Universidade de Brasília
Faculdade UnB Gama
Disciplina: Estrutura de Dados e Algoritmos - EDA

Métodos de Ordenação

Algoritmos e Complexidades

Prof. Nilton Correia da Silva

9 de agosto de 2022

Agenda

Objetivo

- Ordenação

Métodos de Ordenação

- Seleção
- QuickSort
- Inserção
- Bolha
- Intercalação (Merge Sort)

Aplicação

- Análise Comparativa

Objetivo
Ordenação

- Reconhecer diferentes abordagens algorítmicas de ordenação.
- Compreender as complexidades computacionais de diferentes algoritmos de ordenação.
- Compreender os algoritmos básicos e os respectivos códigos dos métodos de Seleção, QuickSort, Inserção, Bolha e Merge.

Ordenação
Utilidade

- O dia-a-dia da computação traz uma alta demanda por encontrar uma dado específico dentro de um conjunto de dados.
- É comprovado que encontrar um dado específico em um conjunto ordenado requer menos esforço computacional do que se o conjunto estiver desordenado.

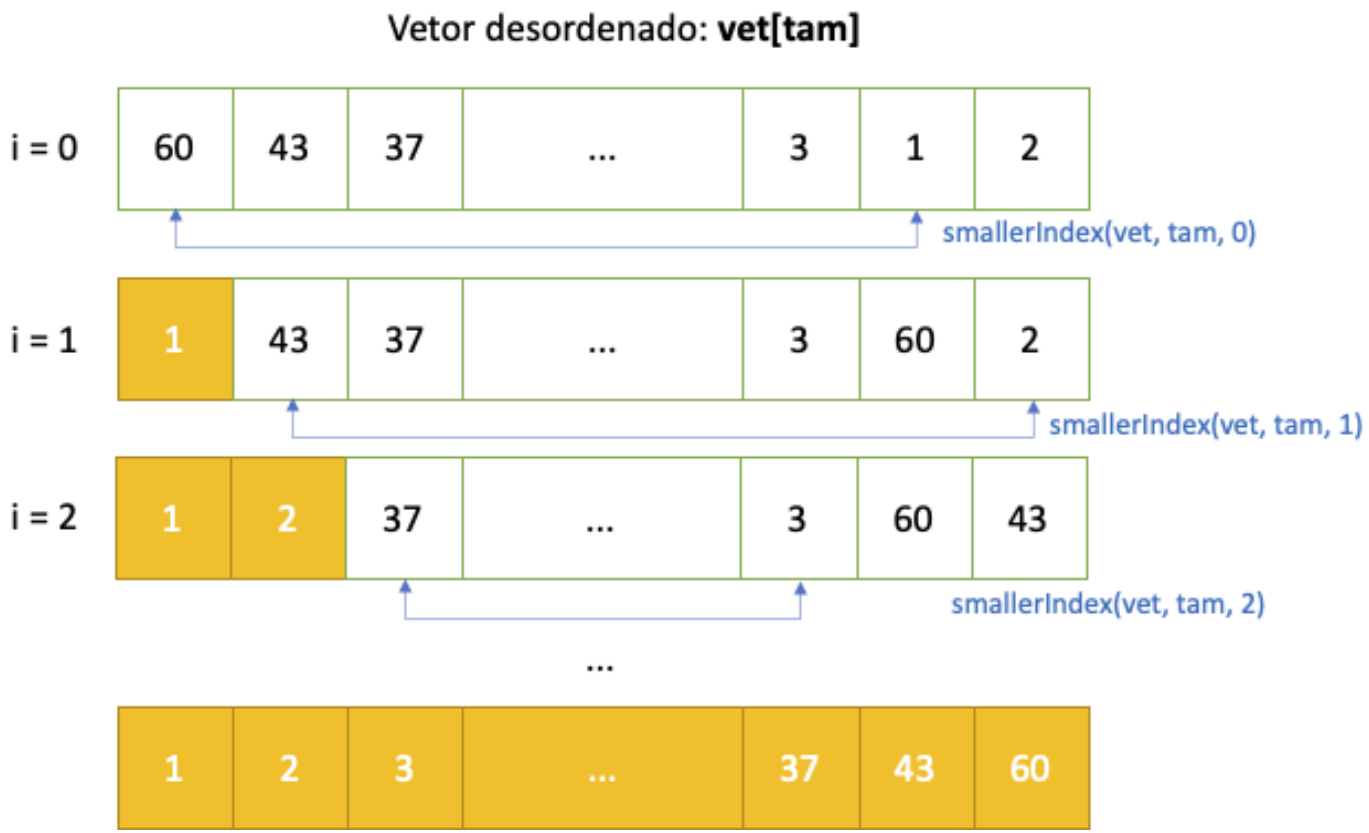
Seleção
Código: Encontrar menor elemento


```
1  int smallerIndex(int vet[], int tam, int ini){
2      int min = ini, j;
3      for(j=ini+1; j<tam; j++){
4          if(vet[min] > vet[j])
5              min = j;
6      }
7      return min;
8  }
```

