Análise da Complexidade de Algoritmos Recursivos IV

Joaquim Madeira

02/04/2020

Sumário

- Recap
- Ordenação por fusão: o Algoritmo Mergesort – Análise da Complexidade
- Ordenação por partição: o Algoritmo Quicksort
- Sugestões de leitura

Recapitulação

The Master Theorem

- Dada uma recorrência, para $n = b^k$, $k \geq 1$

$$T(1) = c \qquad e \qquad T(n) = a\,T(n / b) + f(n)$$

com $a \geq 1$, $b \geq 2$, $c \geq 0$

- Se $f(n) = \Theta(n^d)$, com $d \geq 0$, então

$$T(n) = \Theta(n^d), \text{ se } a < b^d$$

$$T(n) = \Theta(n^d \log n), \text{ se } a = b^d$$

$$T(n) = \Theta(n^{\log_b a}), \text{ se } a > b^d$$

The Smoothness Rule

- Seja $T(n)$ eventualmente não decrescente.
- Seja $g(n)$ uma função suave ("smooth function").
- Se $T(n) = \Theta(g(n))$ para valores de n que são potências de b, $b \geq 2$.
- Então $T(n) = \Theta(f(n))$, para qualquer n.
- Resultados análogos para $O(n)$ e $\Omega(n)$!!
- Boas notícias !!

Multiplicar números inteiros muito longos

- Como fazer?
  - E.g., mais do que 100 algarismos

- Muito maiores do que o maior inteiro representável !

- Usar o algoritmo que aprendemos na escola ?

- Divide-and-Conquer ?

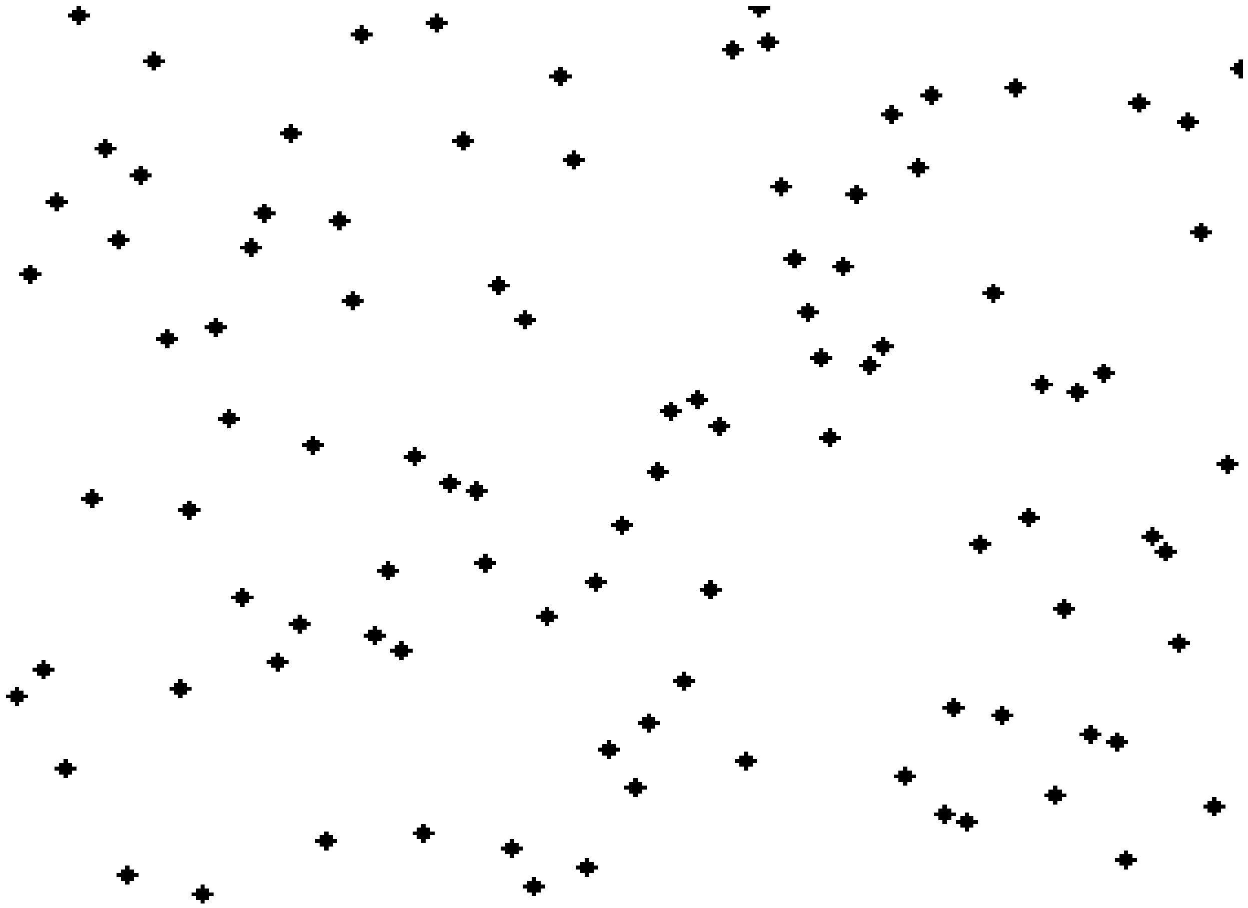
Quantas multplicações algarismo a algarismo?

