AULA 22

Intercalação

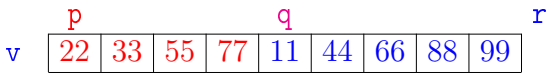
PF 9
http://www.ime.usp.br/~pf/algoritmos/aulas/mrgsrt.html

Intercalação

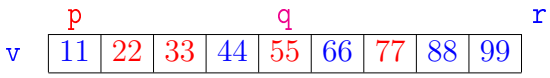
Problema: Dados $v[p\,..\,q-1]$ e $v[q\,..\,r-1]$ crescentes, rearranjar $v[p\,..\,r-1]$ de modo que ele fique em ordem crescente.

Para que valores de q o problema faz sentido?

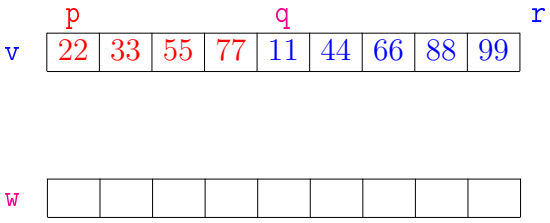
Entra:



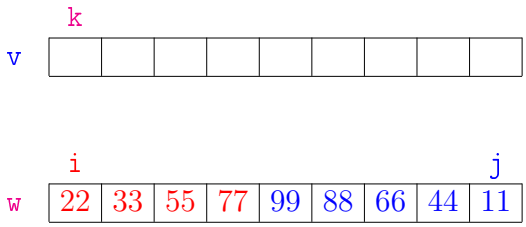
Sai:



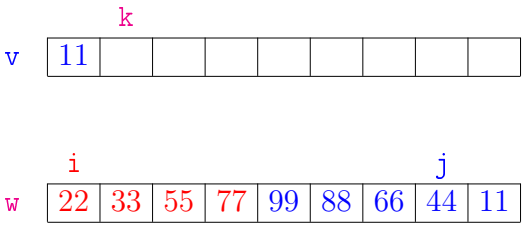
Intercalação



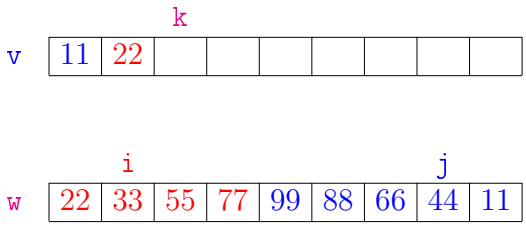
Intercalação



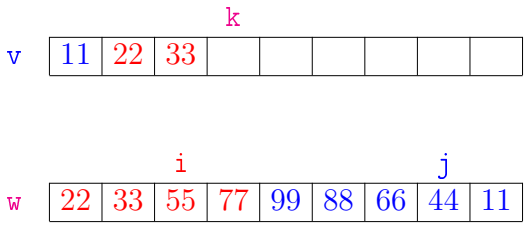
Intercalação



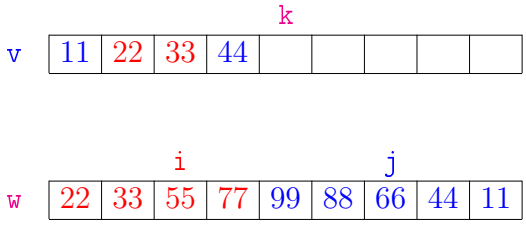
Intercalação



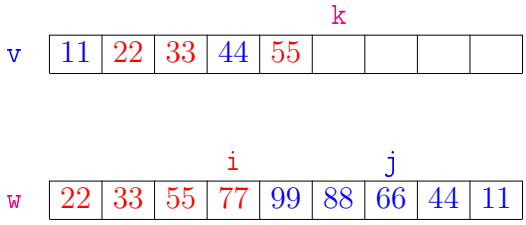
Intercalação



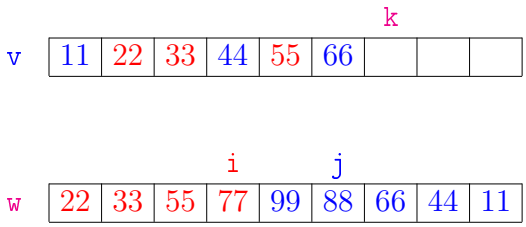
Intercalação



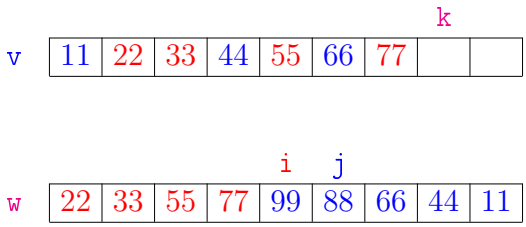
Intercalação



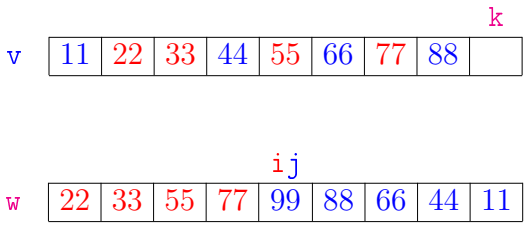
Intercalação



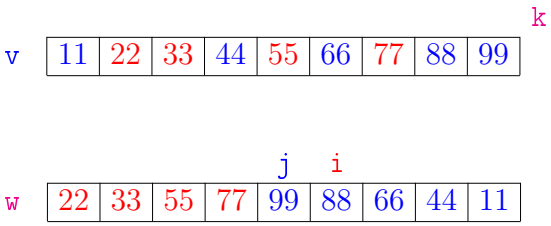
Intercalação



Intercalação



Intercalação



Intercalação

```
void intercala(int p,int q,int r,int v[]){
  int i, j, k, *w;
  w = mallocSafe((r-p)*sizeof(int));
1  for (i = 0, k = p; k < q; i++, k++)
2      w[i] = v[k];
3  for (j = r-p-1; k < r; j--, k++)
4      w[j] = v[k];
5  i = 0; j = r-p-1;
6  for (k = p; k < r; k++)
7      if (w[i] <= w[j])
8          v[k] = w[i++];
9      else v[k] = w[j--];
   free (w);
}
```

Consumo de tempo

Se a execução de cada linha de código consome 1 unidade de tempo o consumo total é:

linha	execuções da linha
1	?
2	?
3	?
4	?
5	?
6	?
7–9	?
total	?

Consumo de tempo

Se a execução de cada linha de código consome 1 unidade de tempo o consumo total é ($n := r - p$):

linha	execuções da linha
1	$= q - p = n - r + q$
2	$= q - p - 1 = n - r + q - 1$
3	$= r - q = n - q + p$
4	$= r - q - 1 = n - q + p - 1$
5	$= 1$
6	$= r - p + 1 = n + 1$
7–9	$= 2\,(r - p) = 2n$
total	$= 7n - 2(r - p) = 5n$

Conclusão

A função `intercala` consome $5n$ unidades de tempo.

O algoritmo `intercala` consome $O(n)$ unidades de tempo.

Também escreve-se

O algoritmo `intercala` consome tempo $O(n)$.

Ordenação: algoritmo Mergesort

PF 9
http://www.ime.usp.br/~pf/algoritmos/aulas/mrgsrt.html

Ordenação

v[0..n−1] é crescente se $v[0] \le \cdots \le v[n-1]$.

Problema: Rearranjar um vetor v[0..n−1] de modo que ele fique crescente.

Entra:

0										n−1
33	55	33	44	33	22	11	99	22	55	77

Sai:

0										n−1
11	22	22	33	33	33	44	55	55	77	99

Função mergeSort

Rearranja v[p..r−1] em ordem crescente.

```
void mergeSort (int p, int r, int v[]) {
1   if (p < r-1) {
2       int q = (p + r)/2;
3       mergeSort(p, q, v);
4       mergeSort(q, r, v);
5       intercala(p, q, r, v);
    }
}
```

	p			q					r
v	55	33	66	44	99	11	77	22	88

função mergeSort

Rearranja v[p..r−1] em ordem crescente.

```
void mergeSort (int p, int r, int v[]) {
1   if (p < r-1) {
2       int q = (p + r)/2;
3       mergeSort(p, q, v);
4       mergeSort(q, r, v);
5       intercala(p, q, r, v);
    }
}
```

	p			q					r
v	33	44	55	66	99	11	77	22	88

função mergeSort

Rearranja v[p..r−1] em ordem crescente.

```
void mergeSort (int p, int r, int v[]) {
1   if (p < r-1) {
2       int q = (p + r)/2;
3       mergeSort(p, q, v);
4       mergeSort(q, r, v);
5       intercala(p, q, r, v);
    }
}
```

	p				q				r
v	33	44	55	66	11	22	77	88	99

função mergeSort

Rearranja v[p..r−1] em ordem crescente.