Seleção
Código


```
1  void selectionSort(int vet[], int tam){
2      int i, min, aux;
3      for(i=0; i<tam; i++){
4  //Acha posicao do menor elemento a partir de i:
5          min = smallerIndex(vet, tam, i);
6          aux = vet[i];
7          vet[i] = vet[min];
8          vet[min] = aux;
9      }
10 }
```

Seleção
Algoritmo



Performance

- Melhor Caso: O(N^2).
- Pior Caso: O(N^2).

Considerações de sua aplicação

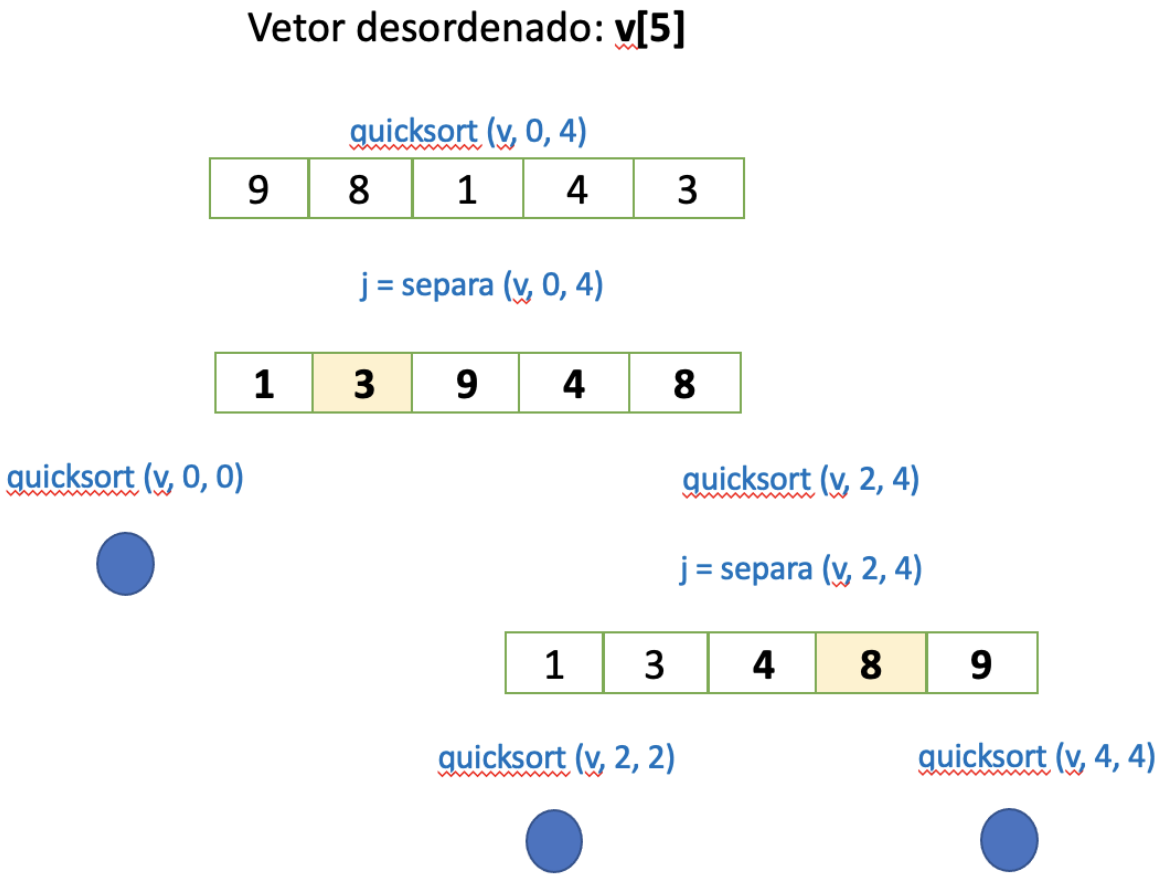
- Estável: não altera a ordem de chaves iguais