- Algoritmo da escola primária : $\Theta(n^2)$
- Multiplicação por partição recursiva – 1ª ideia :
  - M(n) = 4 M (n / 2) $\Theta(n^2)$
- Multiplicação por partição recursiva – Alg. de Karatsuba :
  - M(n) = 3 M (n / 2)

$$\Theta(n^{\log_2 3}) = \Theta(n^{1.585}) \text{ !!}$$

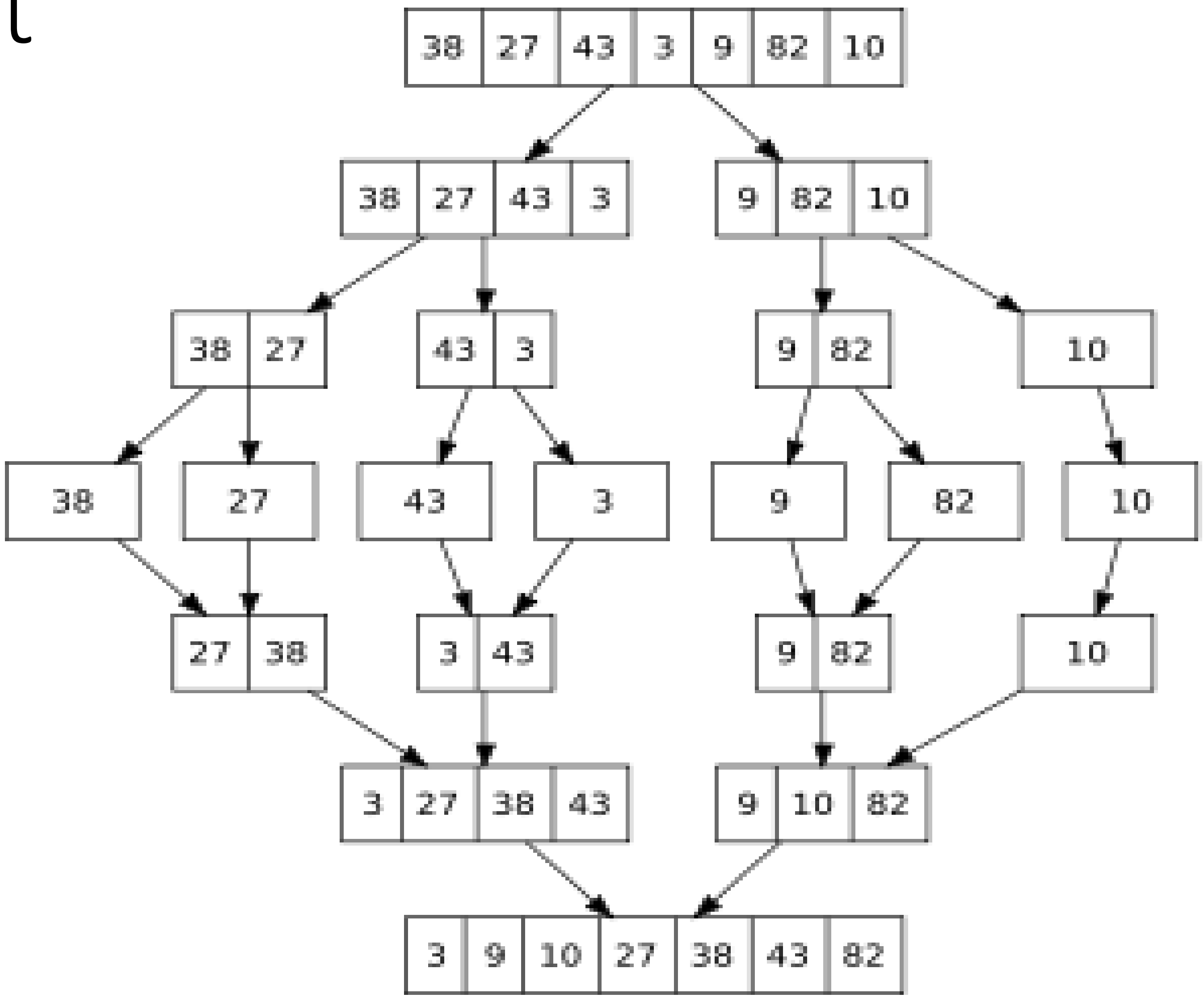
Mergesort
– Ordenação por Fusão

Mergesort



[Wikipedia]

Mergesort



SUBDIVISÃO

FUSÃO

[Wikipedia]

Tarefa: associar a cada seta um rótulo que identifica a sequência pela qual as chamadas são executadas

Mergesort

```c
// mergeSort(A, tmpA, 0, n - 1);

void mergeSort(int* A, int* tmpA, int left, int right) {
  // Mais do que 1 elemento ?

  if (left < right) {
    int center = (left + right) / 2;

    mergeSort(A, tmpA, left, center);
    mergeSort(A, tmpA, center + 1, right);
    merge(A, tmpA, left, center + 1, right);
  }
}
```

Mergesort

```
void merge(int* A, int* tmpA, int lPos, int rPos, int rEnd) {
  int lEnd = rPos - 1;
  int tmpPos = lPos;
  int nElements = rEnd - lPos + 1;

  // COMPARAR O 1o ELEMENTO DE CADA METADE