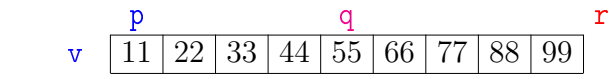
```
void mergeSort (int p, int r, int v[]) {
1   if (p < r-1) {
2       int q = (p + r)/2;
3       mergeSort(p, q, v);
4       mergeSort(q, r, v);
5       intercala(p, q, r, v);
    }
}
```

	p				q				r
v	11	22	33	44	55	66	77	88	99

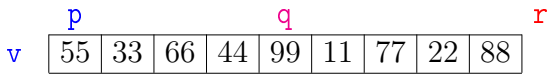
função mergeSort

Rearranja v[p..r−1] em ordem crescente.

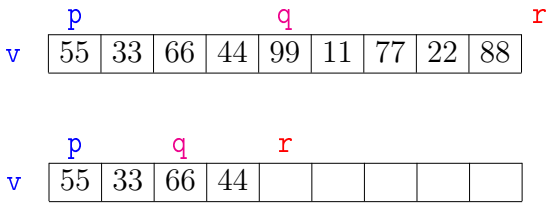
```
void mergeSort (int p, int r, int v[]) {
1   if (p < r-1) {
2       int q = (p + r)/2;
3       mergeSort(p, q, v);
4       mergeSort(q, r, v);
5       intercala(p, q, r, v);
    }
}
```