QuickSort
Código: Particionamento


```
1  // Recebe vetor v[p..r] com p <= r. Rearranja
2  // os elementos do vetor e devolve j em p..r
3  // tal que v[p..j-1] <= v[j] < v[j+1..r].
4  int separa (int v[], int p, int r) {
5     int c = v[r]; // pivô
6     int t, j = p;
7     for (int k = p; k < r; ++k)
8        if (v[k] <= c) {
9           t = v[j], v[j] = v[k], v[k] = t;
10          ++j;
11       }
12    t = v[j], v[j] = v[r], v[r] = t;
13    return j;
14 }
```

QuickSort
Código


```
1   // Esta função rearranja qualquer vetor
2   // v[p..r] em ordem crescente.
3
4   void quicksort (int v[], int p, int r)
5   {
6      if (p < r) {
7         int j = separa (v, p, r);
8         quicksort (v, p, j-1);
9         quicksort (v, j+1, r);
10     }
11  }
```


Vetor desordenado: v[5]
quicksort (v, 0, 4)
9 8 1 4 3
j = separa (v, 0, 4)
1 3 9 4 8
quicksort (v, 0, 0)
quicksort (v, 2, 4)
j = separa (v, 2, 4)
1 3 4 8 9
quicksort (v, 2, 2)
quicksort (v, 4, 4)

Performance

- Melhor Caso: O(N.LogN).
- Pior Caso: O(N^2). Quando vetor já estiver ordenado ou quase ordenado (caso Raro).

