Universidade Federal de Sergipe
Departamento de Sistemas de Informação

SINF0007 – Estrutura de Dados II

Dividir e Conquistar

10

Prof. Dr. Raphael Pereira de Oliveira
raphael.oliveira@academico.ufs.br

Introdução

Relembrando a Busca Sequencial (desordenado)

```
/* Created by Rodrigo Paes
/* a vetor de inteiros
/* n tamanho do vetor
/* x valor a ser procurado

int find_x(int *a, int n, int x){
    for (int i = 0; i < n; ++i){
        if (a[i] == x){
            return i;
        }
    }
    return -1;
}
```

Complexidade no pior caso **O(n)**

Ver código buscaSequencial.c

Relembrando a Busca Binária (ordenado)

```
/* Created by Rodrigo Paes
/* Left inclusivo
/* right exclusivo
int find_x(int *a, int left, int right, int x){
    if (left == right){
        if (x == a[left]) return left;
        else return -1;
    }
    int mid = (left + right) / 2;

    if (a[mid] == x) return mid;
    else if (x < a[mid]) return find_x(a, left, mid, x);
    else return find_x(a, mid+1, right, x);
}
```

Complexidade no pior caso **O(log n)**

Ver código buscaBinaria.c

Dividir e Conquistar

Dividir e Conquistar (Divide-and-Conquer)

Princípios:

- **Dividir**
  - Dividir em subproblemas
- **Conquistar**
  - Explorar cada subproblema
- **Combinar**
  - Combinar as soluções

Dividir e Conquistar (Divide-and-Conquer)

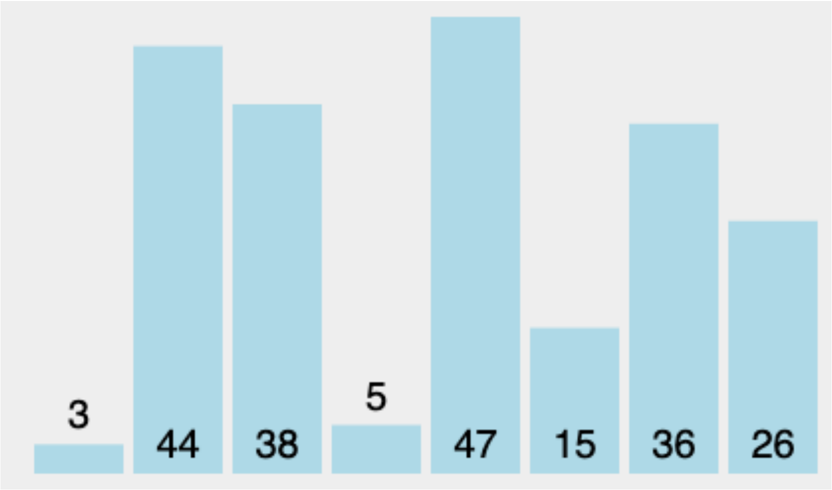
Exemplos:

- Merge Sort
- Potência
- Fibonacci
- Quick Sort

Merge Sort

Merge Sort

Objetivo: Ordenar um vetor desordenado



https://visualgo.net/en/sorting

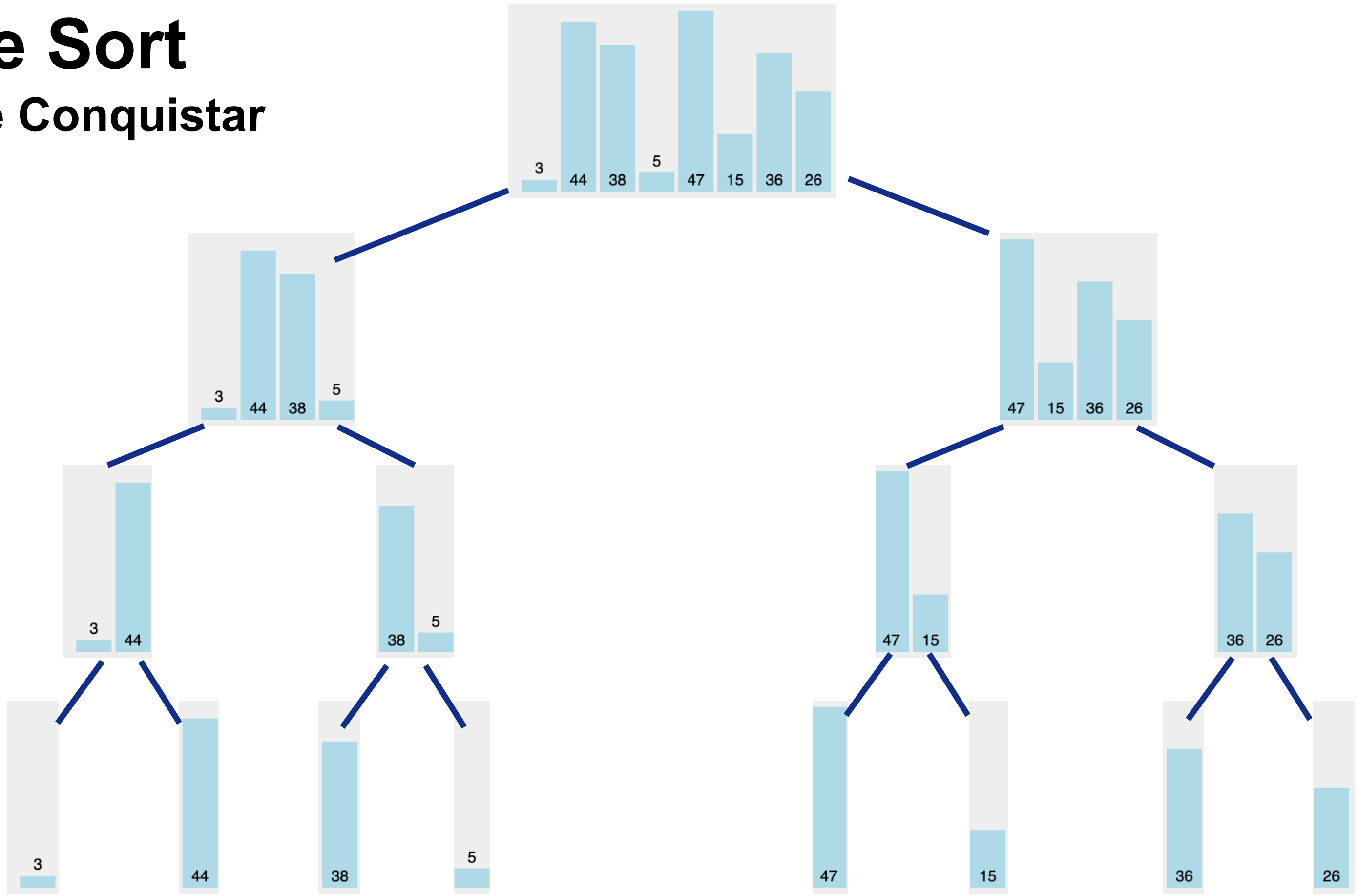
Merge Sort

Vamos aplicar os princípios:

- **Dividir**
  - Dividir em subproblemas
- **Conquistar**
  - Explorar cada subproblema
- **Combinar**
  - Combinar as soluções

