Organização e Recuperação da Informação

Métodos de ordenação

DC – UFSCar
Jander Moreira

Conteúdo

- Agenda
  - Introdução
  - Métodos clássicos para ordenação
    - Métodos eficientes
      - Quick sort
      - Merge sort
      - Radix sort

Introdução

- Ordenação em memória principal
  - Considerações
    - Complexidade de tempo
      - Desempenho dos algoritmos em função do número de itens
    - Complexidade de espaço
      - Uso de espaço adicional de memória em função do número de itens

Métodos eficientes

- Quicksort
  - Conceito
    - Abordagem "divisão e conquista"
      - Trocas de itens dentro de uma partição, criando a partir dela duas partições independentes
    - Ordenação
      - Considerar o arranjo todo como uma partição
      - Aplicar o procedimento de partição a cada partição existente que contenha mais que um elemento, criando a partir de cada uma, duas novas partições

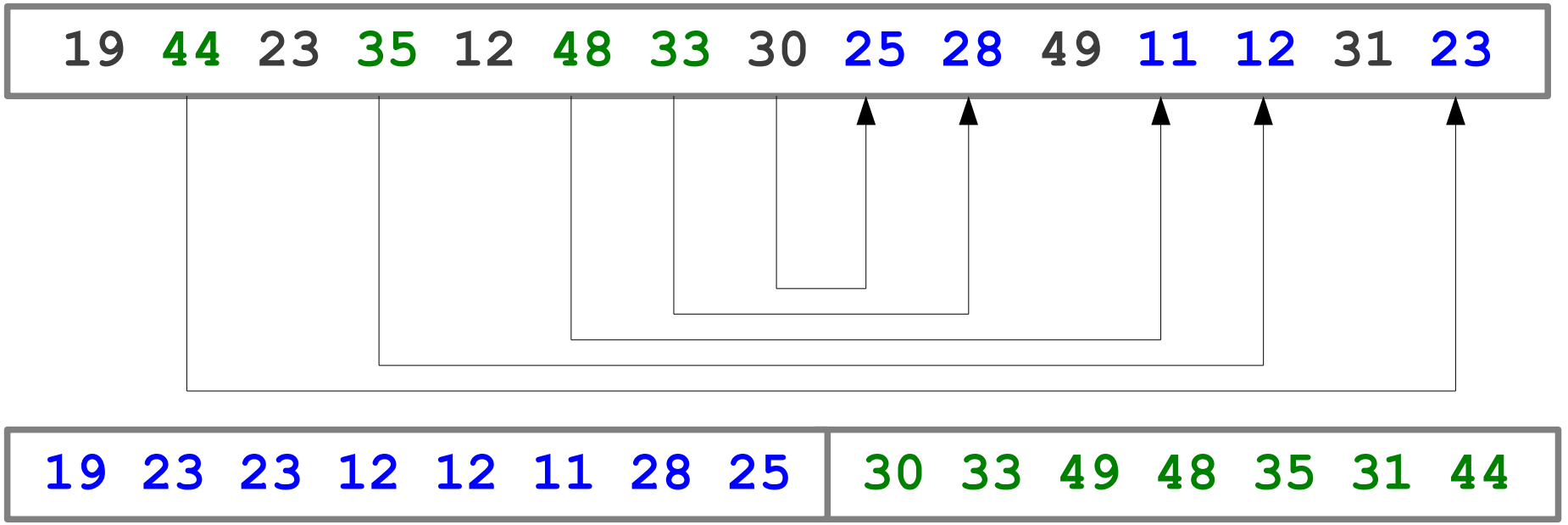
Métodos eficientes

```
partição(dados[]: inteiro, início, fim: inteiro, var esquerda, direita: inteiro)
{ particionamento dos dados entre as posições início e fim do arranjo }
    esquerda ← início
    direita ← fim
    pivô ← dados[(início + fim)/2] { elemento do meio }
    enquanto esquerda ≤ direita faça
        { procura um valor maior ou igual ao pivô }
        enquanto dados[esquerda] < pivô faça
            esquerda ← esquerda + 1
        fim-enquanto

        { procura um valor menor ou igual ao pivô }
        enquanto dados[direita] > pivô faça
            direita ← direita – 1
        fim-enquanto

        { faz a troca }
        se esquerda ≤ direita então
            auxiliar ← dados[esquerda]
            dados[esquerda] ← dados[direita]
            dados[direita] ← auxiliar
            direita ← direita – 1
            esquerda ← esquerda + 1
        fim-se
    fim-enquanto
```