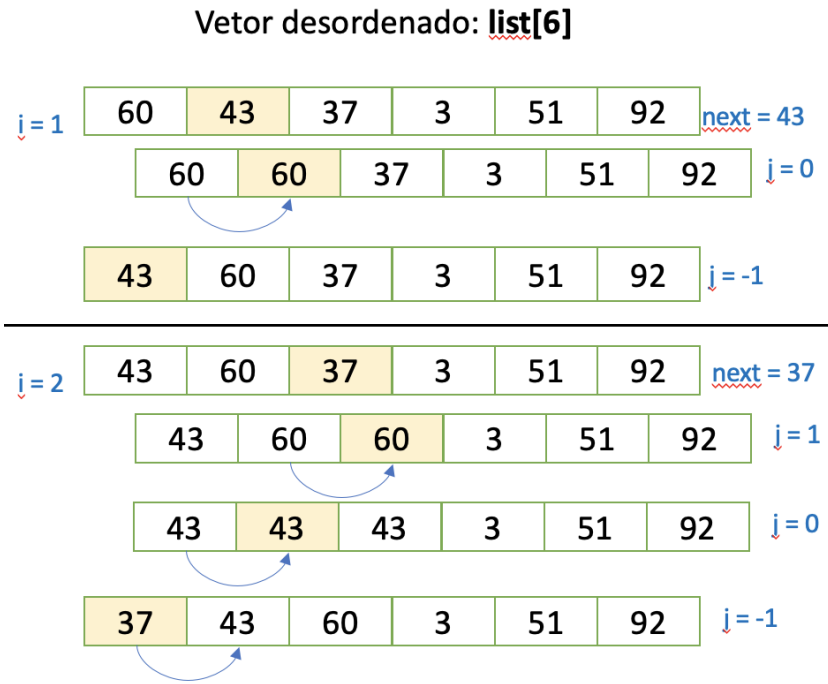
Considerações de sua aplicação

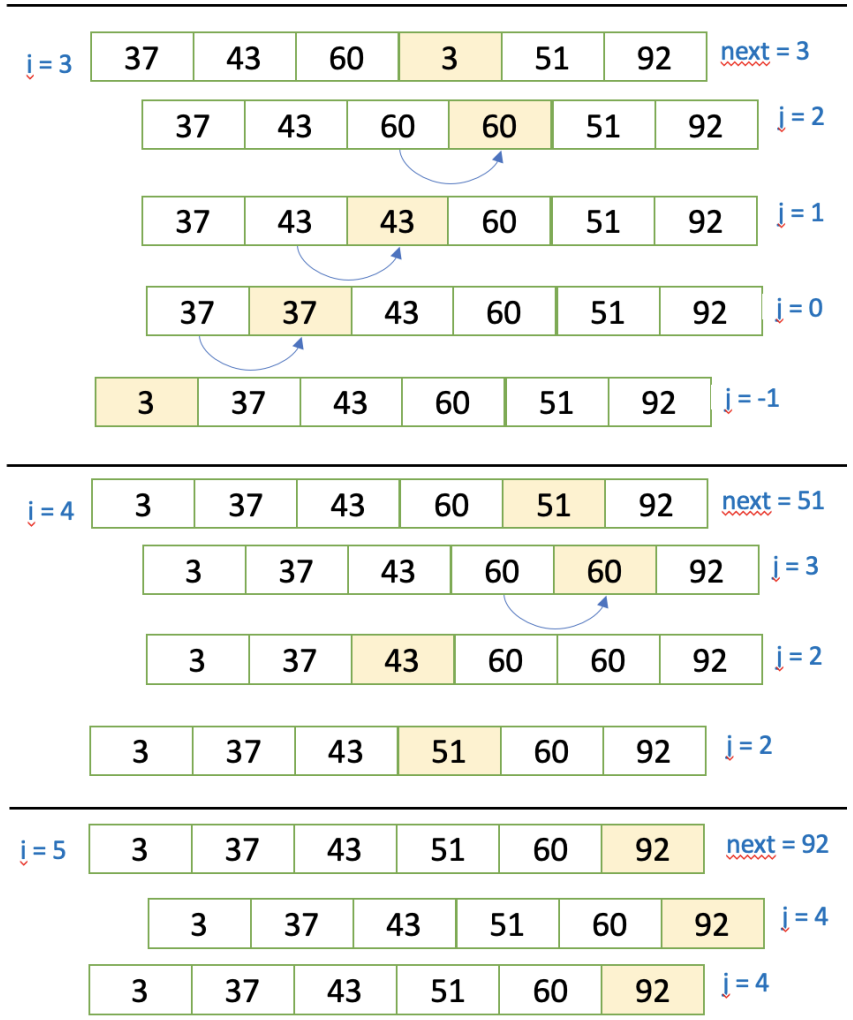
- Recursivo.
- A escolha do pivô afeta sua complexidade.
- Estável: não altera a ordem de chaves iguais

Inserção
Código


```
1  void insertionSort(int list[], int n)
2  {
3    int i, j;
4    int next;
5    for (i=1; i<n; i++) {
6      next= list[i];
7      for (j=i-1; j>=0&&next<list[j];j--)
8        list[j+1] = list[j];
9      list[j+1] = next;
10   }
11 }
```

Inserção
Algoritmo





Performance

- Melhor Caso: O(N).
- Pior Caso: O(N^2).