Merge Sort

Dividir e Conquistar



https://visualgo.net/en/sorting

Merge Sort

Combinar

https://visualgo.net/en/sorting

Merge Sort

Combinar

https://visualgo.net/en/sorting

Merge Sort

Combinar

https://visualgo.net/en/sorting

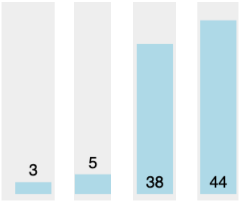
Merge Sort

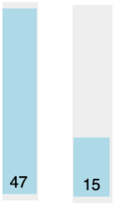
Combinar

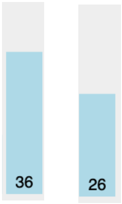
https://visualgo.net/en/sorting

Merge Sort

Combinar

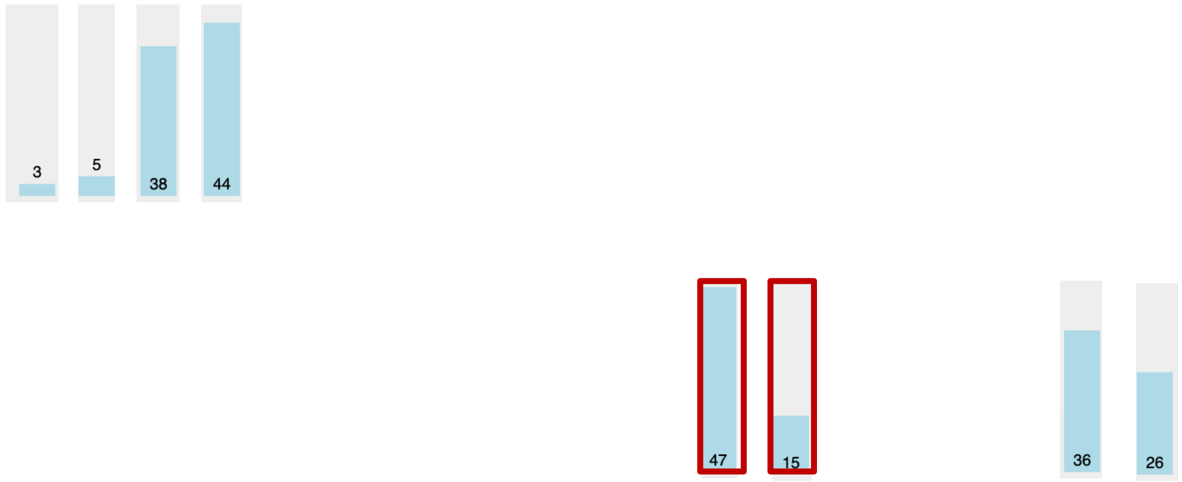






https://visualgo.net/en/sorting

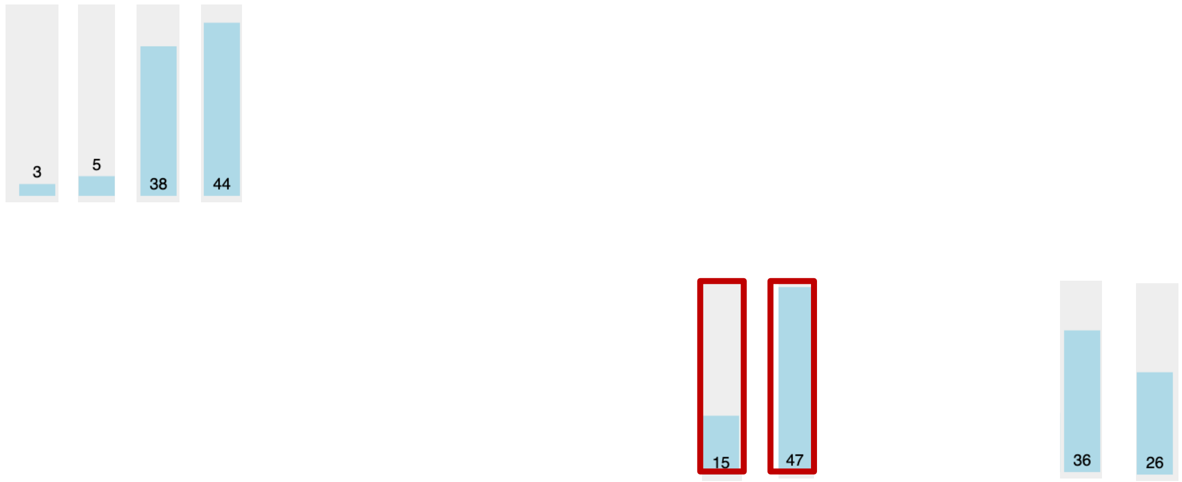
Merge Sort
Combinar



https://visualgo.net/en/sorting

Merge Sort

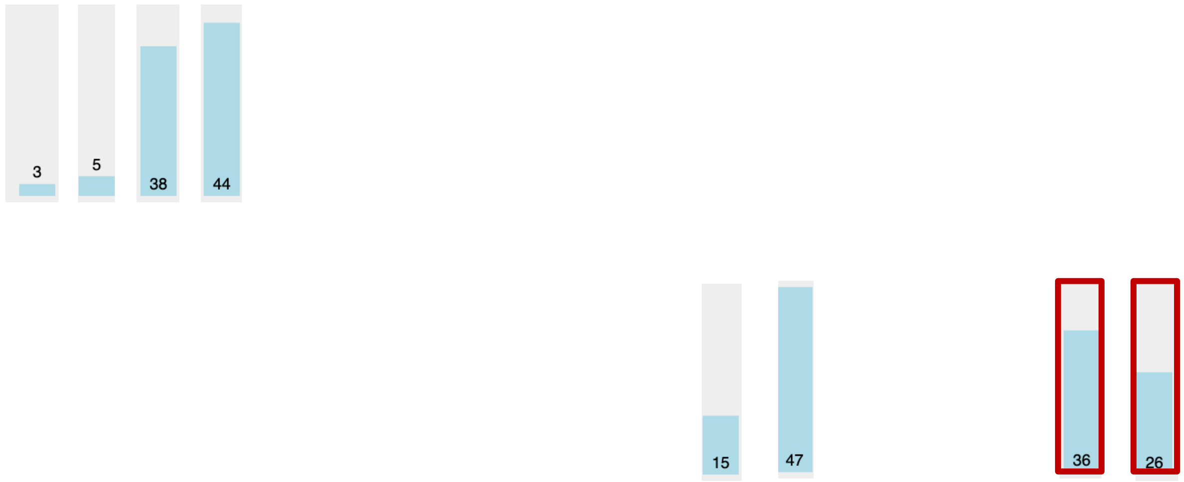
Combinar



https://visualgo.net/en/sorting

Merge Sort

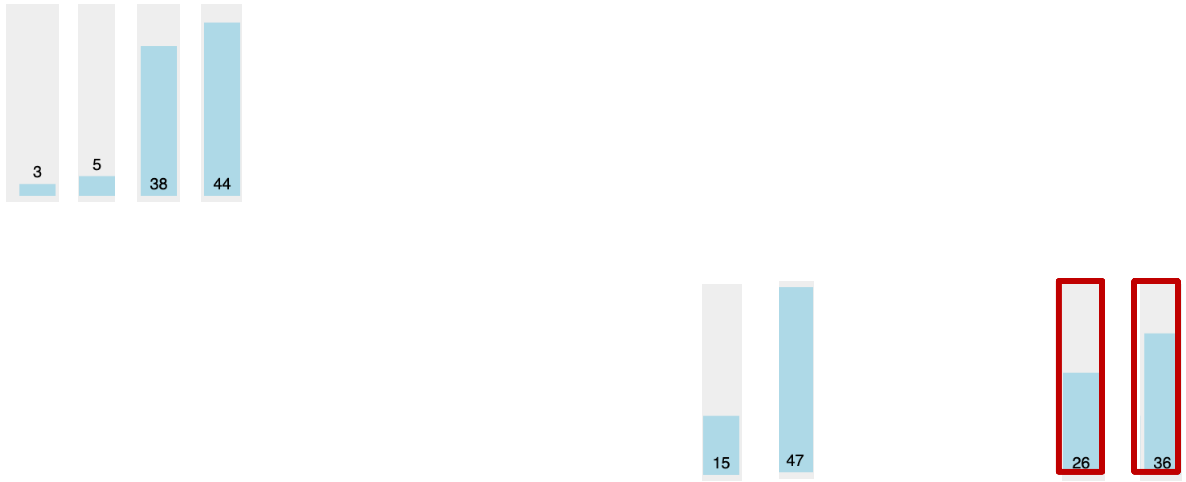
Combinar



https://visualgo.net/en/sorting

Merge Sort

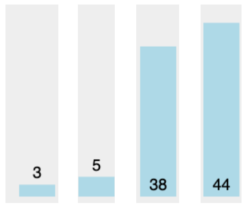
Combinar

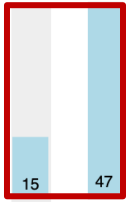


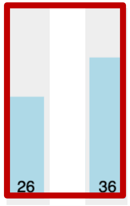
https://visualgo.net/en/sorting

Merge Sort

Combinar

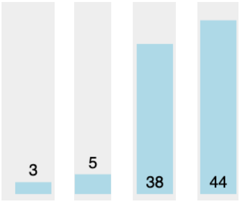


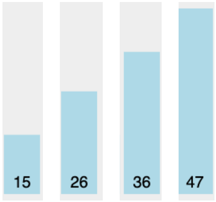




https://visualgo.net/en/sorting

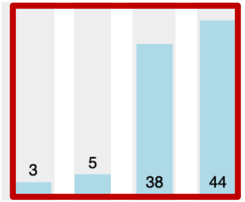
Merge Sort
Combinar

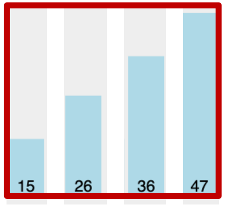




https://visualgo.net/en/sorting

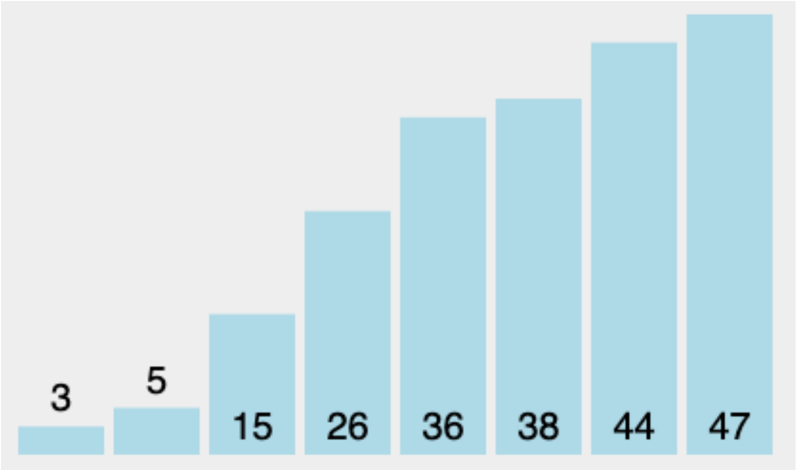
Merge Sort
Combinar





https://visualgo.net/en/sorting

Merge Sort
Combinar



https://visualgo.net/en/sorting

Merge Sort

Vamos começar com o **combinar vetores ordenados**:

```c
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

Algoritmo:
https://www.ime.usp.br/~pf/algoritmos/aulas/mrgsrt.html

Merge Sort

Vamos começar com o **combinar vetores ordenados**:

```c
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

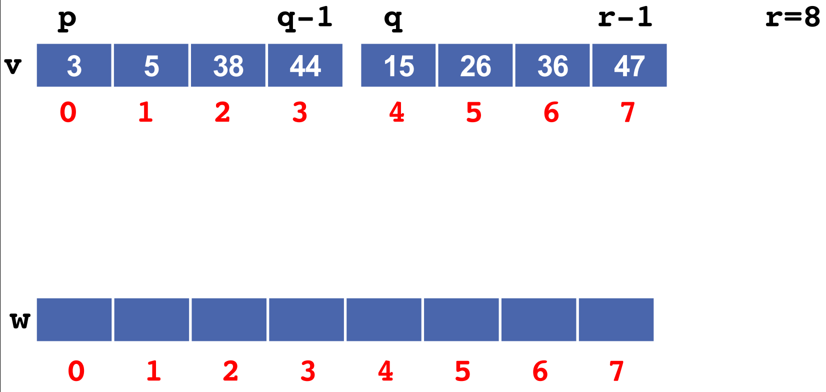
	p			q-1	q			r-1	r=8
v	3	5	38	44	15	26	36	47	
	0	1	2	3	4	5	6	7	

Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

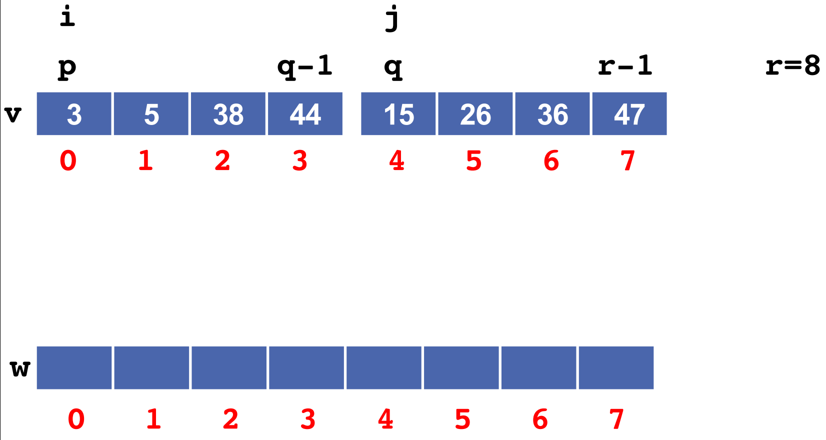


Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

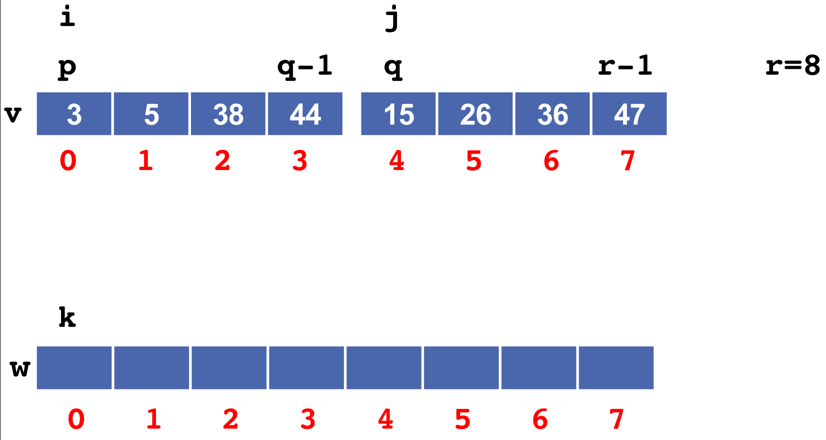


Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```c
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

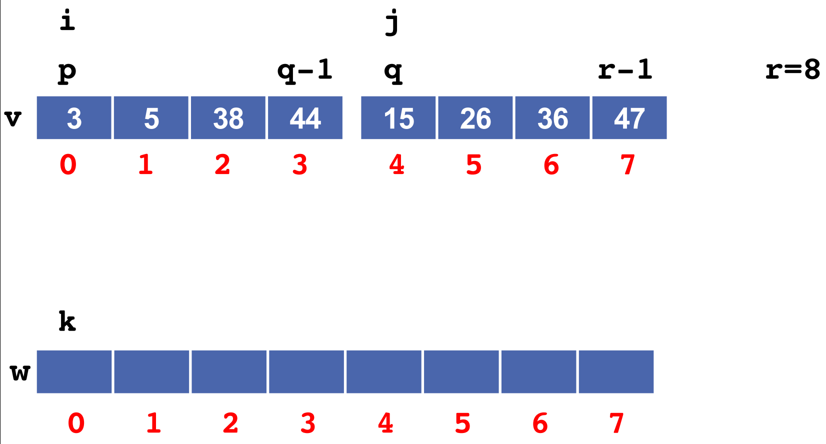


Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```c
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

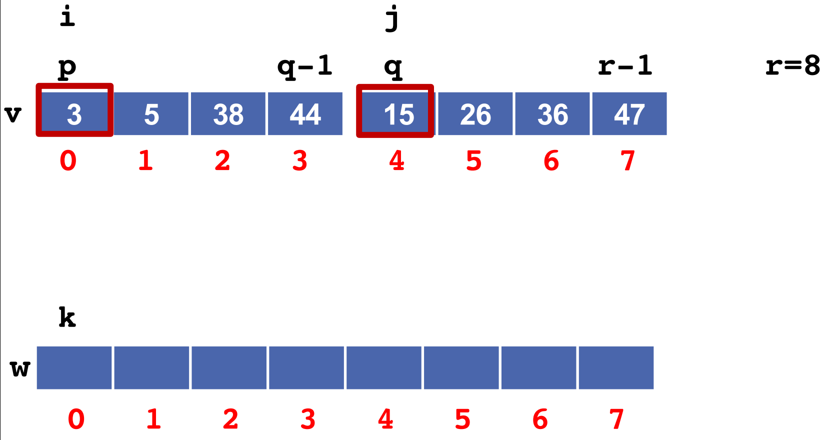


Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

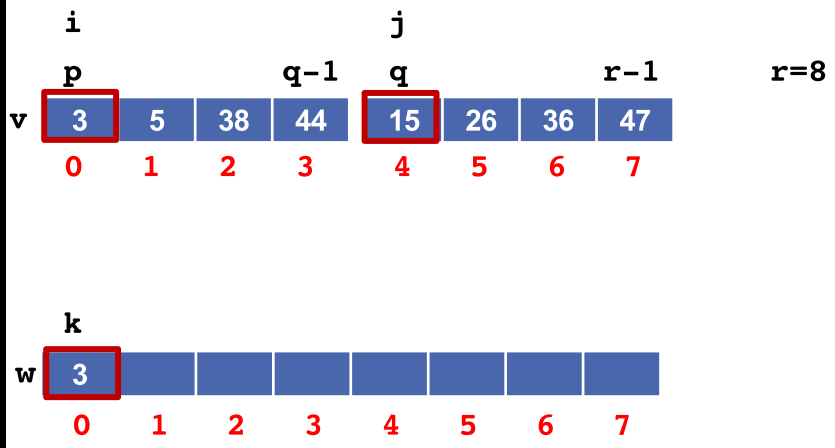


Merge Sort

Vamos começar com o **combinar vetores ordenados**:

```c
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

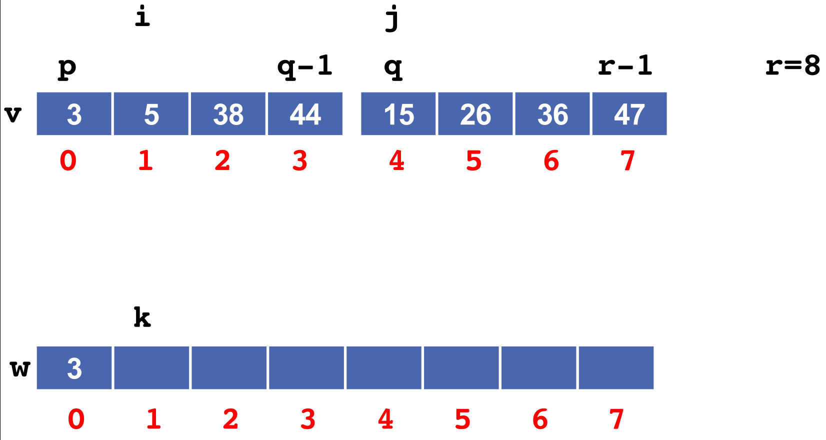


Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

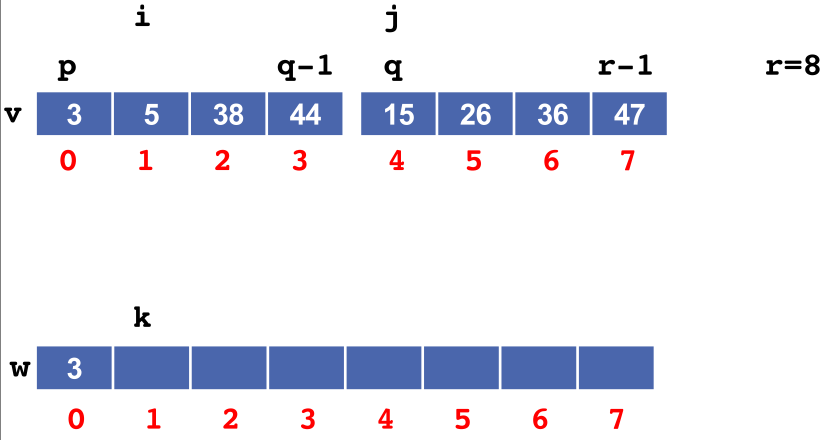


Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```c
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
     int *w;
     w = malloc ((r-p) * sizeof (int));
     int i = p, j = q;
     int k = 0;

     while (i < q && j < r) {
       if (v[i] <= v[j]) w[k++] = v[i++];
       else w[k++] = v[j++];
     }
     while (i < q) w[k++] = v[i++];
     while (j < r) w[k++] = v[j++];
     for (i = p; i < r; ++i) v[i] = w[i-p];
     free (w);
}
```