Métodos eficientes

- Quicksort

pivô = 30



Métodos eficientes

```
quicksort(dados[]: inteiro, tamanho: inteiro)
   quickRecursivo(dados, 0, tamanho – 1)
```

Quicksort
Algoritmo

```
quickRecursivo(dados[]: inteiro, início, fim: inteiro)
   partição(dados, início, fim, esquerda, direita)

   { ordena primeira partição }
   se início < direita então
      quickRecursivo(dados, início, direita)
   fim-se

   { ordena segunda partição }
   se esquerda < fim então
      quickRecursivo(dados, esquerda, fim)
   fim-se
```

Quicksort
Recursão

Métodos eficientes

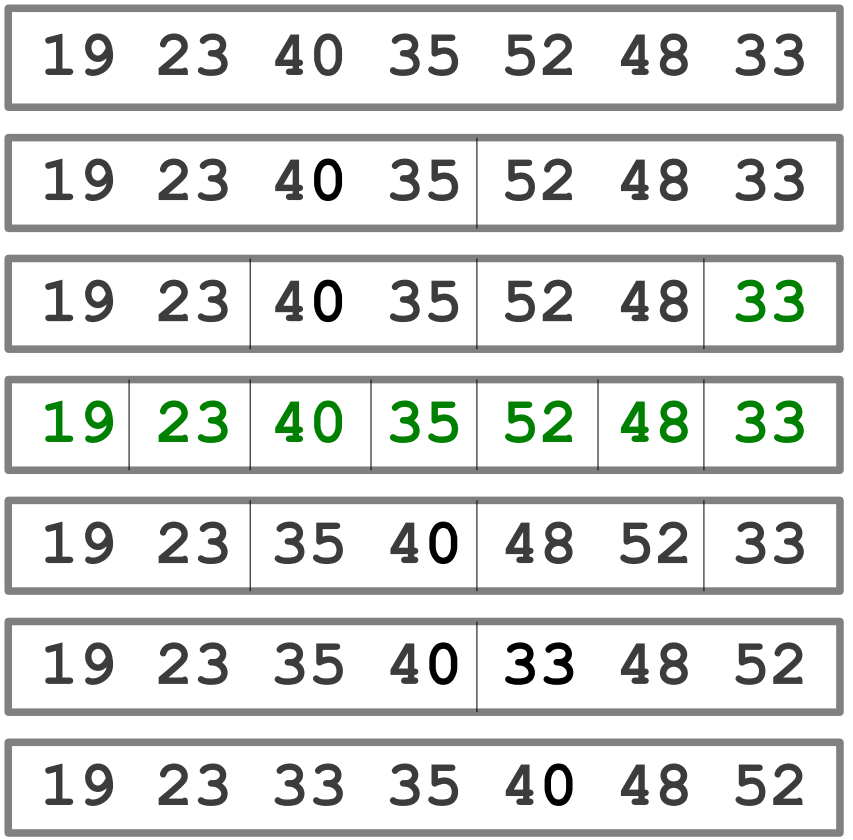
- Quicksort
  - Desempenho
    - Melhor caso
      - As partições criadas possuem o mesmo tamanho
      - O(n log n)
    - Pior caso
      - As partições criadas possuem tamanhos muito diferentes
      - $O(n^2)$
    - Caso médio
      - A maioria das partições criadas possuem tamanho equivalente
      - O(n log n)
  - Consumo de espaço (pilha ou chamadas recursivas)
    - O(log n) para o melhor caso
    - O(n) para o pior caso