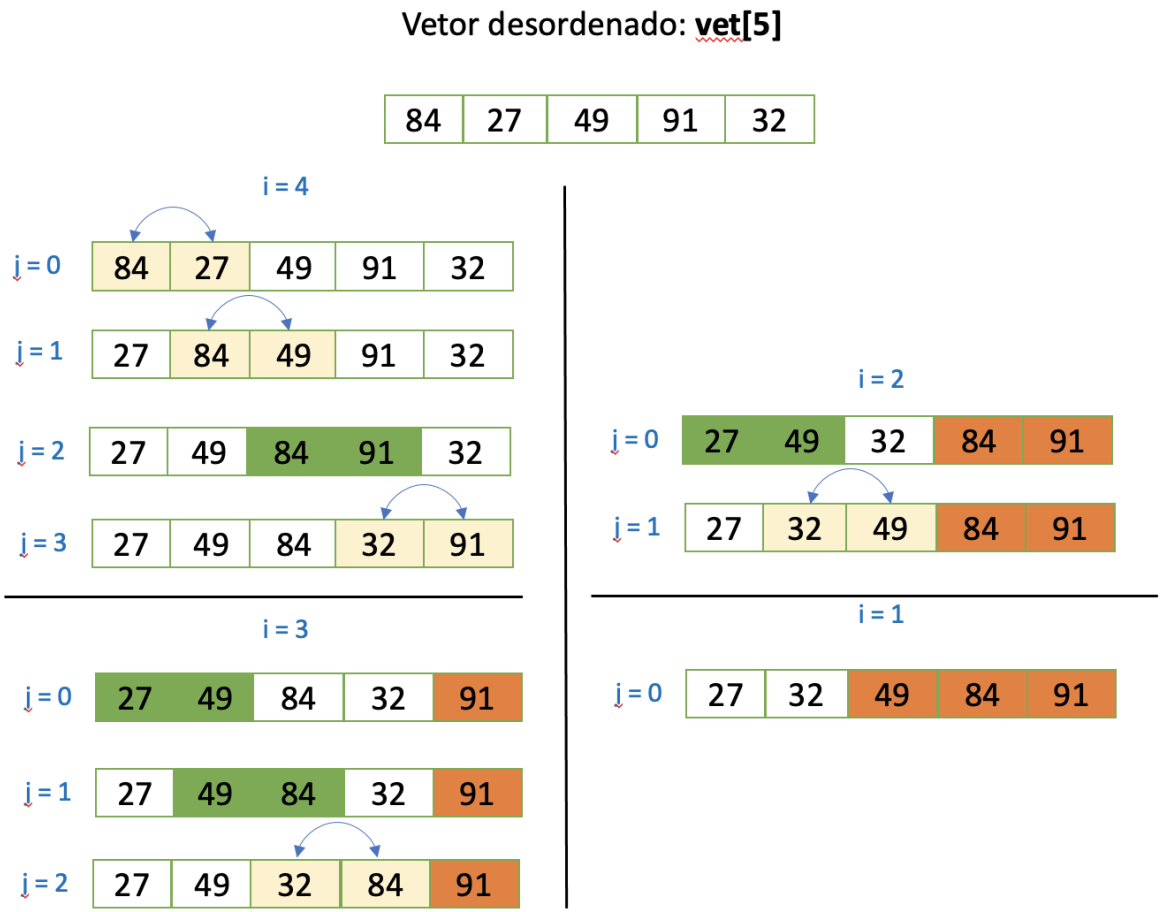
Considerações de sua aplicação

- Estável: não altera a ordem de chaves iguais

Bolha
Código


```
1  void bubbleSort(int vet[], int tam){
2      int i,j, temp;
3      for(i=tam-1; i>0; i--){
4          for(j=0; j < i; j++) //Faz trocas até posição i
5              if( vet[j] > vet[j+1] ){
6                  temp = vet[j];
7                  vet[j] = vet[j+1];
8                  vet[j+1] = temp;
9              }
10     }
11 }
```

Bolha
Algoritmo



Performance

- Melhor Caso: O(N).
- Pior Caso: O(N^2).

