BC1424
Algoritmos e Estruturas de Dados I

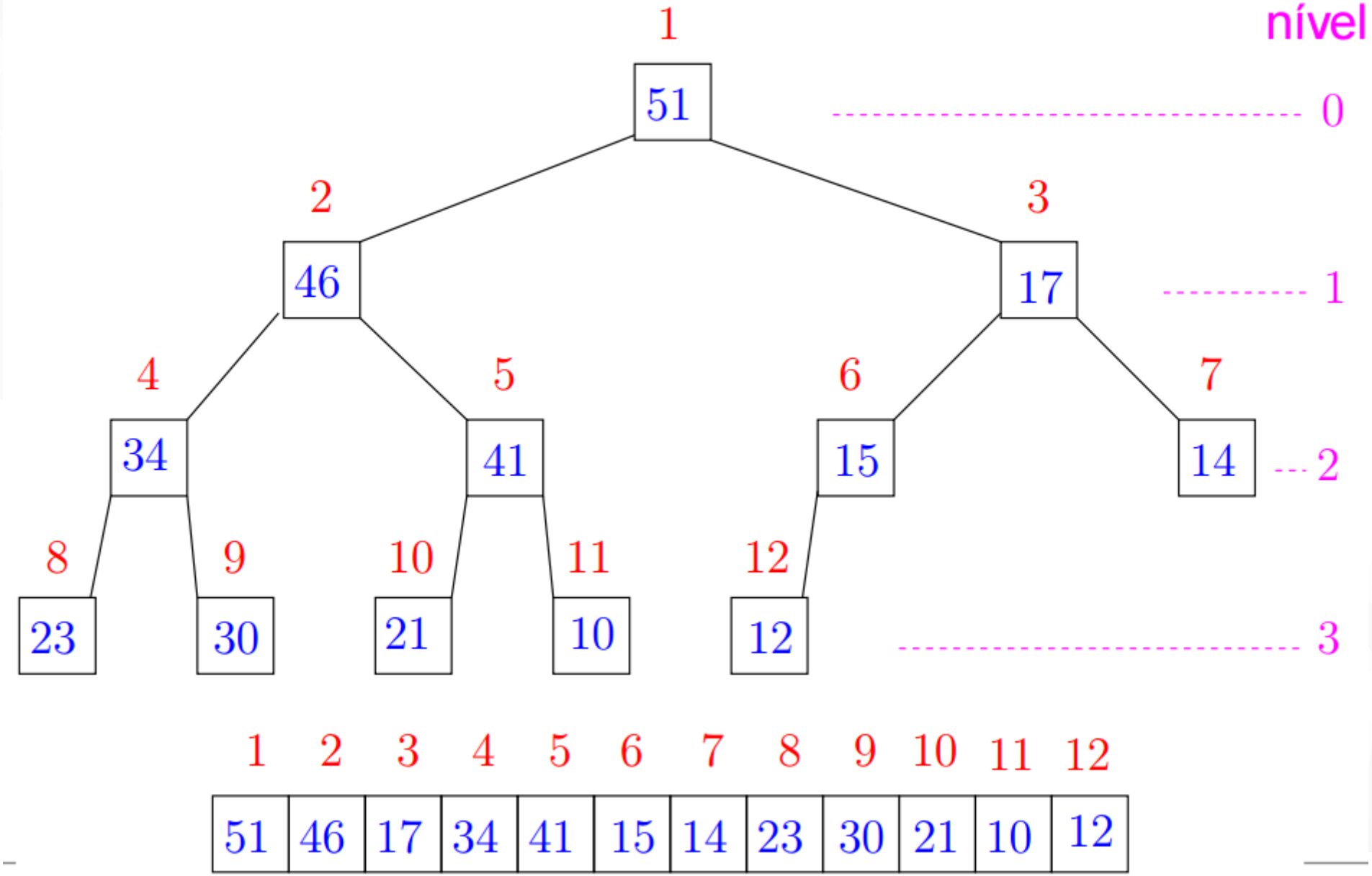
Aula 14:
Métodos eficientes de ordenação
(Quick Sort)

Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

1Q-2015

Da última aula

Max-heap: representação como vetor



Heap Sort

```c
void MaxHeapify (int A[], int m, int i) {
    int e, d, maior, aux;

    e = 2*i;
    d = 2*i+1;

    if (e<=m && A[e]>A[i])
        maior = e;
    else
        maior = i;

    if (d<=m && A[d]>A[maior])
        maior = d;

    if (maior!=i) {
        aux = A[maior];
        A[maior] = A[i];
        A[i] = aux;

        MaxHeapify(A, m, maior);
    }
}
```

```c
void BuildMaxHeap(int A[], int n) {
    int i;
    for (i=n/2; i>=1; i--)
        MaxHeapify (A, n, i);
}
```

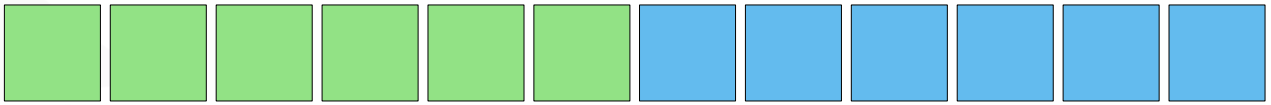
```c
void HeapSort(int A[], int n) {
    int i, m, aux;

    BuildMaxHeap(A, n);
    m = n;

    for (i=n; i>=2; i--) {
        aux = A[i];
        A[i] = A[1];
        A[1] = aux;

        m = m-1;
        MaxHeapify (A, m, 1);
    }
}
```

Consumo de tempo é proporcional a altura da árvore = $lg(m)$

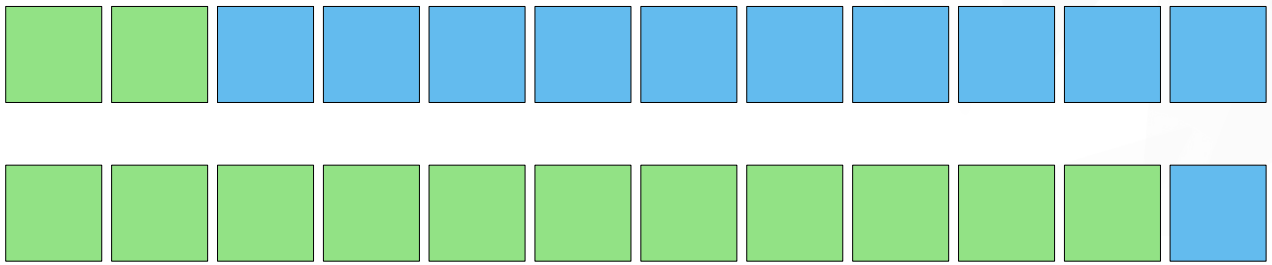
O problema da separação de elementos

Separar elementos em um vetor

- Deseja-se rearranjar um vetor `A[p..r]` de modo que os elementos pequenos fiquem todos do lado esquerdo e os grandes fiquem todos do lado direito.



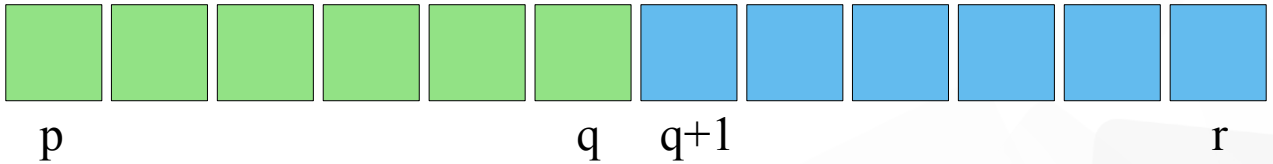
- Idealmente, deseja-se que os 2 lados tivessem aproximadamente o mesmo número de elementos (i.e. divisão/separação equilibrada.



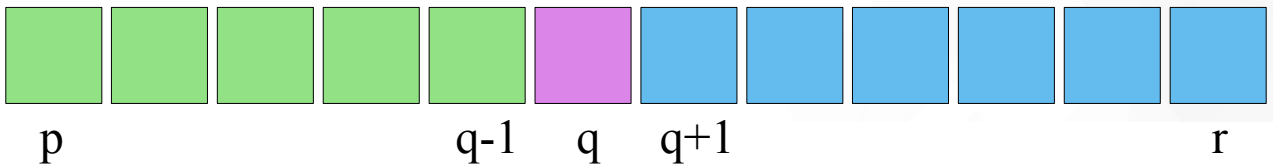
Separar elementos em um vetor

A dificuldade está em construir um algoritmo que resolva o problema de **maneira rápida (*O(n)*) e não use um vetor auxiliar**.