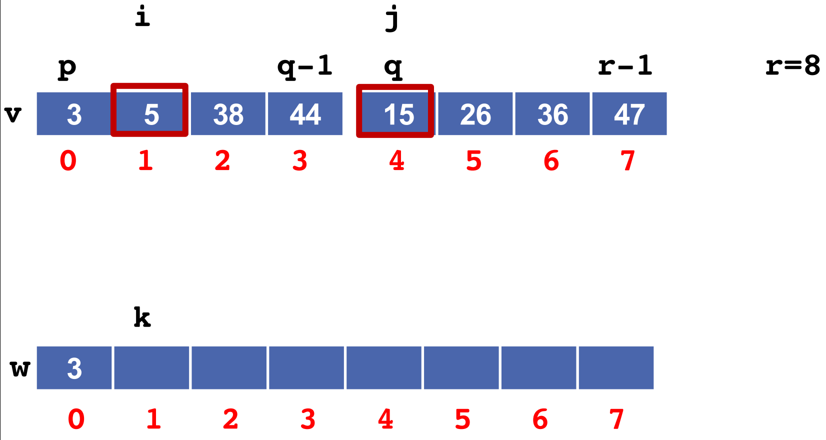


Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```c
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

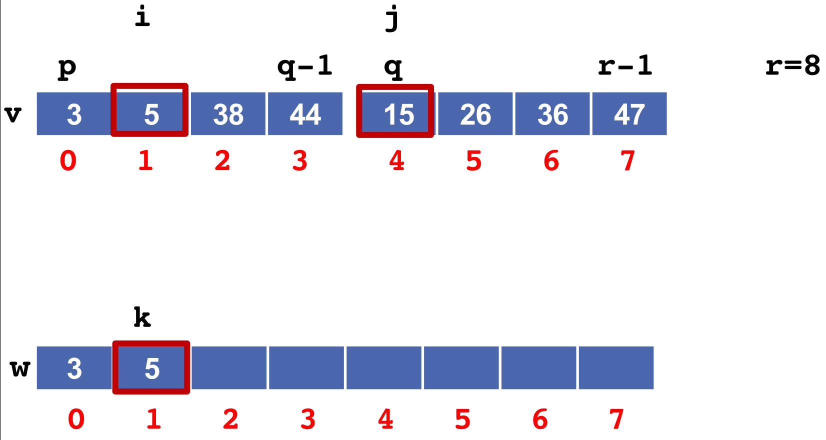


Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```c
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

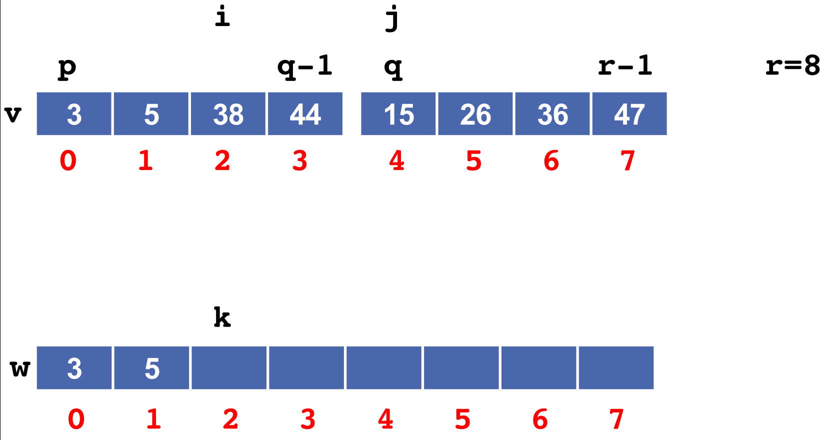


Merge Sort

Vamos começar com o **combinar vetores ordenados**:

```c
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

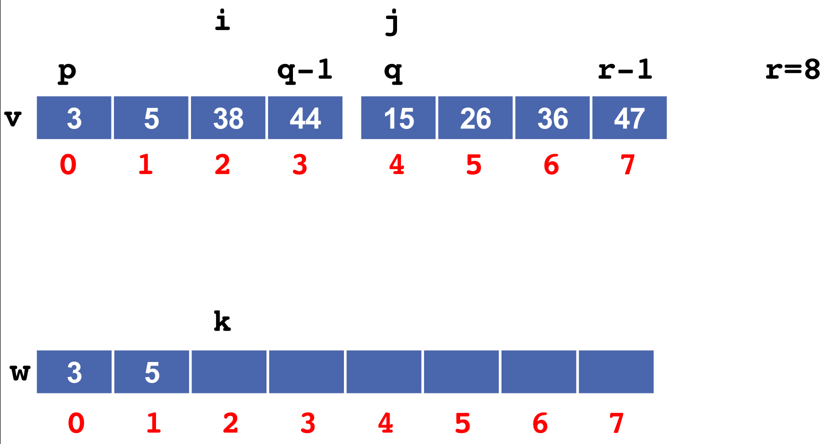


Merge Sort

Vamos começar com o **combinar vetores ordenados**:

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

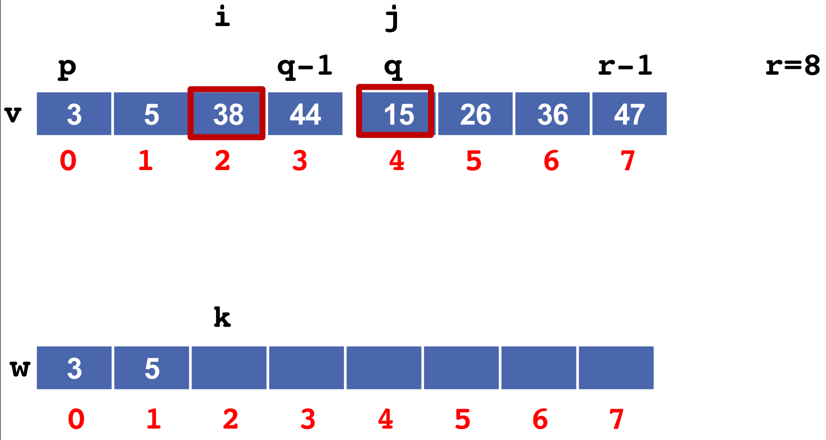


Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

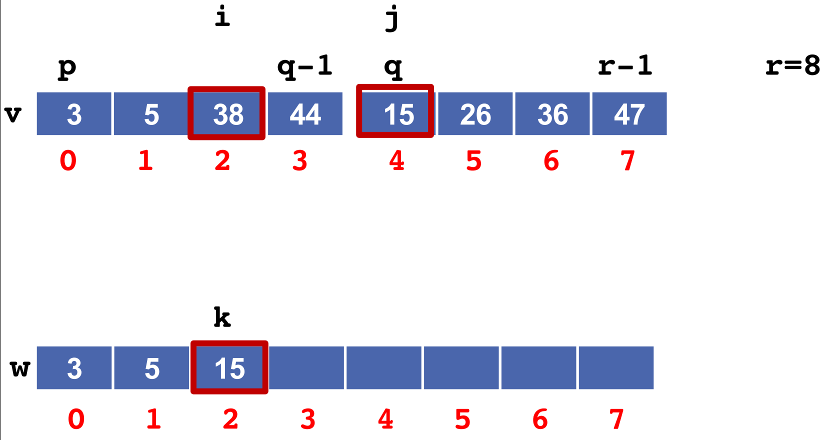


Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
     int *w;
     w = malloc ((r-p) * sizeof (int));
     int i = p, j = q;
     int k = 0;

     while (i < q && j < r) {
        if (v[i] <= v[j]) w[k++] = v[i++];
        else w[k++] = v[j++];
     }
     while (i < q) w[k++] = v[i++];
     while (j < r) w[k++] = v[j++];
     for (i = p; i < r; ++i) v[i] = w[i-p];
     free (w);
}
```