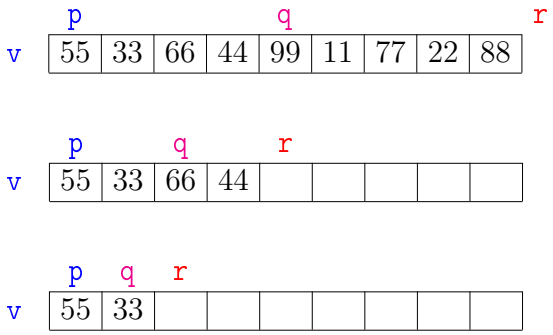
Mergesort



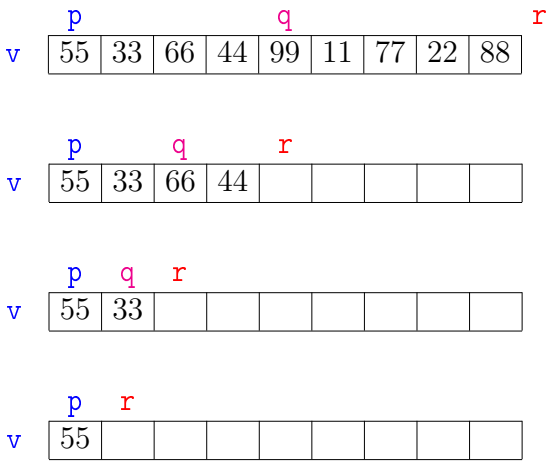
Mergesort



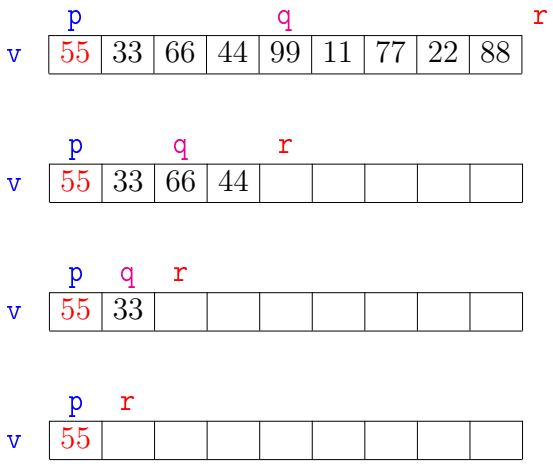
Mergesort



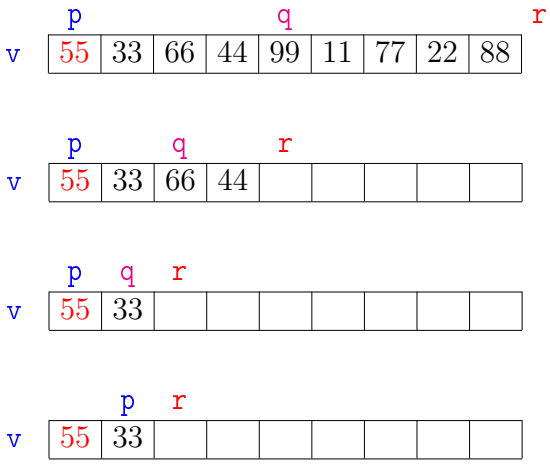
Mergesort



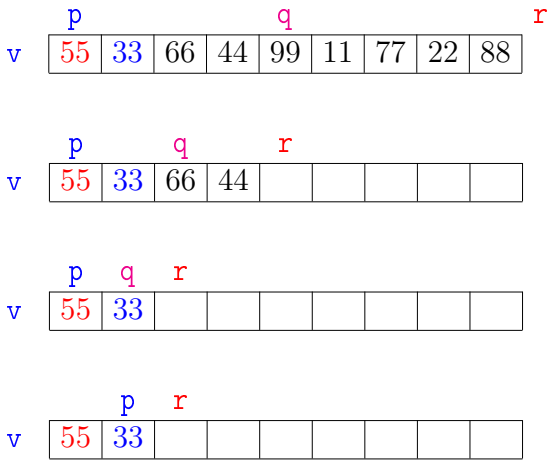
Mergesort



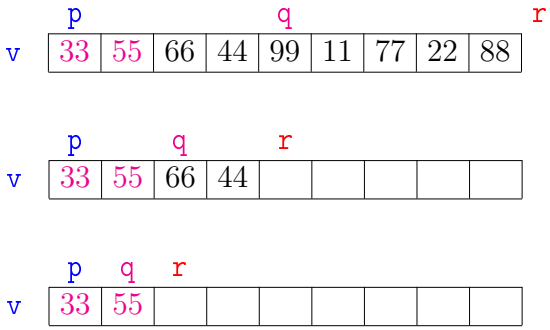
Mergesort



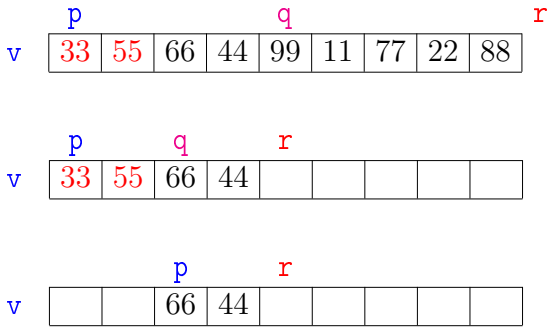
Mergesort



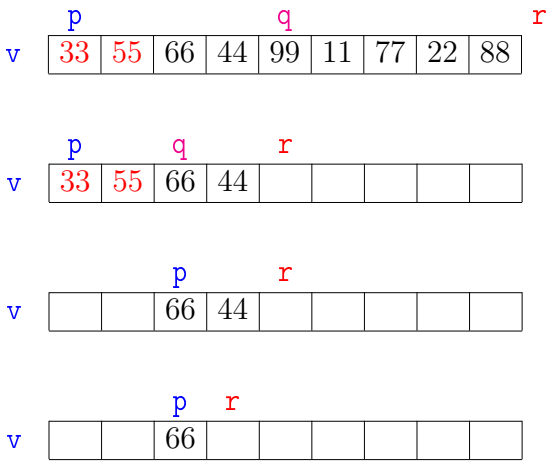
Mergesort



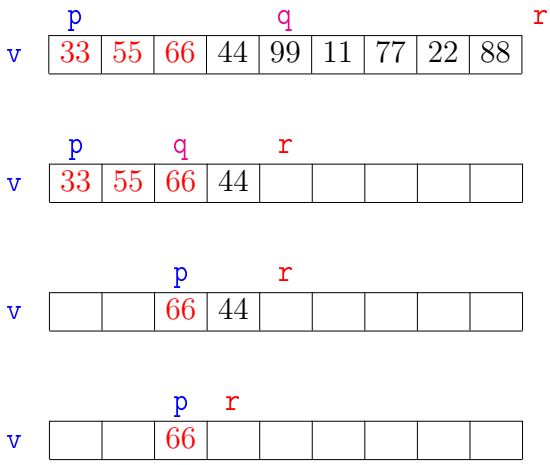
Mergesort



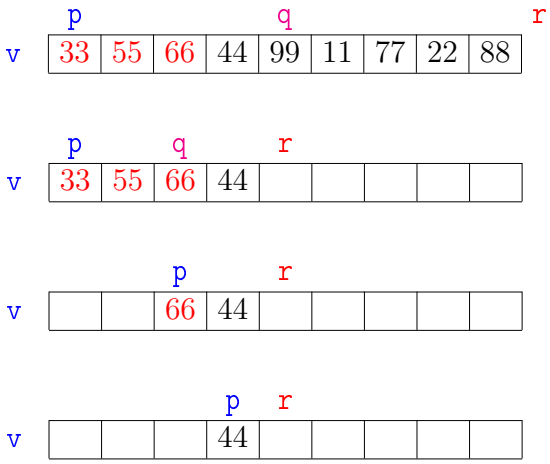
Mergesort



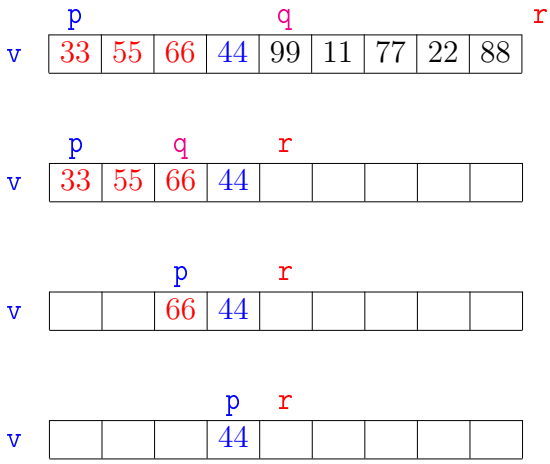
Mergesort