  // E COPIAR ORDENADAMENTE PARA O ARRAY TEMPORÁRIO

  while (lPos <= lEnd && rPos <= rEnd) {
    if (A[lPos] <= A[rPos])
      tmpA[tmpPos++] = A[lPos++];
    else
      tmpA[tmpPos++] = A[rPos++];
  }
```

Mergesort

```
    // SOBRA, PELO MENOS, 1 ELEMENTO NUMA DAS METADES

    while (lPos <= lEnd) {⋯
    }

    while (rPos < rEnd) {⋯
    }

    // COPIAR DE VOLTA PARA O ARRAY ORIGINAL

    for (int i = 0; i < nElements; i++, rEnd--) {
        A[rEnd] = tmpA[rEnd];
    }
}
```

Eficiência

- Todas as comparações são feitas pela função de fusão
- $C_{merge}(n)$: nº de comparações efetuadas para fundir 2 sub-arrays ordenados, usando um array auxiliar
- Caso particular : $n = 2^k$

$$C(1) = 0$$

$$C(n) = 2 \times C(n / 2) + C_{merge}(n)$$

- $C_{merge}(n) = ?$

Eficiência – $C_{merge}(n)$ – Melhor Caso

- Todos elementos de um dos sub-arrays são copiados primeiro
- Apenas $n / 2$ comparações para fazer isso !!
- $C(n) = 2 \times C(n / 2) + n / 2$
- Teorema Mestre : $\Theta(n \log n)$
- Construir um exemplo !