- Formulação 1: $A[p..q] \leq A[q+1..r]$



- Formulação 2: $A[p..q-1] \leq A[q] \leq A[q+1..r]$

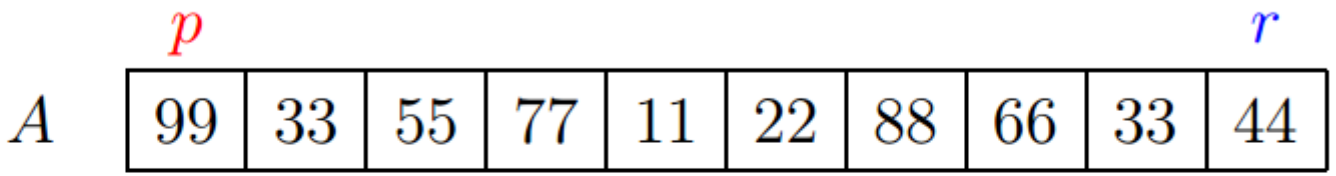


Partição: Separar elementos em um vetor

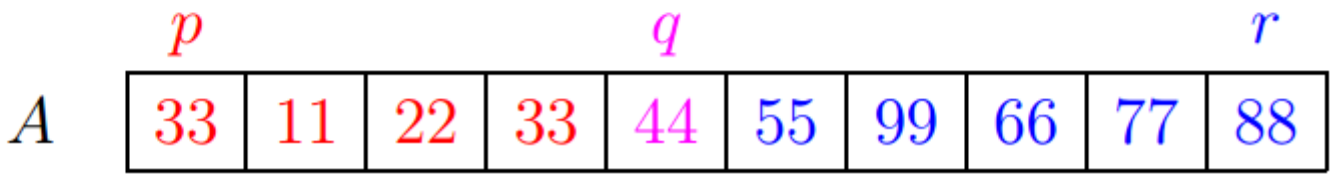
Problema: Rearranjar um dado vetor `A[p..r]` e devolver um índice `q`, p≤q≤r, tais que

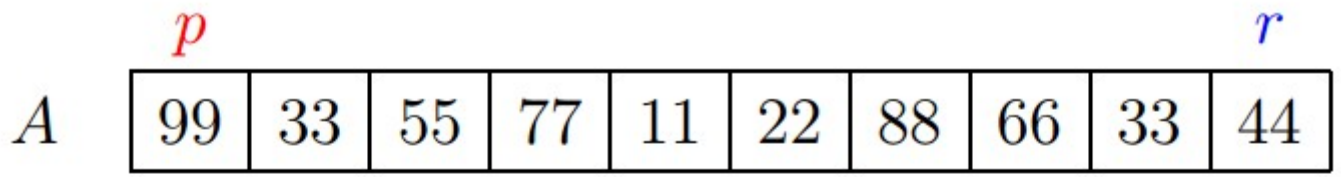
$$A[p..q-1] \leq A[q] \leq A[q+1..r]$$

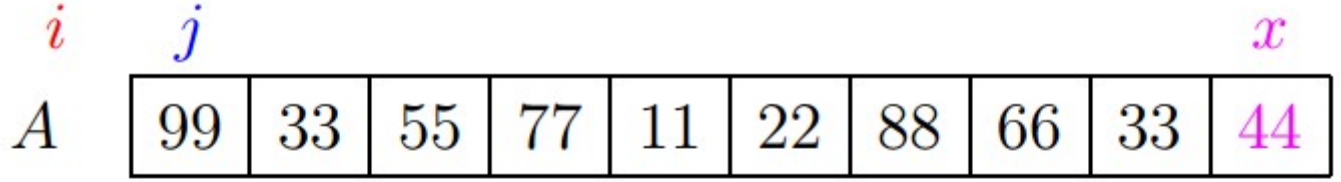
Entra:

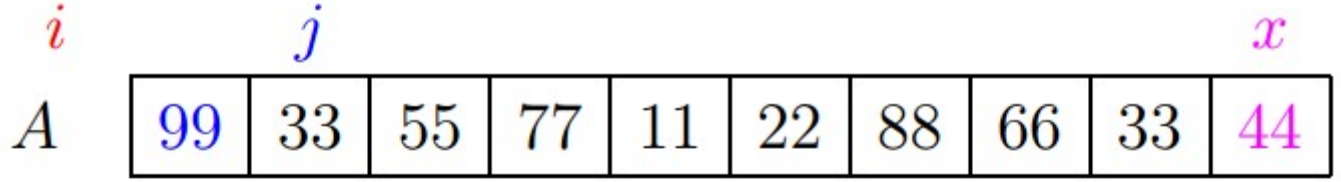


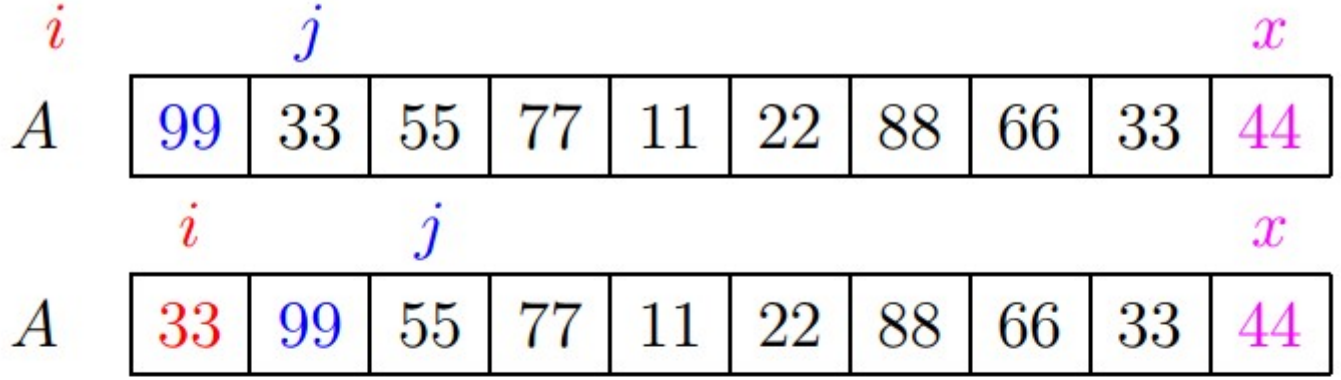
Sai:

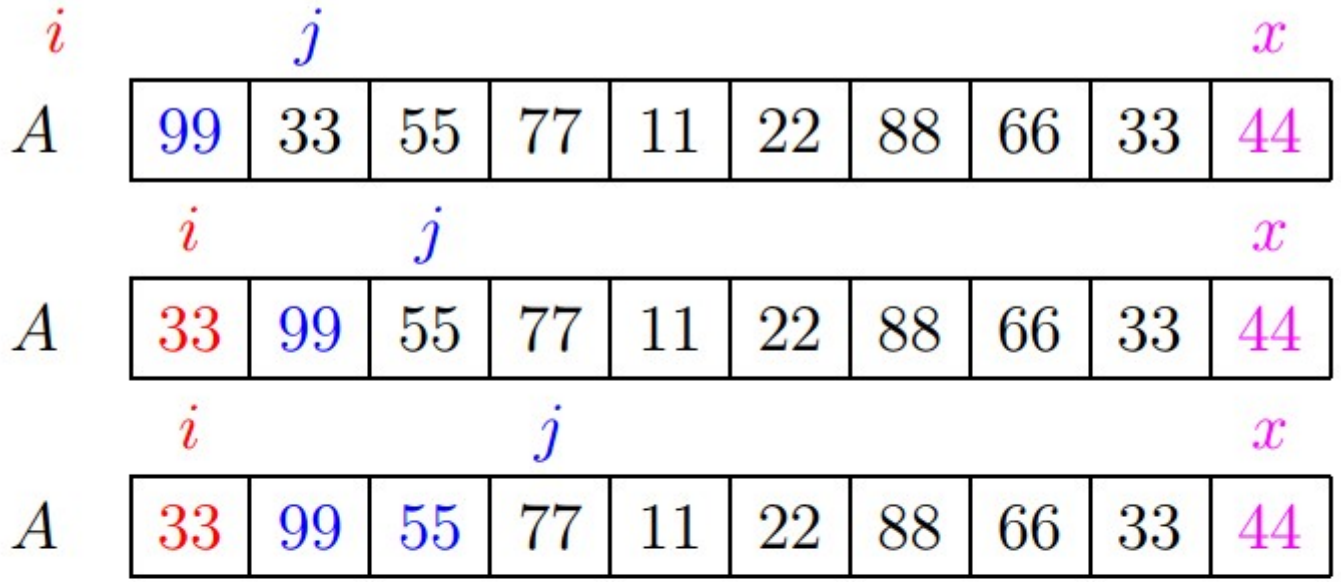


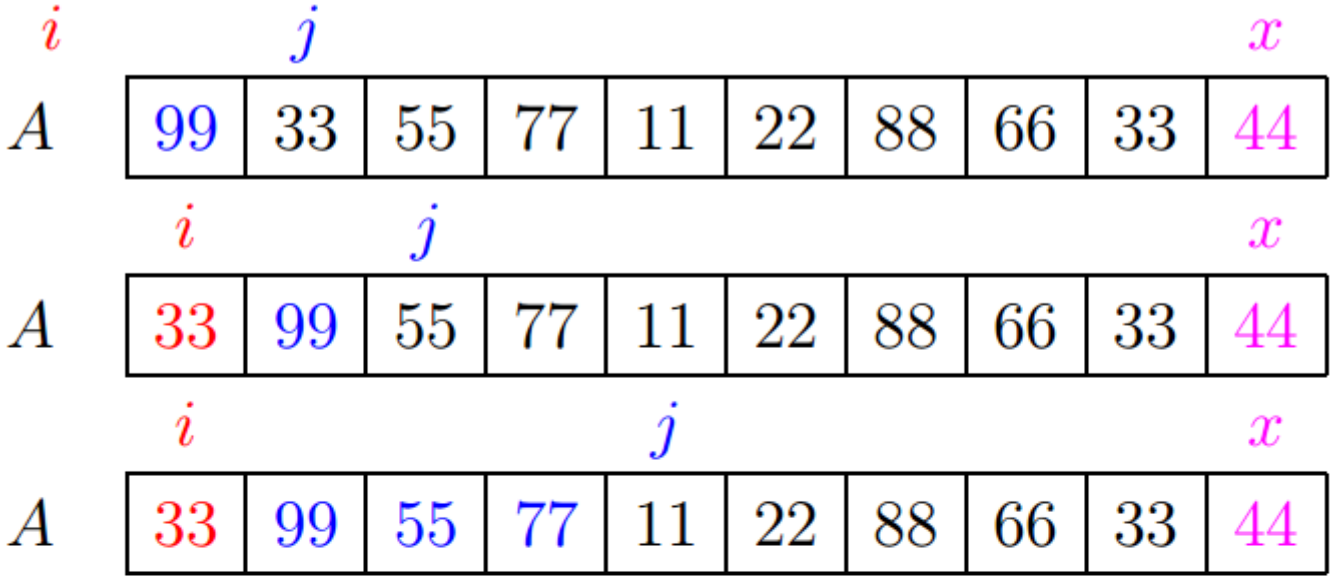

p
r
A
99
33
55
77
11
22
88
66
33
44