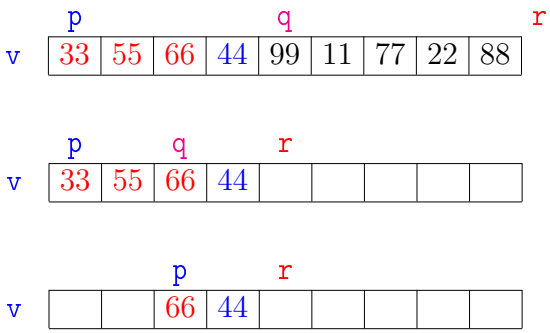
Mergesort



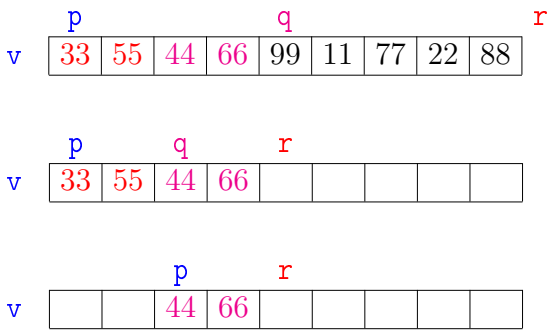
Mergesort



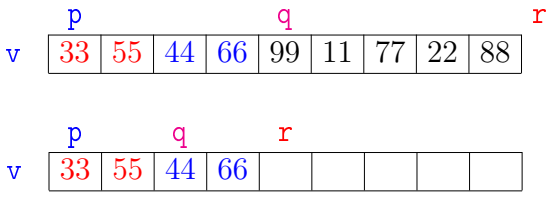
Mergesort



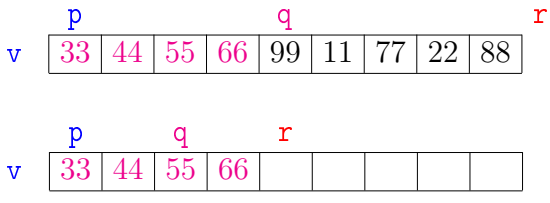
Mergesort



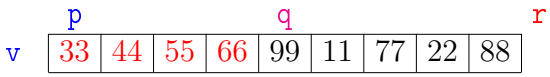
Mergesort



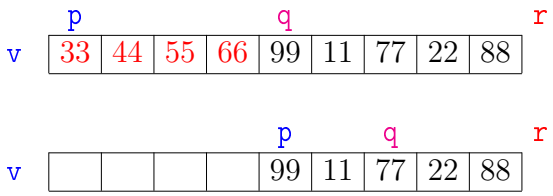
Mergesort



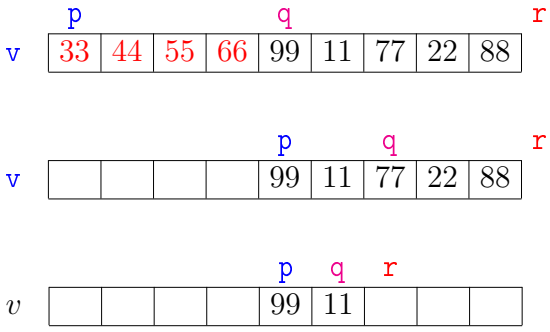
Mergesort



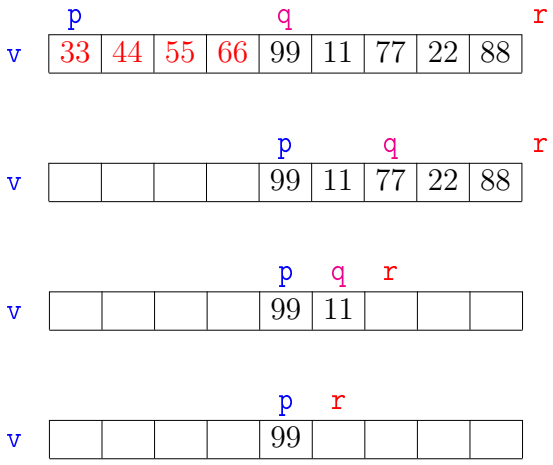
Mergesort



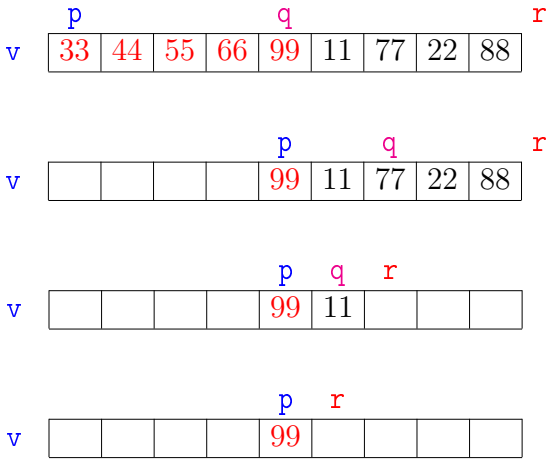
Mergesort



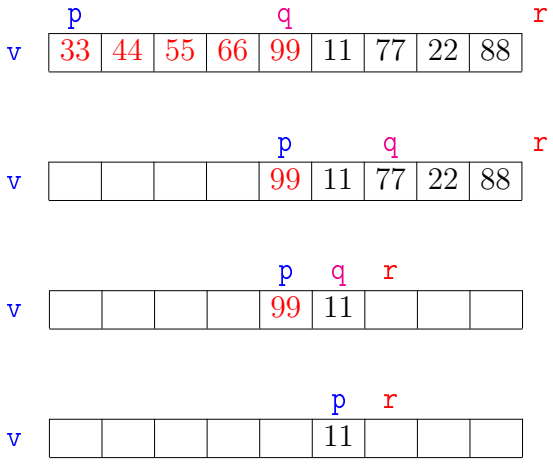
Mergesort



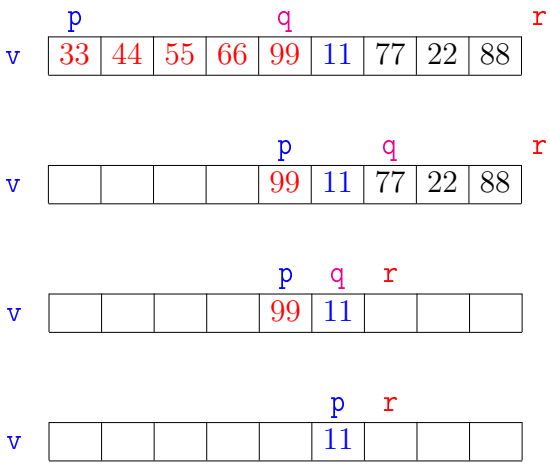
Mergesort



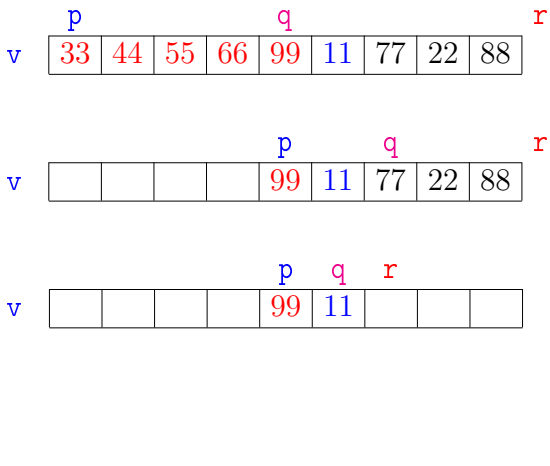