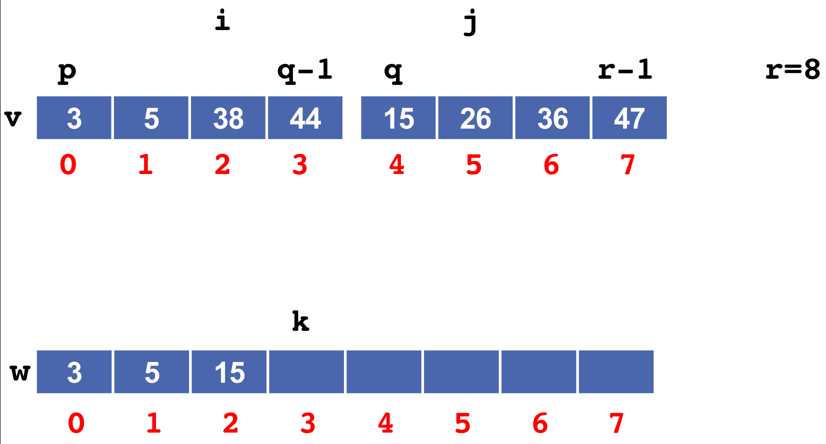


Merge Sort

Vamos começar com o **combinar vetores ordenados**:

```c
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

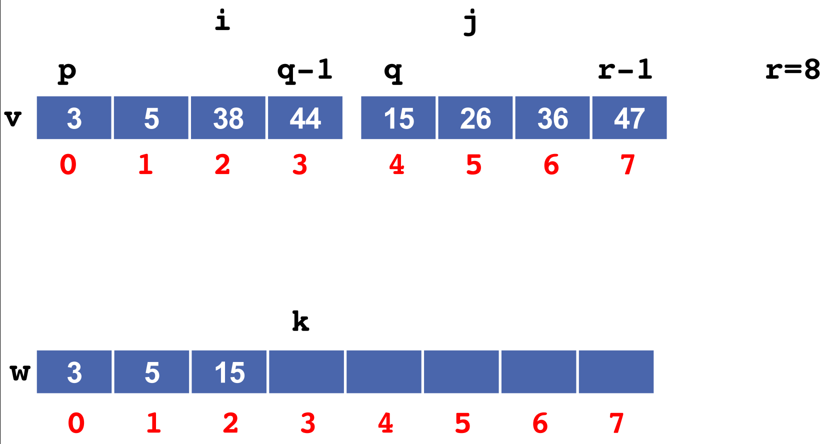


Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```



Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```c
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

			i			j			
	p			q-1	q			r-1	r=8
v	3	5	38	44	15	26	36	47	
	0	1	2	3	4	5	6	7	

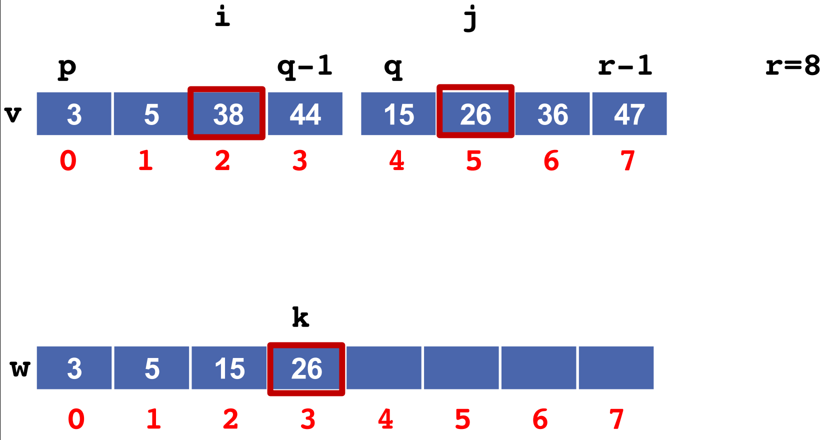
				k				
w	3	5	15					
	0	1	2	3	4	5	6	7

Merge Sort

Vamos começar com o **combinar vetores ordenados**:

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
     int *w;
     w = malloc ((r-p) * sizeof (int));
     int i = p, j = q;
     int k = 0;

     while (i < q && j < r) {
        if (v[i] <= v[j]) w[k++] = v[i++];
        else w[k++] = v[j++];
     }
     while (i < q) w[k++] = v[i++];
     while (j < r) w[k++] = v[j++];
     for (i = p; i < r; ++i) v[i] = w[i-p];
     free (w);
}
```

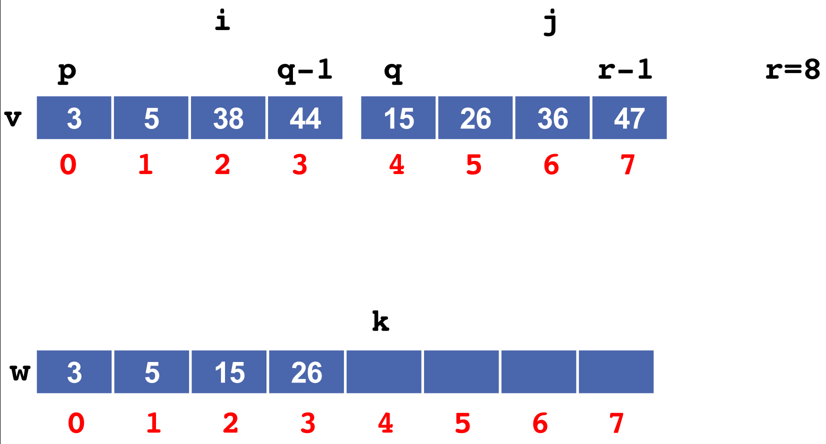


Merge Sort

Vamos começar com o **combinar vetores ordenados**:

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
     int *w;
     w = malloc ((r-p) * sizeof (int));
     int i = p, j = q;
     int k = 0;

     while (i < q && j < r) {
        if (v[i] <= v[j]) w[k++] = v[i++];
        else w[k++] = v[j++];
     }
     while (i < q) w[k++] = v[i++];
     while (j < r) w[k++] = v[j++];
     for (i = p; i < r; ++i) v[i] = w[i-p];
     free (w);
}
```

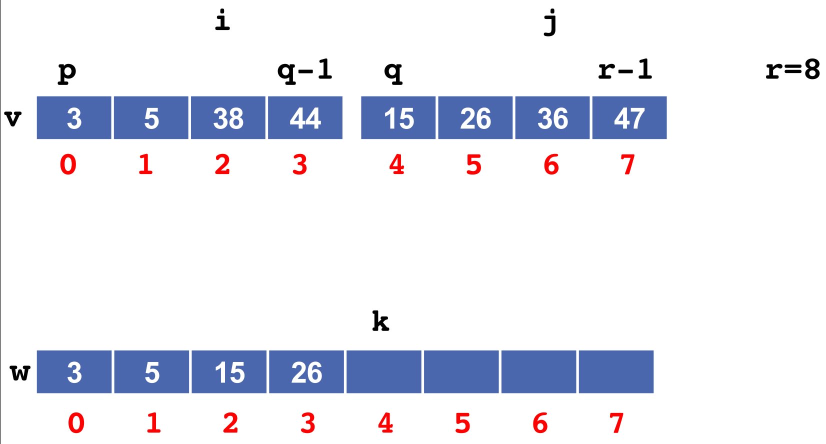


Merge Sort

Vamos começar com o **combinar vetores ordenados**:

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
     int *w;
     w = malloc ((r-p) * sizeof (int));
     int i = p, j = q;
     int k = 0;

     while (i < q && j < r) {
       if (v[i] <= v[j]) w[k++] = v[i++];
       else w[k++] = v[j++];
     }
     while (i < q) w[k++] = v[i++];
     while (j < r) w[k++] = v[j++];
     for (i = p; i < r; ++i) v[i] = w[i-p];
     free (w);
}
```