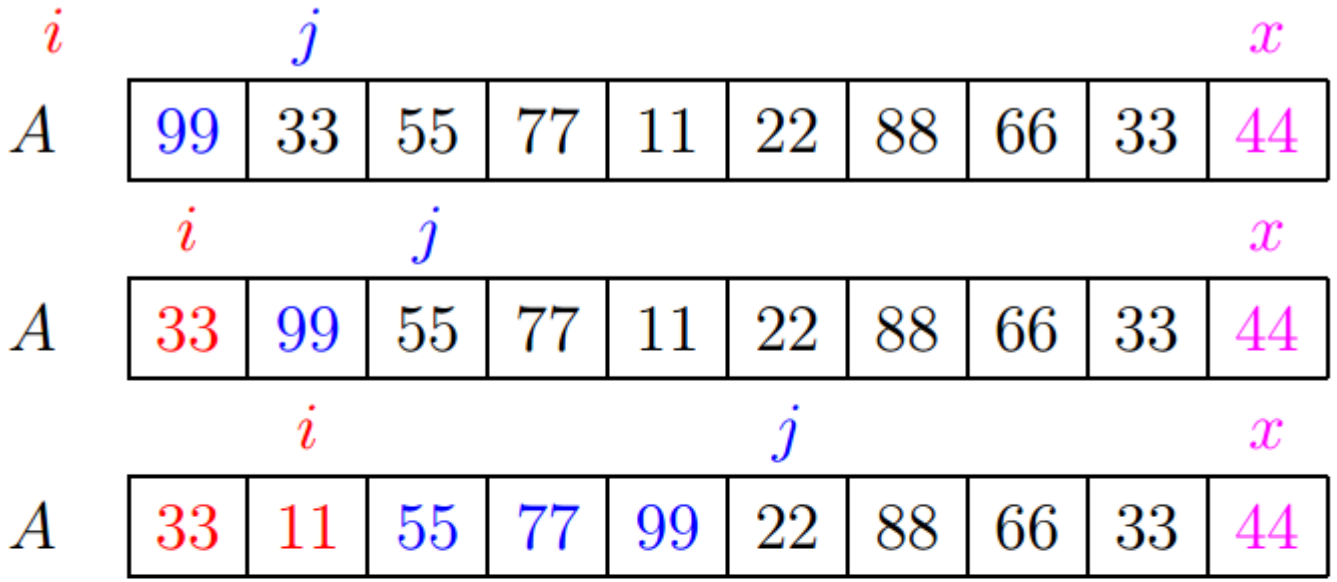

i
j
x
A
99
33
55
77
11
22
88
66
33
44

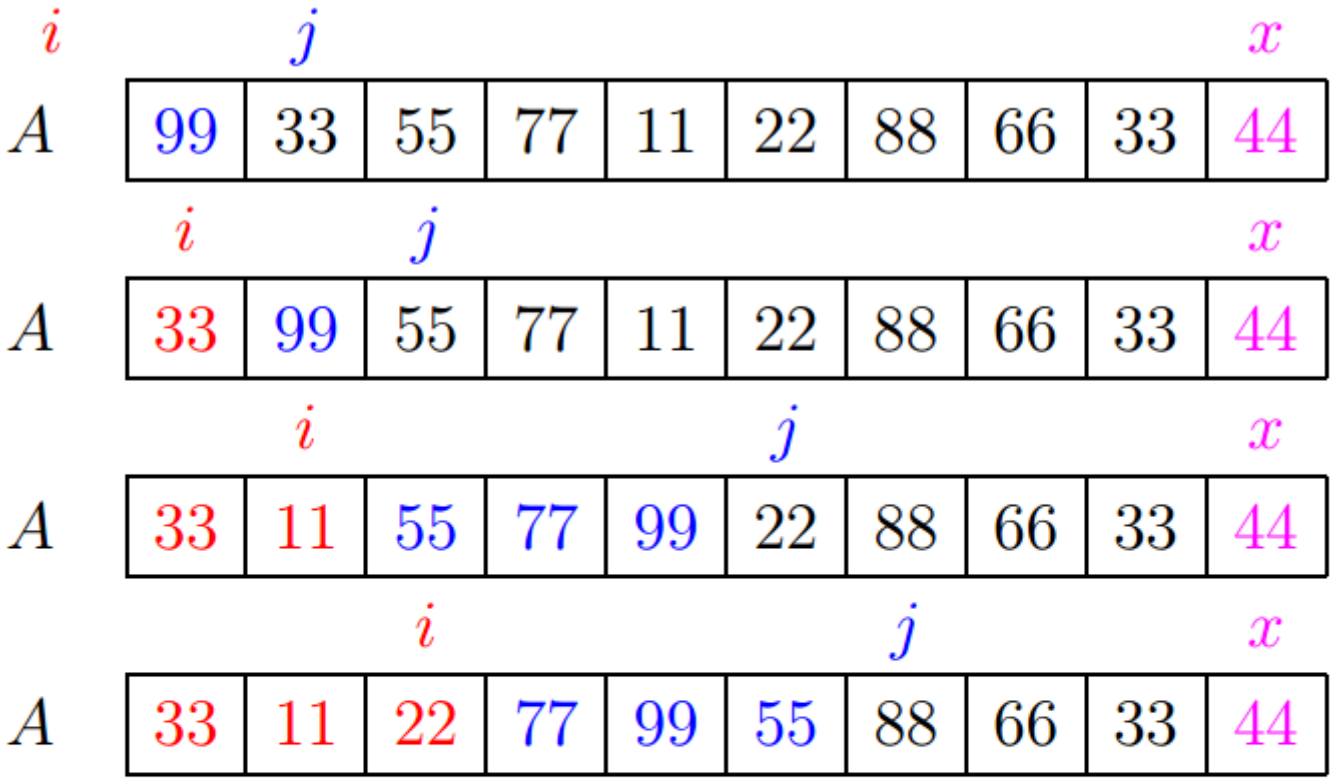

i
j
x
A
99
33
55
77
11
22
88
66
33