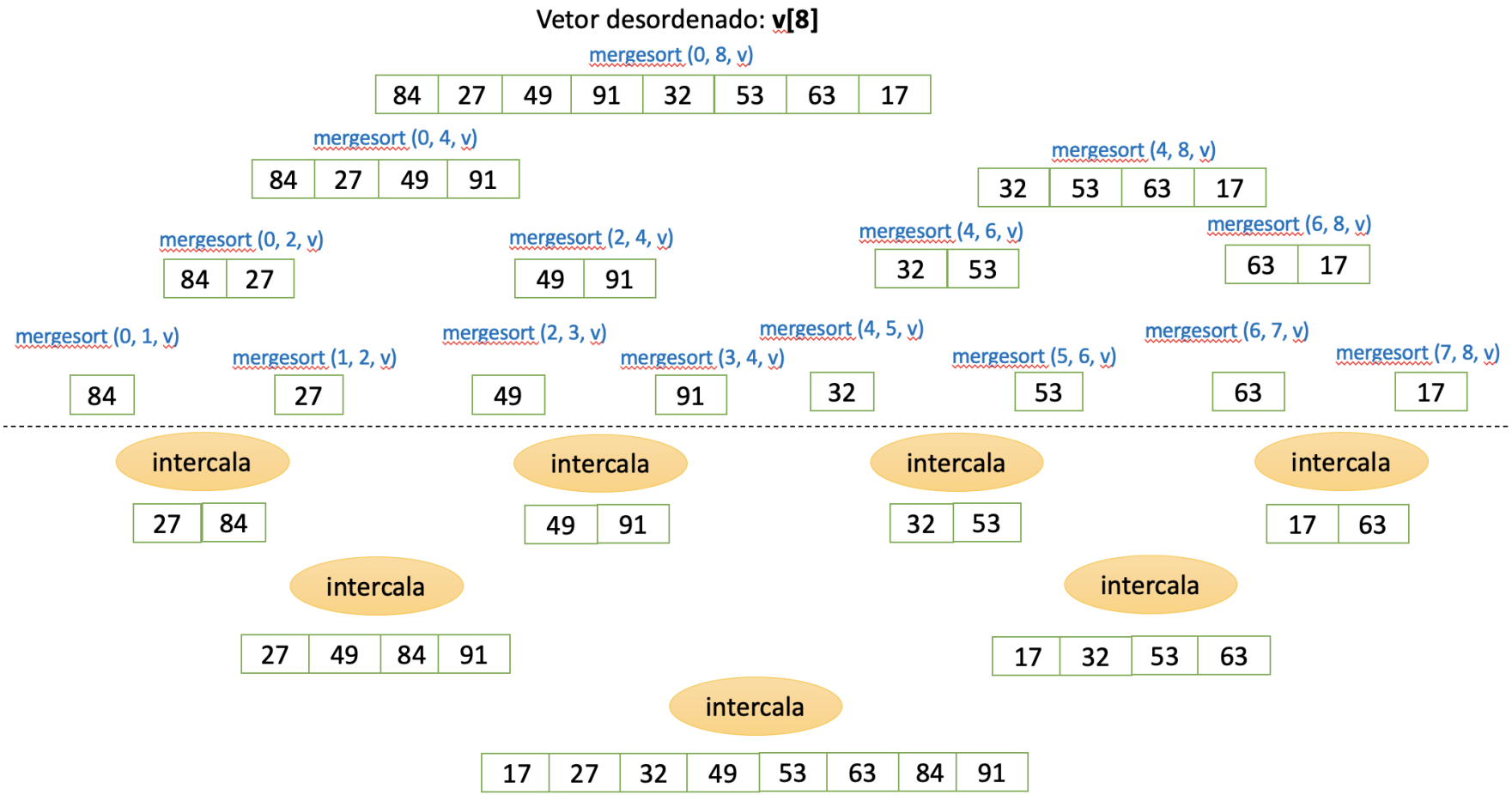
Intercalação
Código: Intercala


```
1  // A função recebe vetores crescentes v[p..q-1] e v[q..r-1]
2  // e rearranja v[p..r-1] em ordem crescente.
3  void intercala (int p, int q, int r, int v[])
4  {
5     int *w;
6     w = malloc ((r-p) * sizeof (int));
7     int i = p, j = q;
8     int k = 0;
9
10    while (i < q && j < r) {
11       if (v[i] <= v[j])  w[k++] = v[i++];
12       else  w[k++] = v[j++];
13    }
14    while (i < q)  w[k++] = v[i++];
15    while (j < r)  w[k++] = v[j++];
16    for (i = p; i < r; ++i)  v[i] = w[i-p];
17    free (w);
18 }
```

Intercalação
Código


```
1   // A função mergesort rearranja o vetor
2   // v[p..r-1] em ordem crescente.
3
4   void mergesort (int p, int r, int v[])
5   {
6      if (p < r-1) {
7         int q = (p + r)/2;
8         mergesort (p, q, v);
9         mergesort (q, r, v);
10        intercala (p, q, r, v);
11     }
12  }
```

Intercalação
Algoritmo



Performance

- Melhor Caso: O(N.LogN).
- Pior Caso: O(N.LogN).

Considerações de sua aplicação

- Recursivo.
- Usa um vetor temporário durante a ordenação.
- Estável: não altera a ordem de chaves iguais

Enunciado

Apresente o tempo de execução dos métodos de ordenação sobre o arquivo de população das cidades brasileiras - ordene pelo campo população 2010.

Dados

`http://blog.mds.gov.br/redesuas/lista-de-municipios-brasileiros/`