Métodos eficientes

- Mergesort
  - Conceito
    - Abordagem
      - Fusão de listas ordenadas
      - Exige complexidade de espaço O(n)
    - Ordenação
      - Divisão do arranjo em sequências de 1 elemento, com fusão sucessiva das listas até que a fusão deixe apenas uma lista

Métodos eficientes

- Mergesort



Métodos eficientes

- Mergesort

```
mergesort(dados[]: inteiro)
   mergeRecursivo(dados, 0, n - 1)
```

```
mergeRecursivo(dados[]: inteiro, início, fim: inteiro)
   se início > fim então
      meio ← (início + fim)/2
      mergeRecursivo(dados, início, meio)
      mergeRecursivo(dados, meio + 1, fim)

      façaFusão(dados, início, fim)
   fim-se
```

Métodos eficientes

- Mergesort

```
façaFusão(dados[]: inteiro; início, fim: inteiro)
    meio ← (início + fim)/2

    i1 ← início
    i2 ← meio + 1
    destino = início
    enquanto destino < fim faça
        se dados[i1] > dados[i2] e i2 <= fim então
            dadosTemp[destino] ← dados[i1]
            i1 ← i1 + 1
        senão
            dadosTemp[destino] ← dados[i2]
            i2 ← i2 + 1
        fim-se
        destino ← destino + 1
    fim-enquanto
    para destino ← início até fim faça
        dados[destino] ← dadosTemp[destino]
    fim-para
```

Métodos eficientes

- Mergesort
  - Desempenho
    - O(n log n)
    - Consumo de espaço: O(n)

Métodos eficientes

- Radixsort
  - Conceito
    - Abordagem
      - Uso de uma base de representação para a chave, com ordenações sucessivas da parte menos significativa para a mais significativa, considerando um *radical* (raiz)
    - Ordenação
      - Divisão dos dados em listas para cada elemento de base, com concatenação dessas listas antes do novo elemento de base
      - Repetição até que todos os elementos de base tenham sido processados ou haja uma única lista

Métodos eficientes

- Radixsort

```
129 23 420 35 52 548 32 519
```

Unidades
base = 10^0

```
0: 420
1:
2: 52, 32
3: 23
4:
5: 35
6:
7:
8: 548
9: 129, 519
```

```
420 52 32 23 35 548 129 519
```

Métodos eficientes

- Radixsort

```
420 52 32 23 35 548 129 519
```

Dezenas
base = 10^1

```
0:
1: 519
2: 420, 23, 129
3: 32, 35
4: 548
5: 52
6:
7:
8:
9:
```

```
519 420 23 129 32 35 548 52
```

Métodos eficientes

- Radixsort

```
519 420 23 129 32 35 548 52
```

Centenas
base = 10^2

```
0: 23, 32, 35, 52
1: 129
2:
3:
4: 420
5: 519, 548
6:
7:
8:
9:
```

```
23 32 35 52 129 420 519 548
```