Mergesort



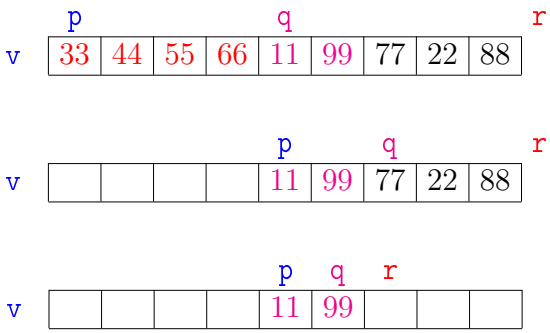
Mergesort



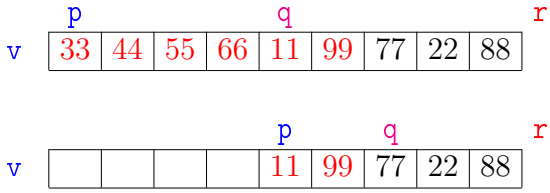
Mergesort



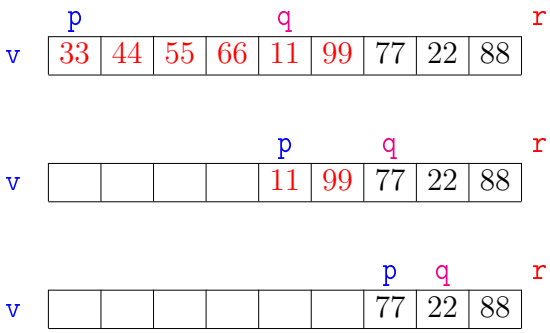
Mergesort



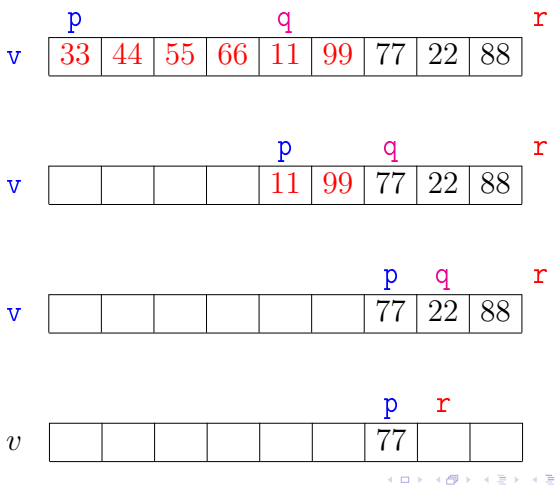
Mergesort



Mergesort



Mergesort



Mergesort

	p				q					r
v	33	44	55	66	11	99	77	22	88	
					p		q			r
v					11	99	77	22	88	
							p	q		r
v							77	22	88	
							p	r		
v							77			

Mergesort

	p				q					r
v	33	44	55	66	11	99	77	22	88	
					p		q			r
v					11	99	77	22	88	
							p	q		r
v							77	22	88	
								p	q	r
v								22	88	

Mergesort

	p				q					r