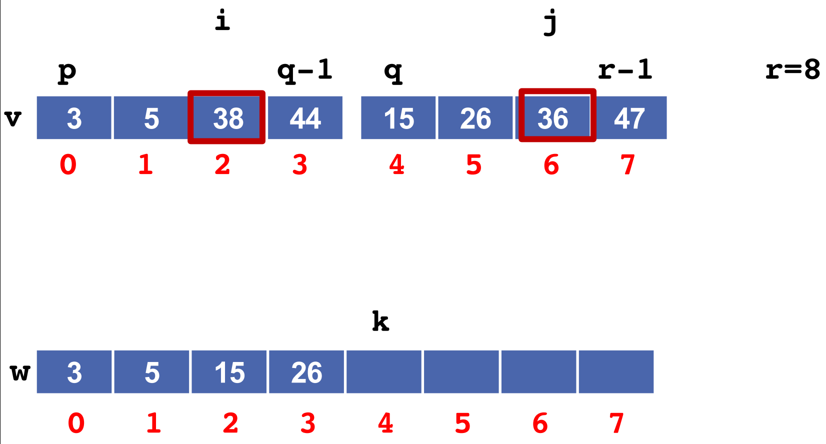


Merge Sort

Vamos começar com o **combinar vetores ordenados**:

```c
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

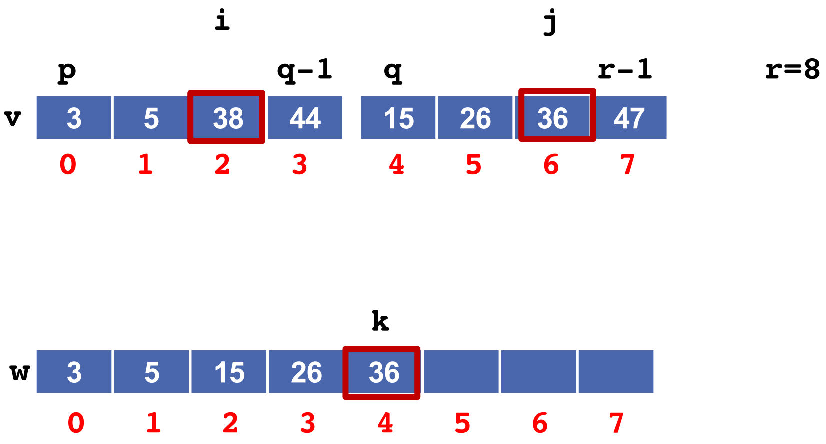


Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

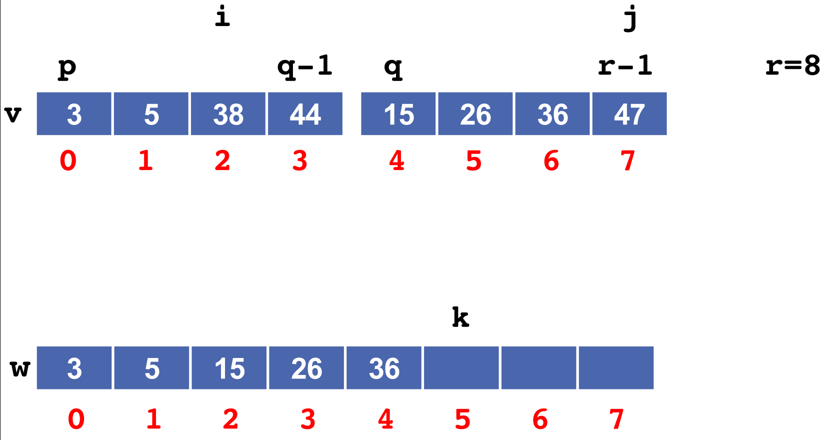


Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```c
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```



Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```c
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
     int *w;
     w = malloc ((r-p) * sizeof (int));
     int i = p, j = q;
     int k = 0;

     while (i < q && j < r) {
        if (v[i] <= v[j]) w[k++] = v[i++];
        else w[k++] = v[j++];
     }
     while (i < q) w[k++] = v[i++];
     while (j < r) w[k++] = v[j++];
     for (i = p; i < r; ++i) v[i] = w[i-p];
     free (w);
}
```

			i					j	
	p			q-1	q			r-1	r=8
v	3	5	38	44	15	26	36	47	
	0	1	2	3	4	5	6	7	

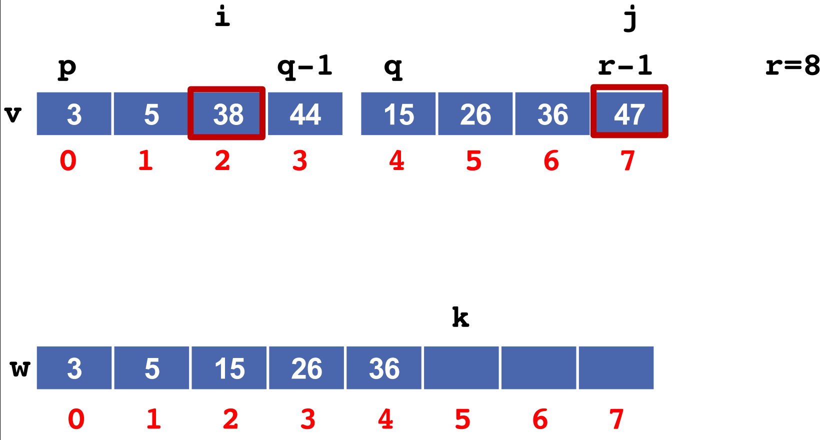
						k		
w	3	5	15	26	36			
	0	1	2	3	4	5	6	7

Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```c
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