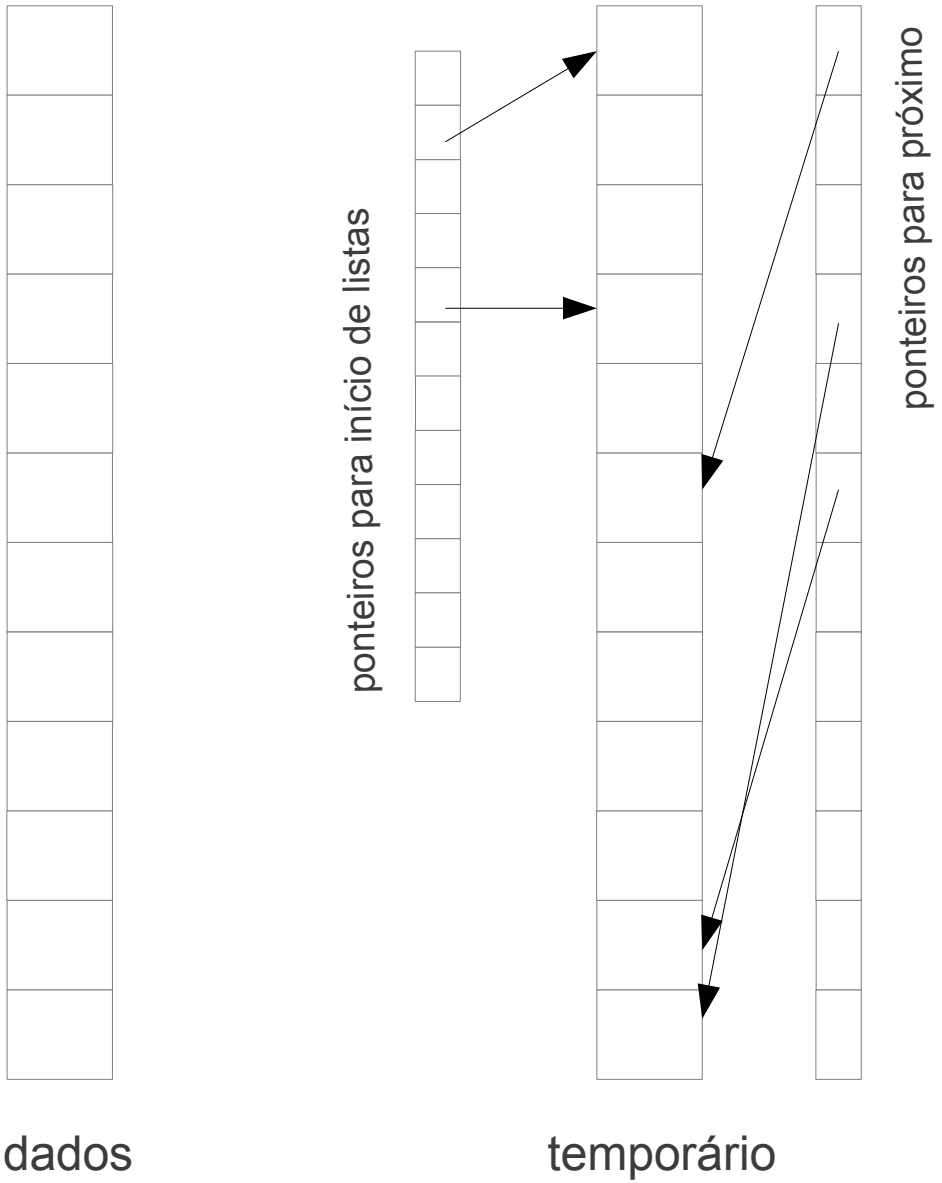
Métodos eficientes

- Radixsort

```
radixsort(dados[]: inteiro; int tamanho)
    passos ← maior_radical
    base ← 1 { unidades }
    passo ← 0
    enquanto passo < passos faça
        { distribuição }
        passos ← 1
        para i ← 0 até tamanho – 1 faça
            índice ← (dados[i] / base) % 10
            insira(lista[índice], dados[i])
            maxRadical(passos, dados[i]) { redefine }
        fim-para

        { concatenação }
        pos ← 0
        para i ← 0 até 9 faça
            copia(lista[i], dados, pos)
        fim-para

        { próxima base }
        base ← base * 10
    fim-enquanto
```

Métodos eficientes

- Radixsort



Métodos eficientes

- Radixsort
  - Desempenho
    - O(nk)
      - k: número de passos (comprimento da cadeia chave)
  - Espaço
    - O(n) para dados
    - O(n + s)
      - s: número de radicais
      - para ponteiros para próximo e área de dados

Leitura

- Leituras
  - Moreira, J. Ordenação (apostila)
  - Drozdek, Capítulo 9, seções 9.1 e 9.3