v	33	44	55	66	11	99	77	22	88	
					p		q			r
v					11	99	77	22	88	
							p	q		r
v							77	22	88	
								p	q	r
v								22	88	

Mergesort

	p				q					r
v	33	44	55	66	11	99	77	22	88	
					p		q			r
v					11	99	77	22	88	
							p	q		r
v							77	22	88	
								p	q	r
v								22	88	

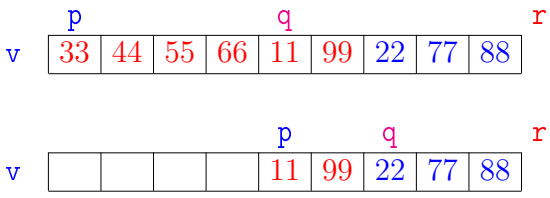
Mergesort

	p				q					r
v	33	44	55	66	11	99	77	22	88	
					p		q			r
v					11	99	77	22	88	
							p	q		r
v							77	22	88	

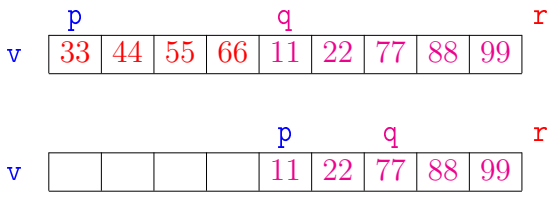
Mergesort

	p				q					r
v	33	44	55	66	11	99	22	77	88	
					p		q			r
v					11	99	22	77	88	
							p	q		r
v							22	77	88	