44

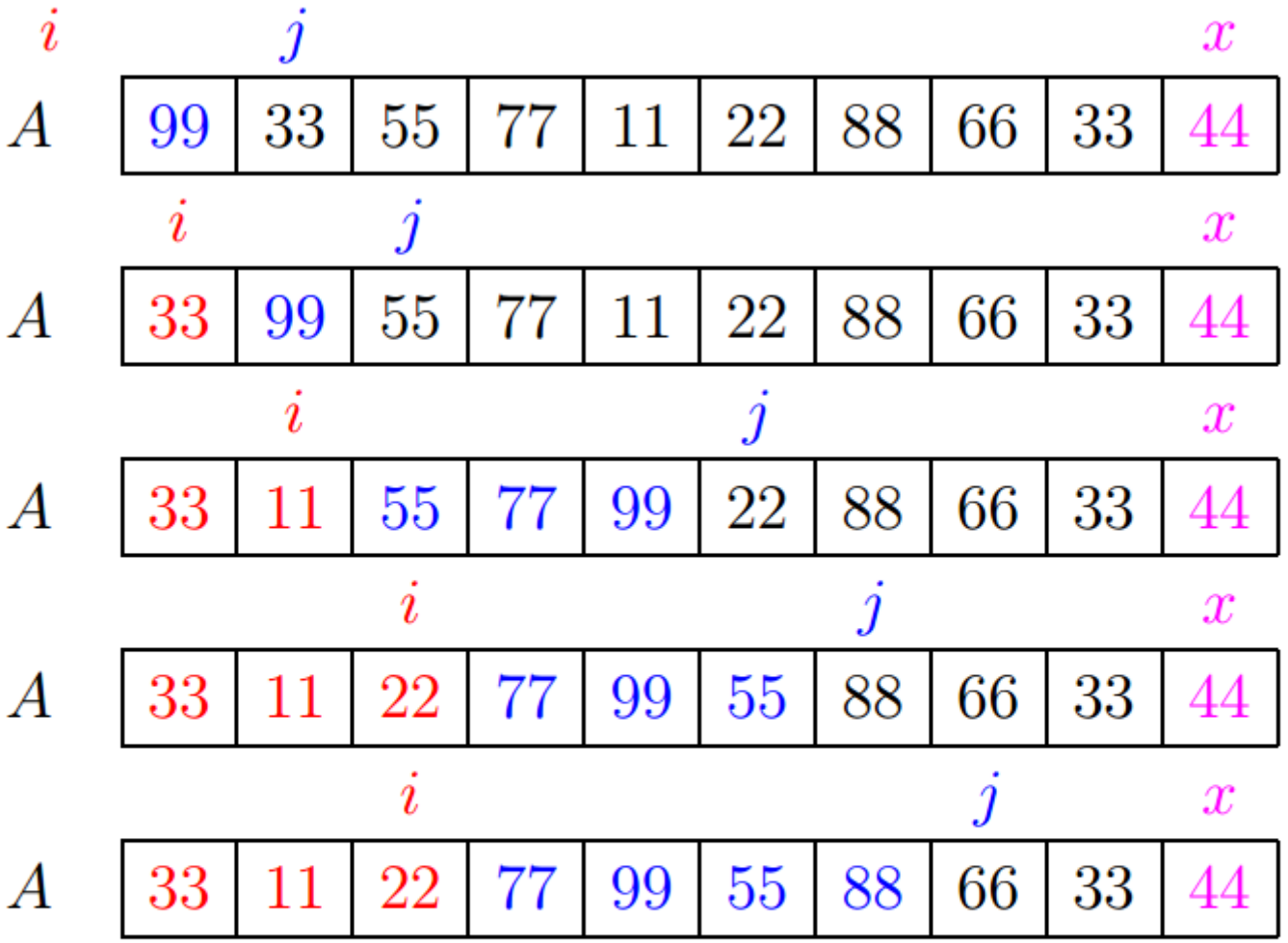

i
j
x
A
99
33
55
77
11
22
88
66
33
44
i
j
x
A
33
99
55
77
11
22
88
66
33
44