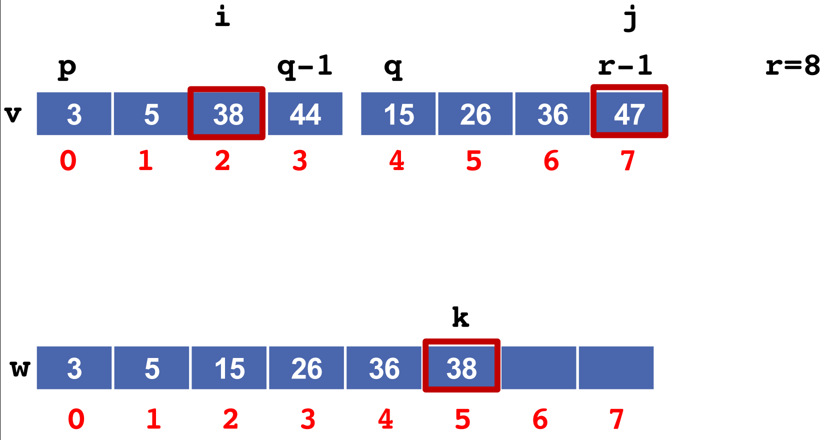


Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

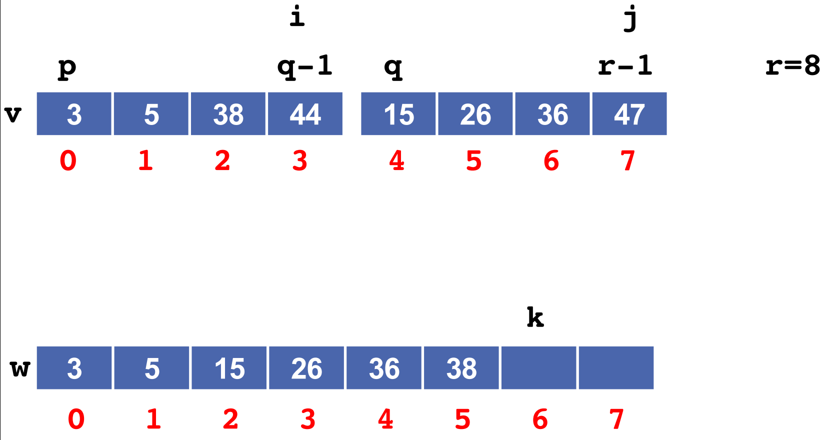


Merge Sort

Vamos começar com o **combinar vetores ordenados**:

```c
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```



Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```c
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

				i				j	
	p			q-1	q			r-1	r=8
v	3	5	38	44	15	26	36	47	
	0	1	2	3	4	5	6	7	

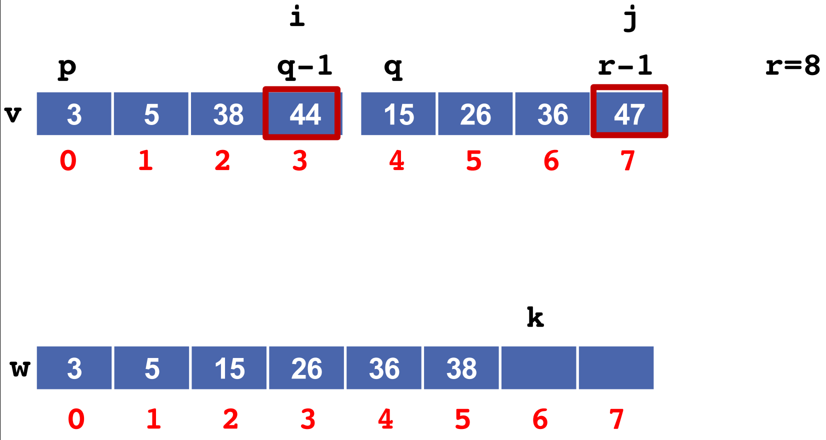
							k	
w	3	5	15	26	36	38		
	0	1	2	3	4	5	6	7

Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

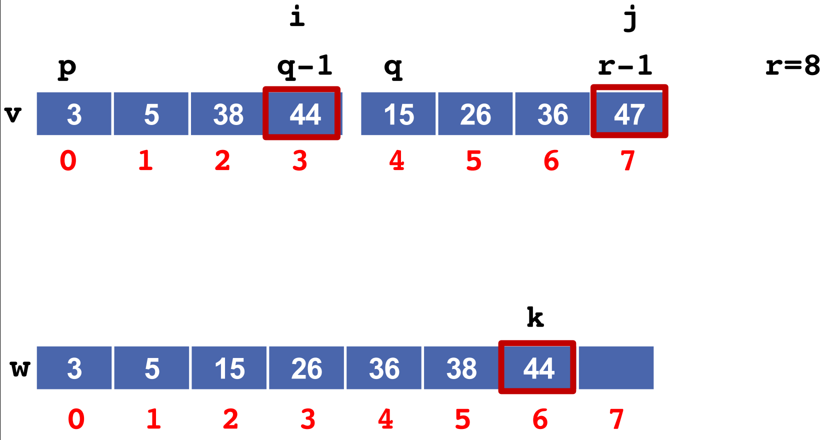


Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

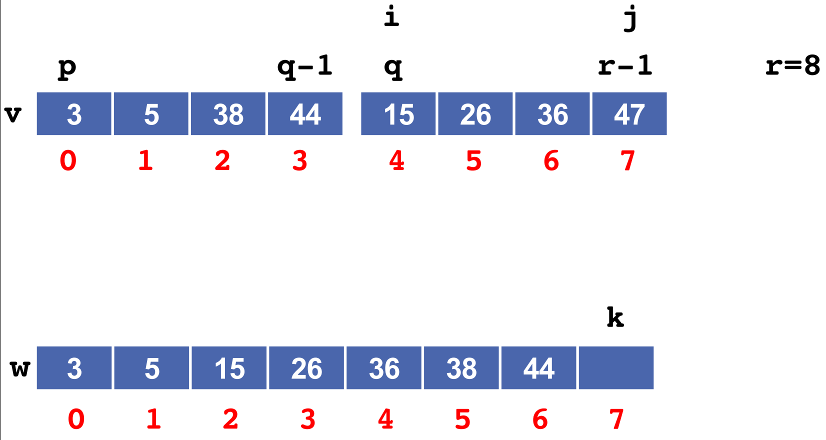


Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

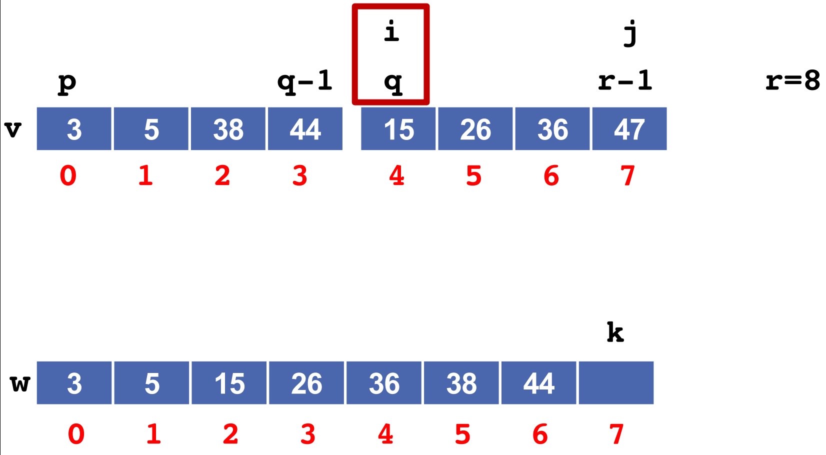


Merge Sort

Vamos começar com o **combinar vetores ordenados**:

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```



Merge Sort

Vamos começar com o **combinar vetores ordenados:**

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
     int *w;
     w = malloc ((r-p) * sizeof (int));
     int i = p, j = q;
     int k = 0;

     while (i < q && j < r) {
        if (v[i] <= v[j]) w[k++] = v[i++];
        else w[k++] = v[j++];
     }
     while (i < q) w[k++] = v[i++];
     while (j < r) w[k++] = v[j++];
     for (i = p; i < r; ++i) v[i] = w[i-p];
     free (w);
}
```

	p			q-1	i q			j r-1	r=8
v	3	5	38	44	15	26	36	47	
	0	1	2	3	4	5	6	7	

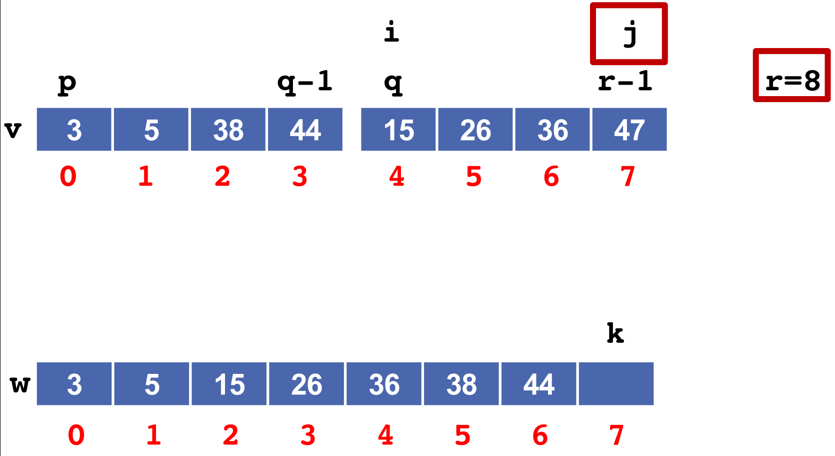
								k
w	3	5	15	26	36	38	44	
	0	1	2	3	4	5	6	7

Merge Sort

Vamos começar com o **combinar vetores ordenados**:

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

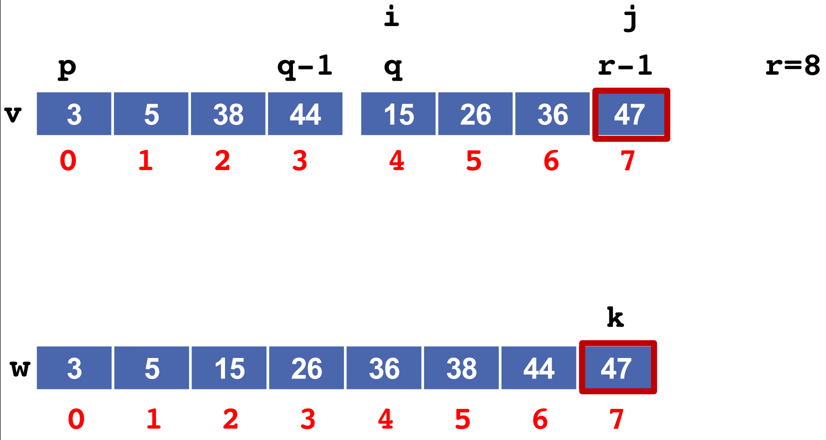


Merge Sort

Vamos começar com o **combinar vetores ordenados**:

```c
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```



Merge Sort

Vamos começar com o **combinar vetores ordenados**:

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

j=8

r=8

					i			
	p			q-1	q			r-1
v	3	5	38	44	15	26	36	47
	0	1	2	3	4	5	6	7

k=8

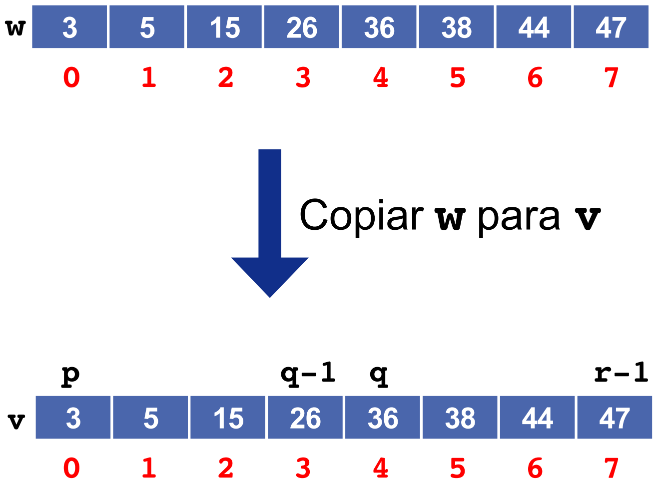
w	3	5	15	26	36	38	44	47
	0	1	2	3	4	5	6	7

Merge Sort

Vamos começar com o **combinar vetores ordenados**:

```c
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```

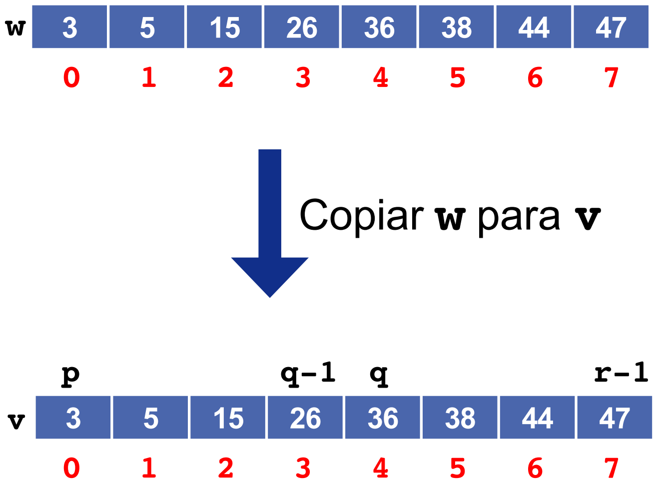


Merge Sort

Vamos começar com o **combinar vetores ordenados**:

```
// A função recebe vetores crescentes v[p..q-1]
// e v[q..r-1] e rearranja v[p..r-1] em ordem
// crescente.
void intercala (int p, int q, int r, int v[]){
    int *w;
    w = malloc ((r-p) * sizeof (int));
    int i = p, j = q;
    int k = 0;

    while (i < q && j < r) {
      if (v[i] <= v[j]) w[k++] = v[i++];
      else w[k++] = v[j++];
    }
    while (i < q) w[k++] = v[i++];
    while (j < r) w[k++] = v[j++];
    for (i = p; i < r; ++i) v[i] = w[i-p];
    free (w);
}
```



Merge Sort

Basta chamar o método **mergeSort** para ordenar todo o *array*:

```c
// A função mergesort rearranja o vetor
// v[p..r-1] em ordem crescente.
void mergeSort (int p, int r, int v[]){
    if (p < r-1) {
      int q = (p + r)/2;
      mergeSort (p, q, v);
      mergeSort (q, r, v);
      intercala (p, q, r, v);
    }
}
```

	p			q-1	q			r-1
v	3	44	38	5	47	15	36	26
	0	1	2	3	4	5	6	7

Complexidade no pior caso **O(n log n)**

Ver código mergeSort.c

Algortimo: https://www.ime.usp.br/~pf/algoritmos/aulas/mrgsrt.html

Potência

Potência

Entrada

- um número **x**
- um inteiro **n** **>= 0**

Saída

- x^n

Potência (naive)

Podemos resolver utilizando:

```
double potencia(int x, int n){
    int total = 1;
    for (int i=0; i<n; i++)
        total *= x;
    return total;
}
```

Porém, não é muito eficiente Θ(n)

Podemos resolver melhor que Θ(n) ?

