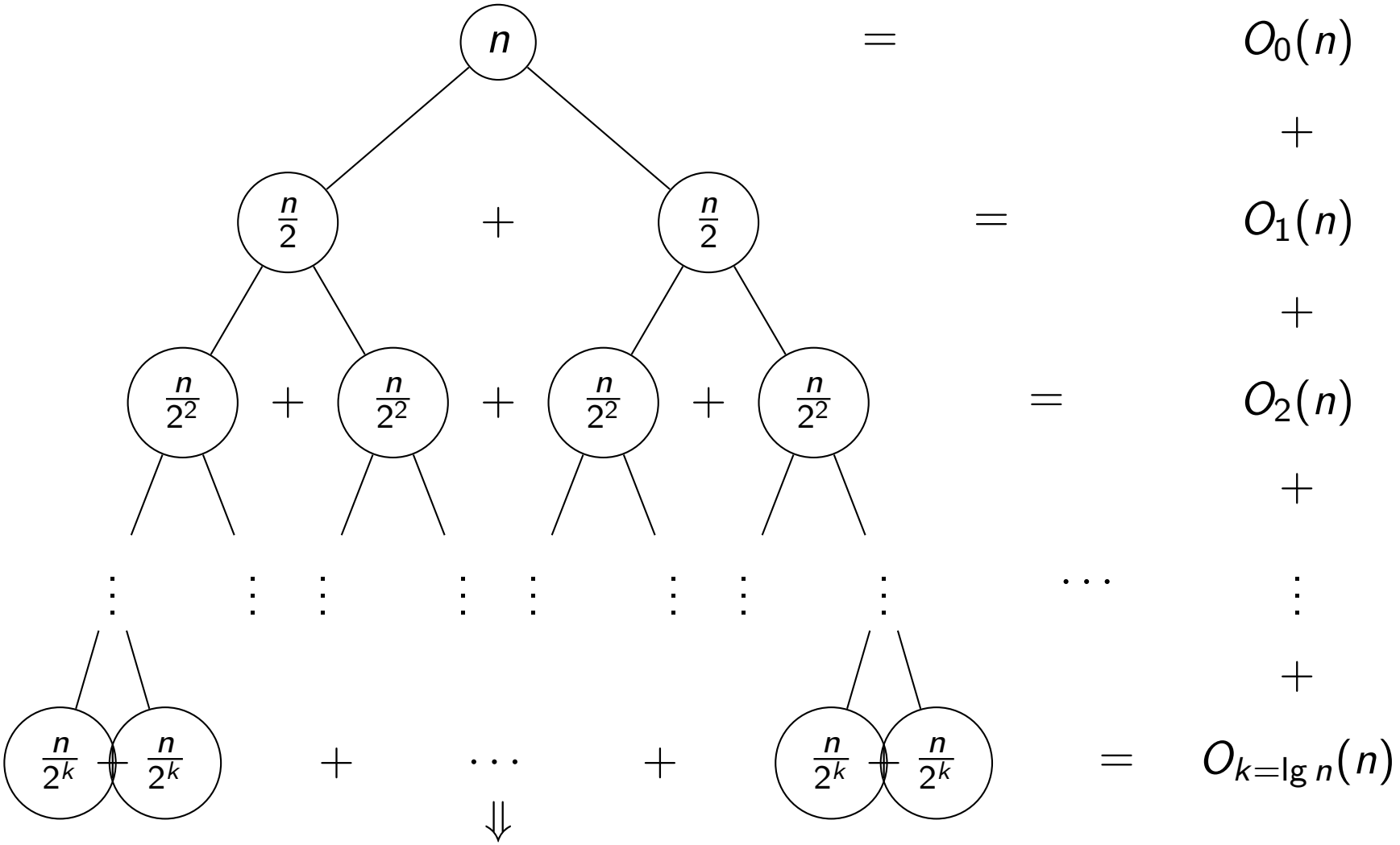
Mergesort recursivo

Em cinza: posições desconsideradas do `merge` atual.
Em vermelho: subvetor com posições a partir de `inf` até `med`.
Em azul: subvetor com posições depois de `med` até `sup`.
Em verde: subvetor resultante do `merge`.

6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
6	8	2	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	9	0	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
2	6	8	0	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	7	4	1	3	5
0	2	6	8	9	4	7	1	3	5

0	2	6	8	9	4	7	1	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	4	7	3	5
0	2	6	8	9	1	3	4	5	7
0	2	6	8	9	1	3	4	5	7
0	1	2	3	4	5	6	7	8	9

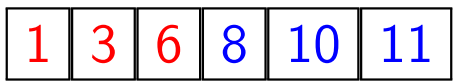
Análise de complexidade



$$O\left(\sum_{i=0}^{k} 2^i \cdot \frac{n}{2^i}\right) = O\left(\sum_{i=0}^{k} n\right) = O(k \cdot n) \Leftrightarrow O(n \cdot \lg n)$$

Mergesort - bottom up

Melhorias possíveis

- não fazer merge de dois subvetores já ordenados
  - quando o maior elemento do subvetor nas menores posições for menor que o menor elemento do subvetor nas maiores posições
  - exemplo: 

- eliminar recursão: algoritmo bottom-up
  - começar desde o ínicio com subvetores menores
  - aplicar merge para aumentar subvetores

Mergesort bottom-up

Ideia

- Varrer vetor fazendo merge de subvetores de tamanho 1
- Repetir operação para subvetores de tamanho 2, 4, 8, 16, ...