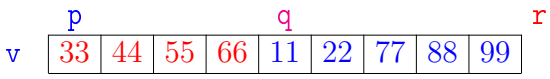
Mergesort



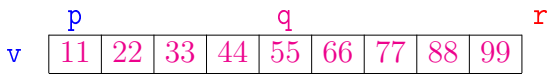
Mergesort



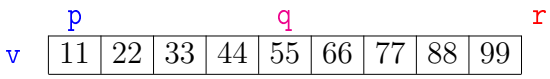
Mergesort



Mergesort



Mergesort

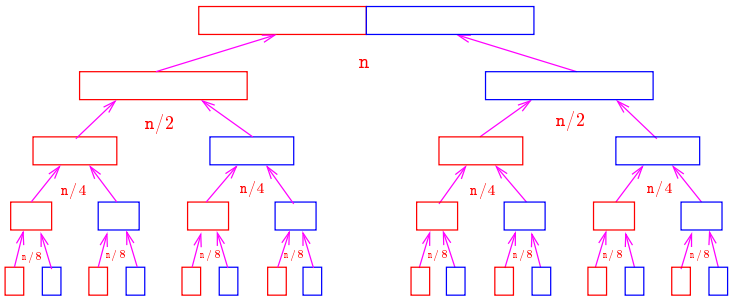


Correção

```
void mergeSort (int p, int r, int v[]) {
1   if (p < r-1) {
2       int q = (p + r)/2;
3       mergeSort(p, q, v);
4       mergeSort(q, r, v);
5       intercala(p, q, r, v);
    }
}
```

A função está correta?
A correção da função, que se apóia na correção do `intercala`, pode ser demonstrada por indução em $n := r - p$.

Consumo de tempo: versão MAC0122



Consumo de tempo: versão MAC0122

O consumo de tempo em cada nível da recursão é proporcional a n.

Há cerca de $\lg n$ níveis de recursão.

nível	consumo de tempo (proporcional a)
1	$\approx n$
2	$\approx n/2 + n/2$
3	$\approx n/4 + n/4 + n/4 + n/4$
...	...
$\lg n$	$\approx 1 + 1 + 1 + 1 \cdots + 1 + 1 + 1 + 1$
Total	$\approx n \lg n = O(n \lg n)$

Consumo de tempo: versão MAC0338

```
void mergeSort (int p, int r, int v[]) {
1   if (p < r-1) {
2       int q = (p + r)/2;
3       mergeSort(p, q, v);
4       mergeSort(q, r, v);
5       intercala(p, q, r, v);
    }
}
```

Consumo de tempo?

$T(n)$:= consumo de tempo quando $n = r - p$

Consumo de tempo: versão MAC0338

```
void mergeSort (int p, int r, int v[]) {
1   if (p < r-1) {
2       int q = (p + r)/2;
3       mergeSort(p, q, v);
4       mergeSort(q, r, v);
5       intercala(p, q, r, v);
    }
}
```

linha	consumo na linha (proporcional a)
1	?
2	?
3	?
4	?
5	?
$T(n)$	$= ?$

Consumo de tempo: versão MAC0338

```
void mergeSort (int p, int r, int v[]) {
1   if (p < r-1) {
2       int q = (p + r)/2;
3       mergeSort(p, q, v);
4       mergeSort(q, r, v);
5       intercala(p, q, r, v);
    }
}
```

linha	consumo na linha (proporcional a)
1	$= 1$
2	$= 1$
3	$= T(\lfloor n/2 \rfloor)$
4	$= T(\lceil n/2 \rceil)$
5	$= n$
$T(n)$	$= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n + 2$

Consumo de tempo: versão MAC0338

$T(n)$:= consumo de tempo quando $n = r - p$

$T(1) = 1$

$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n$ para $n = 2, 3, 4, \ldots$

Solução: $T(n)$ é $O(n \log n)$.

Demonstração: ...

Em MAC0338 vocês verão como estimar "a ordem" de recorrências.

Conclusão

O consumo de tempo da função `mergeSort` é proporcional a $n \lg n$.

O consumo de tempo da função `mergeSort` é $O(n \lg n)$.

Divisão e conquista

Algoritmos por divisão-e-conquista têm três passos em cada nível da recursão:

Dividir: o problema é dividido em subproblemas de tamanho menor;

Conquistar: os subproblemas são resolvidos recursivamente e subproblemas "pequenos" são resolvidos diretamente;

Combinar: as soluções dos subproblemas são combinadas para obter uma solução do problema original.

Exemplo: ordenação por intercalação (`mergeSort`).

`mergeSort`: versão iterativa

```
void mergeSort (int n, int v[]){
  int p, r;
  int b = 1;
  while (b < n) {
      p = 0;
      while (p + b < n) {
          r = p + 2*b;
          if (r > n) r = n;
          intercala(p, p+b, r, v);
          p = p + 2*b;
      }
      b = 2*b;
  }
}
```