Potência

Vamos aplicar o princípio:

- **Dividir e Conquistar**

Potência

- **Dividir e Conquistar**
  - $\mathbf{x^n = x^{n/2} * x^{n/2}}$, se $\mathbf{n}$ for **par**
  - $\mathbf{x^n = x^{(n-1)/2} * x^{(n-1)/2} * x}$, se $\mathbf{n}$ for **ímpar**

- Parece que dividimos em 2 subproblemas de tamanho n/2
- Mas na verdade são o mesmo subproblema
- Então, só precisamos computar 1
- Se conseguirmos isso, teremos o problema com a mesma "cara" da busca binária
- Conseguiremos **O(lg(n))**

Potência (exemplo)

Ver código potencia.c

```
// n >=1
double powering(double x, int n){
    if (n == 1)
        return x;

    double half;

    if (n % 2 == 0){
        half = powering(x, n/2);
        return half * half;
    }else{
        half = powering(x, (n-1)/2 );
        return half * half * x;
    }
}
```

#1. `printf("%.2lf", powering(2, 5));`

#10. 32.00

#2. Call 0. `powering(2, 5) // x=2 e n=5`

#3. `half = powering(2, (5-1)/2) //(?)`

#9. `return 4 * 4 * 2`

#4. Call 1. `powering(2, 2) // x=2 e  n=2`

#5. `half = powering(2, 2/2) //(?)`

#8. `return 2 * 2`

#6. Call 2. `powering(2, 1) // x=2 e  n=1`

#7. `return 2`

Algoritmo:
https://github.com/r0drigopaes/paa/blob/master/powering.cpp

Fibonacci

Fibonacci

Definição:

$$f(n) = \begin{cases} n = 0 & 0 \\ n = 1 & 1 \\ n > 1 & f(n-1) + f(n-2) \end{cases}$$

Fib	0	1	2	3	4	5	6	7	8	9	10	11	…
Res	0,	1,	1,	2,	3,	5,	8,	13,	21,	34,	55,	89,	…

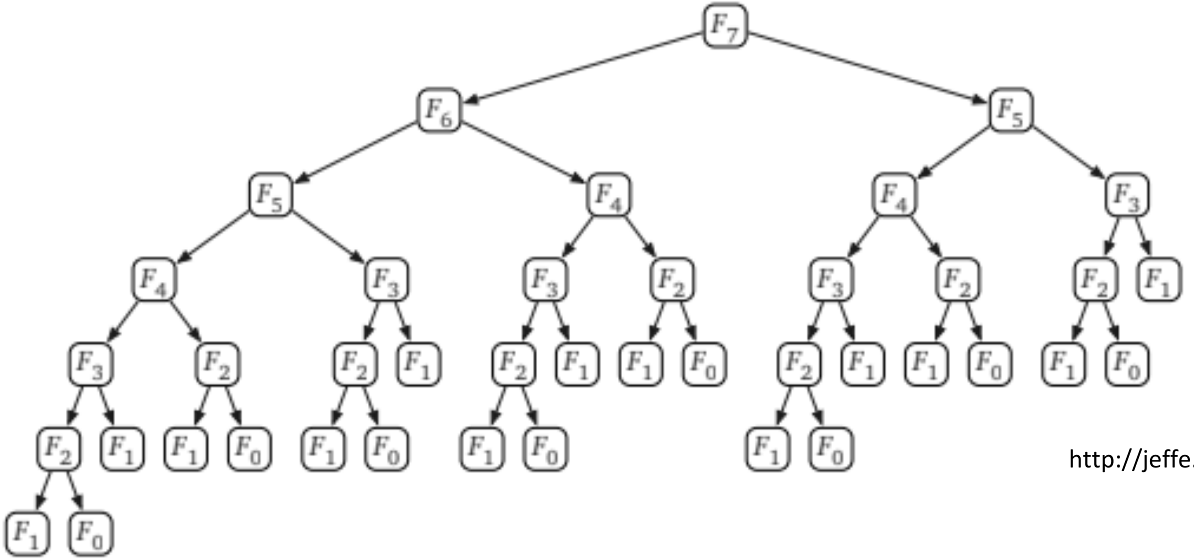
Fibonacci (naive)

Podemos resolver da seguinte forma:

```
// n >=0
double fibNaive(int n){
    if (n <= 1)
        return n;
    return fibNaive(n-1) + fibNaive(n-2);
}
```

- O problema é dividido em 2, ligeiramente menores
- Contudo, ao fazer a árvore, veremos que ela cresce exponencialmente!

Fibonacci



Source:
http://jeffe.cs.illinois.edu/teaching/algorithms/

Que tal montar a árvore para o Fibonacci de 50?

Fibonacci (bottom up)

Podemos começar resolvendo o de 0 e o de 1 e depois subir até n:

```c
/* n>=0 */
long long int fibBottomUp(int n){
    if (n <= 1) return n;
    long long int f1, f2, f;
    f1 = 1;
    f2 = 0;
    for (int i = 2; i <= n; ++i){
        f = f1 + f2;
        f2 = f1;
        f1 = f;
    }
    return f;
}
```

Resolvemos em O(n)

Ver o código fibonacci.c

Fibonacci (usando Exponenciação recursiva)

Teorema:

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

Ora, se sabemos fazer a exponenciação em **O(lg n),** também resolveremos fibonacci na mesma ordem;

Para n=2

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

$$\begin{pmatrix} \boxed{1} & \boxed{1} \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \boxed{1} & 1 \\ \boxed{1} & 0 \end{pmatrix} = \begin{pmatrix} 1*1 + 1*1 & \\ & \end{pmatrix}$$

$$\begin{pmatrix} \boxed{1} & \boxed{1} \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & \boxed{1} \\ 1 & \boxed{0} \end{pmatrix} = \begin{pmatrix} 2 & 1*1 + 1*0 \\ & \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 \\ \boxed{1} & \boxed{0} \end{pmatrix} \begin{pmatrix} \boxed{1} & 1 \\ \boxed{1} & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1*1 + 0*1 & \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 \\ \boxed{1} & \boxed{0} \end{pmatrix} \begin{pmatrix} 1 & \boxed{1} \\ 1 & \boxed{0} \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 1*1 + 0*0 \end{pmatrix}$$

Ou seja, quando n=2

2	1
1	1

F_{n+1}	F_n
F_n	F_{n-1}

$F_{n-1} = F_1 = 1$

$F_n = F_2 = 1$

$F_{n+1} = F_3 = 2$

n=3

3	2
2	1

n=4

5	3
3	2

n=5

8	5
5	3

n=6

13	8
8	5

...

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

F_{n+1}	F_n
F_n	F_{n-1}

Fib	0	1	2	3	4	5	6	7	8	9	10	11	…
Res	0,	1,	1,	2,	3,	5,	8,	13,	21,	34,	55,	89,	…

Ver o código fibonacciPowering.c

Fibonacci - curiosidade

- Quem é ***melhor***?

n	10	20	30	50	100
Recursão	8ms	1s	2min	21dias	10^9anos
Iteração	1/6ms	1/3ms	1/2ms	3/4ms	1,5ms

- Estimativa de tempo para ***Fibonacci*** (Brassard e Bradley, 1996) implementado em ***Pascal*** em um **CDC CYBER 835** (foto ao lado).

Quick Sort

(ver a aula NA03 a explicação e algoritmo)

Referências

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. **Introduction to Algorithms, Third Edition** (3rd. ed.). The MIT Press.
- Robert Sedgewick. 2002. **Algorithms in C** (3rd. ed.). Addison-Wesley Longman Publishing Co., Inc., USA.
- Material baseado nos slides de **Rodrigo Paes**, Programação Avançada. Instituto de Computação. Universidade Federal de Alagoas (UFAL), Maceió, Brasil. (https://docs.google.com/presentation/d/14zBvXvvaB2sbjqOeIfRQoEjSjOzuWTWf87rVDZHoeYQ/edit?usp=sharing)
- Material baseado no material de **Paulo Feofiloff**, Algoritmos. Departamento de Computação. Universidade de São Paulo (USP), São Paulo, Brasil. (https://www.ime.usp.br/~pf/algoritmos/aulas/mrgsrt.html)