Eficiência – $C_{merge}(n)$ – Pior Caso

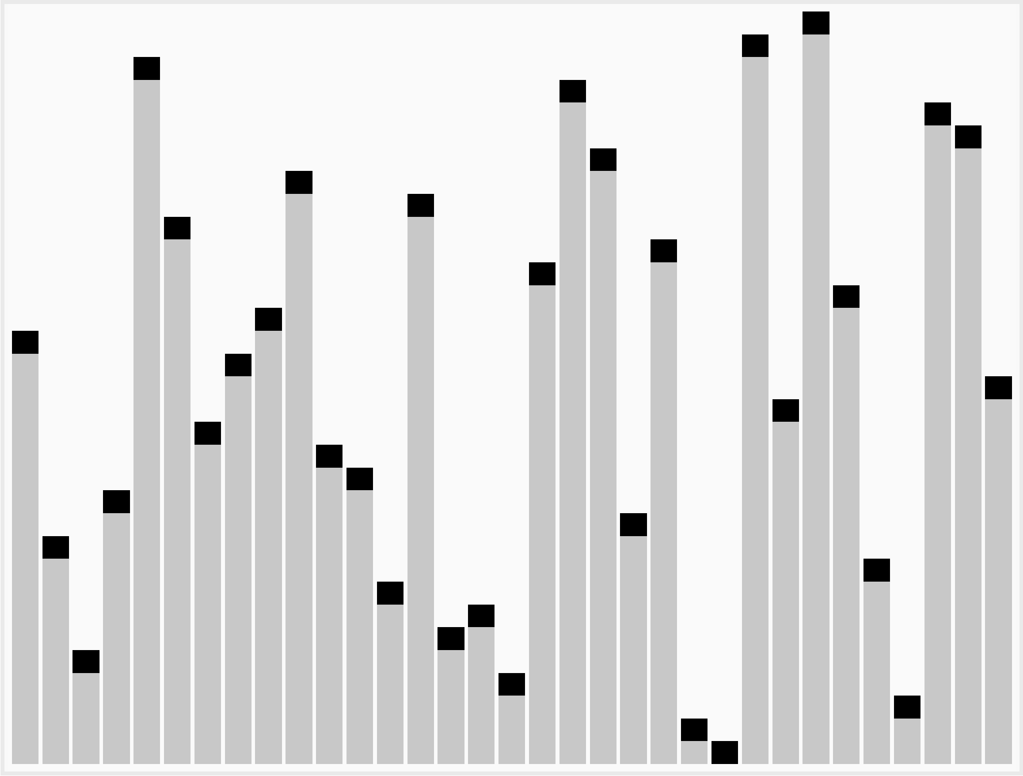
- Os elementos de um dos sub-arrays são copiados de modo intercalado, um a um !
- Necessárias $(n - 1)$ comparações !
- $C(n) = 2 \times C(n / 2) + (n - 1)$
- Teorema Mestre : $\Theta(n \log n)$
- Construir um exemplo !

Eficiência – $C_{merge}(n)$ – Caso Médio

- Podemos assumir que ocorre o nº médio de comparações
- ~ 3 x n / 4 comparações
- C(n) = 2 x C(n / 2) + 3 x n / 4
- Teorema Mestre : Θ(n log n)
- Construir um exemplo !

Quicksort
– Ordenação por Partição

Quicksort



[Wikipedia]

Quicksort

- Ordenar o array de modo recursivo, sem usar memória adicional
- Particionar o conjunto de elementos, trocando de posição, se necessário
- Com base no valor de um elemento pivot

Quicksort

- Escolher o valor do element pivot

- Particionar o array
- Elementos da 1ª partição são menores ou iguais do que o pivot
- Elementos da 2ª partição são maiores ou iguais do que o pivot