Mergesort bottom-up

```
1  mergesort(int[] v) {
2   int n = v.length;
3   int[] aux = new int[n];
4   // tamanho dobra a cada iteração
5   for (int tam = 1; tam < n; tam = tam +tam) {
6    for (int inf = 0; inf < n -tam; inf += tam+tam) {
7      // subvetor à esquerda em [inf, inf+tam-1]
8      // subvetor à direita em [inf+tam, inf+tam+tam-1]
9      // ou, se necessário, em [inf+tam, n-1]
10     merge(v, inf, inf+tam-1,
11           Math.min(inf+tam+tam-1, n-1));
12   }
13  }
14 }
```

Limites

$v[inf]$	$v[\ldots]$	$v[inf + tam - 1]$	$v[inf + tam]$	$v[\ldots]$	$v[inf + tam + tam - 1]$

ou

$v[inf]$	$v[\ldots]$	$v[inf + tam - 1]$	$v[inf + tam]$	$v[\ldots]$	$v[n - 1]$

Mergesort bottom-up

Mergesort bottom-up

5	4	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1

Mergesort bottom-up

5	4	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1

Mergesort bottom-up

5	4	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1

Mergesort bottom-up

5	4	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1

Mergesort bottom-up

5	4	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	0	6	3	8	7	1

Mergesort bottom-up

5	4	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1

Mergesort bottom-up

5	4	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1

Mergesort bottom-up

5	4	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1

Mergesort bottom-up

5	4	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	1	7

Mergesort bottom-up

5	4	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	1	7

4	5	2	9	0	6	3	8	1	7

Mergesort bottom-up

5	4	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	1	7

4	5	2	9	0	6	3	8	1	7
2	4	5	9	0	6	3	8	1	7

Mergesort bottom-up

5	4	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	1	7

4	5	2	9	0	6	3	8	1	7
2	4	5	9	0	6	3	8	1	7
2	4	5	9	0	6	3	8	1	7

Mergesort bottom-up

5	4	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	1	7

4	5	2	9	0	6	3	8	1	7
2	4	5	9	0	6	3	8	1	7
2	4	5	9	0	6	3	8	1	7
2	4	5	9	0	3	6	8	1	7

Mergesort bottom-up

5	4	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	1	7

4	5	2	9	0	6	3	8	1	7
2	4	5	9	0	6	3	8	1	7
2	4	5	9	0	6	3	8	1	7
2	4	5	9	0	3	6	8	1	7
2	4	5	9	0	3	6	8	1	7

Mergesort bottom-up

5	4	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	1	7

4	5	2	9	0	6	3	8	1	7
2	4	5	9	0	6	3	8	1	7
2	4	5	9	0	6	3	8	1	7
2	4	5	9	0	3	6	8	1	7
2	4	5	9	0	3	6	8	1	7
0	2	3	4	5	6	8	9	1	7

Mergesort bottom-up

5	4	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	1	7

4	5	2	9	0	6	3	8	1	7
2	4	5	9	0	6	3	8	1	7
2	4	5	9	0	6	3	8	1	7
2	4	5	9	0	3	6	8	1	7
2	4	5	9	0	3	6	8	1	7
0	2	3	4	5	6	8	9	1	7
0	2	3	4	5	6	8	9	1	7

Mergesort bottom-up

5	4	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	6	0	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	7	1
4	5	2	9	0	6	3	8	1	7

4	5	2	9	0	6	3	8	1	7
2	4	5	9	0	6	3	8	1	7
2	4	5	9	0	6	3	8	1	7
2	4	5	9	0	3	6	8	1	7
2	4	5	9	0	3	6	8	1	7
0	2	3	4	5	6	8	9	1	7
0	2	3	4	5	6	8	9	1	7
0	1	2	3	4	5	6	7	8	9

Complexidades

Melhor caso

Podemos representar qualquer algoritmo de ordenação baseado em comparações em uma árvore de comparações contém $n!$ elementos. Menor altura, que é proporcional ao número de comparações, dessa árvore é $\log(n!)$.

Usando a fórmula de Stirling

$\log(n!) \approx n \log(n)$

Algoritmo ótimo

Como o pior caso do mergesort é $O(n \log n)$ é igual ao melhor caso do problema, o mergesort é um algoritmo ótimo em relação ao número de comparações.

Estabilidade

Cenário

Se ordenarmos pela nome e depois pela idade, a ordem relativa pelo irá se manter? Se sim, algoritmo é **estável**.

nome	idade	cidade
aline	22	ituiutaba
joão	23	uberlândia
marcos	26	araguari
maria	22	uberaba

nome	idade	cidade
maria	22	uberaba
aline	22	ituiutaba
joão	23	uberlândia
marcos	26	araguari

Exemplo de algoritmo de ordenação NÃO estável.

Quais algoritmos vistos em aula são estáveis?

Bubblesort, Selectionsort, Insertionsort, Shellsort,
Quicksort simplificado, Quicksort em memória,
Quicksort de Dijkstra, Mergesort recursivo, Mergesort bottom-up