- Ordenar de modo recursivo a 1ª partição e a 2ª partição

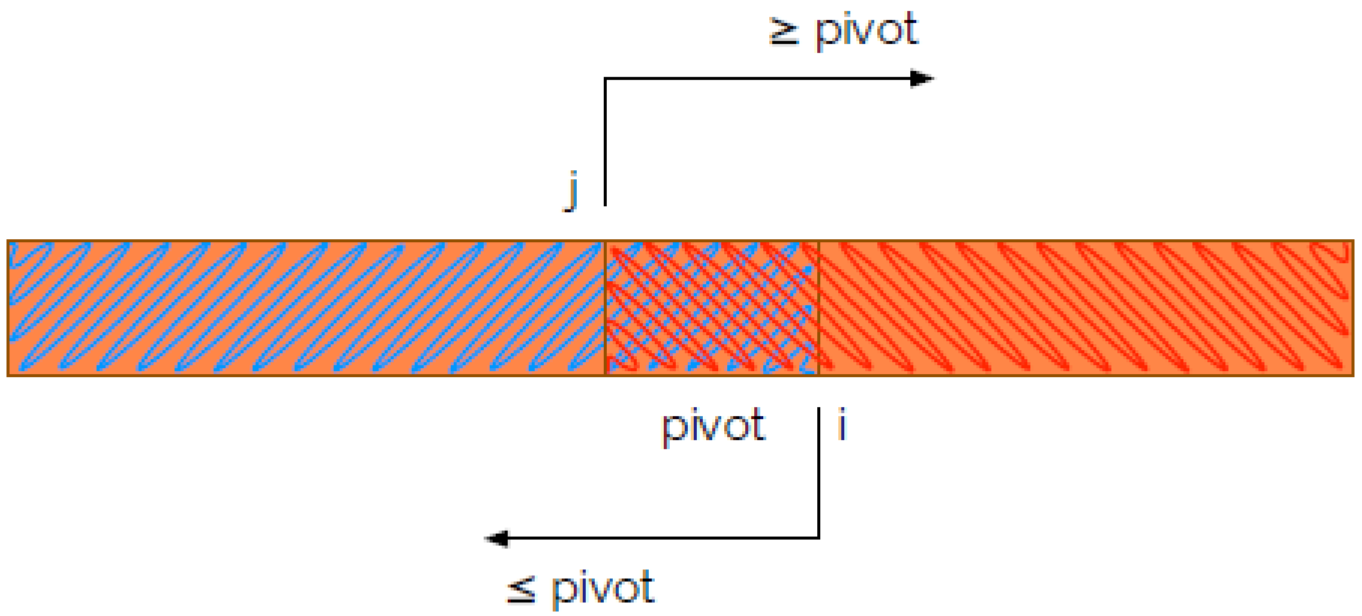
Questões

- Como esolher o pivot ?
  - O elemento do meio? O 1º elemento? O último elemento?
  - O elemento mediano dos 3 anteriores?
  - Um elemento escolhido de modo aleatório?

- Como particionar ?

- Atenção : pode surgir uma terceira partição central, cujos elementos têm o valor do pivot !

Partições



[Rui Lopes]

Exemplo

- Vamos começar um exemplo !!
- Ordenar: 6 5 4 3 2 1 -> pivot = 4

…

- Terminem o exemplo em casa ☺
- O que tem este exemplo de particular ?

Código ?

- Vamos começar !!

1ª versão

```c
void quicksort(int* A, int left, int right) {
  // Casos de base
  if (left >= right) return;

  // Caso recursivo
  // FASE DE PARTIÇÃO
  int pivot = (left + right) / 2;
  int i = left;
  int j = right;

  do {
    while (A[i] < A[pivot]) i++;
    while (A[j] > A[pivot]) j--;

    if (i <= j) {
      trocar(&A[i], &A[j]);
      i++;
      j--;
    }
  } while (i <= j);

  // Chamadas recursivas
  quicksort(A, left, j);
  quicksort(A, i, right);
}
```

Eficiência

- Todas as comparações são feitas na fase de partição !!

- Qual é a recorrência ?
- O caso geral é mais difícil de desenvolver e analisar !!

- $O(n \log n)$ para o melhor caso e o caso médio
- MAS $O(n^2)$ para o pior caso !!
    - Muito raro, se escolhermos “bem” o pivot

Sugestões de leitura

Sugestões de leitura

- A. Levitin, Introduction to the Design and Analysis of Algorithms, 3^{rd} Edition, 2012
  - Capítulo 5: secção 5.1
  - Capítulo 5: secção 5.2
  - Apêndice B