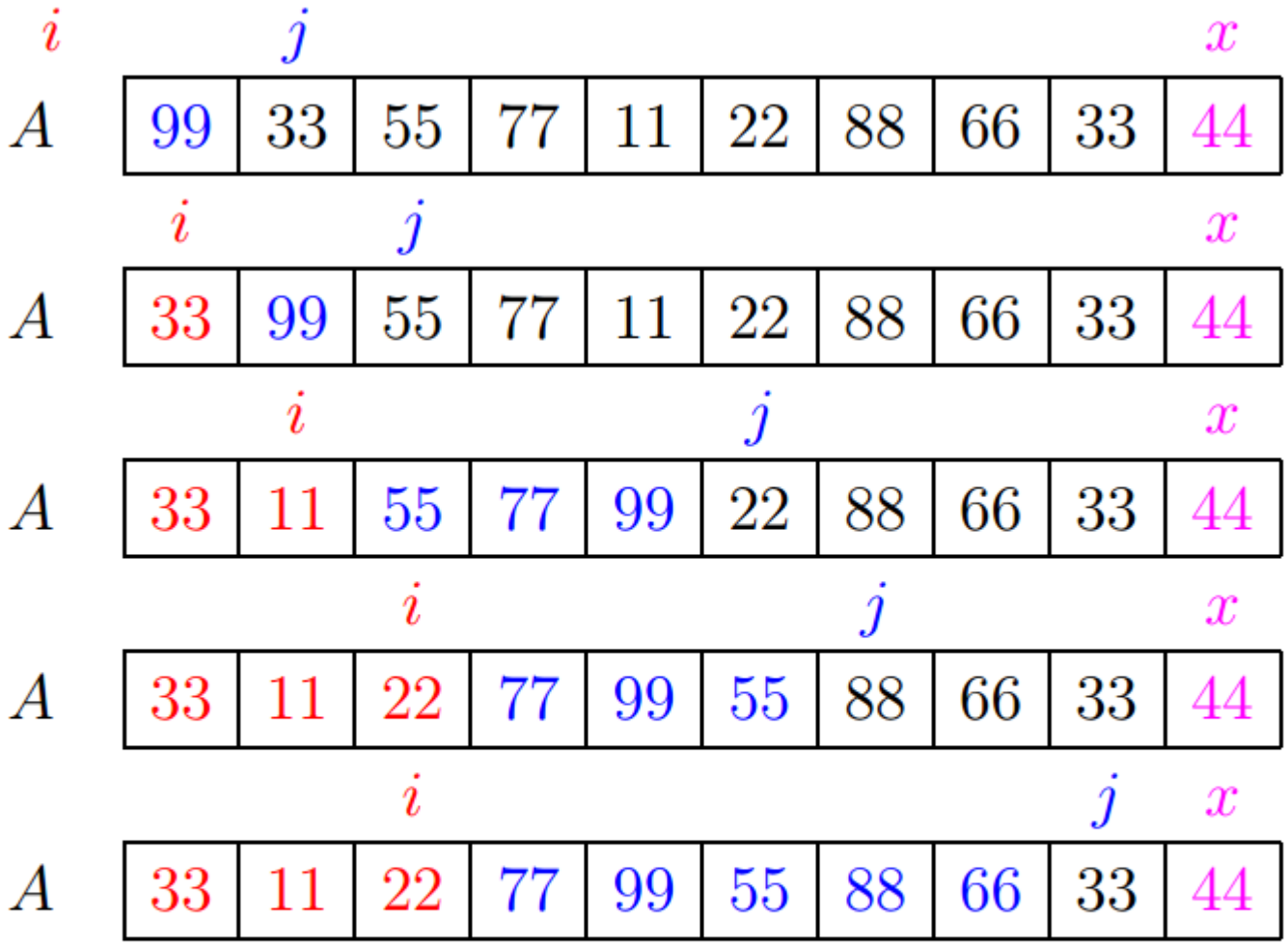

i
j
x
A
99 33 55 77 11 22 88 66 33 44
i
j
x
A
33 99 55 77 11 22 88 66 33 44
i
j
x
A
33 99 55 77 11 22 88 66 33 44

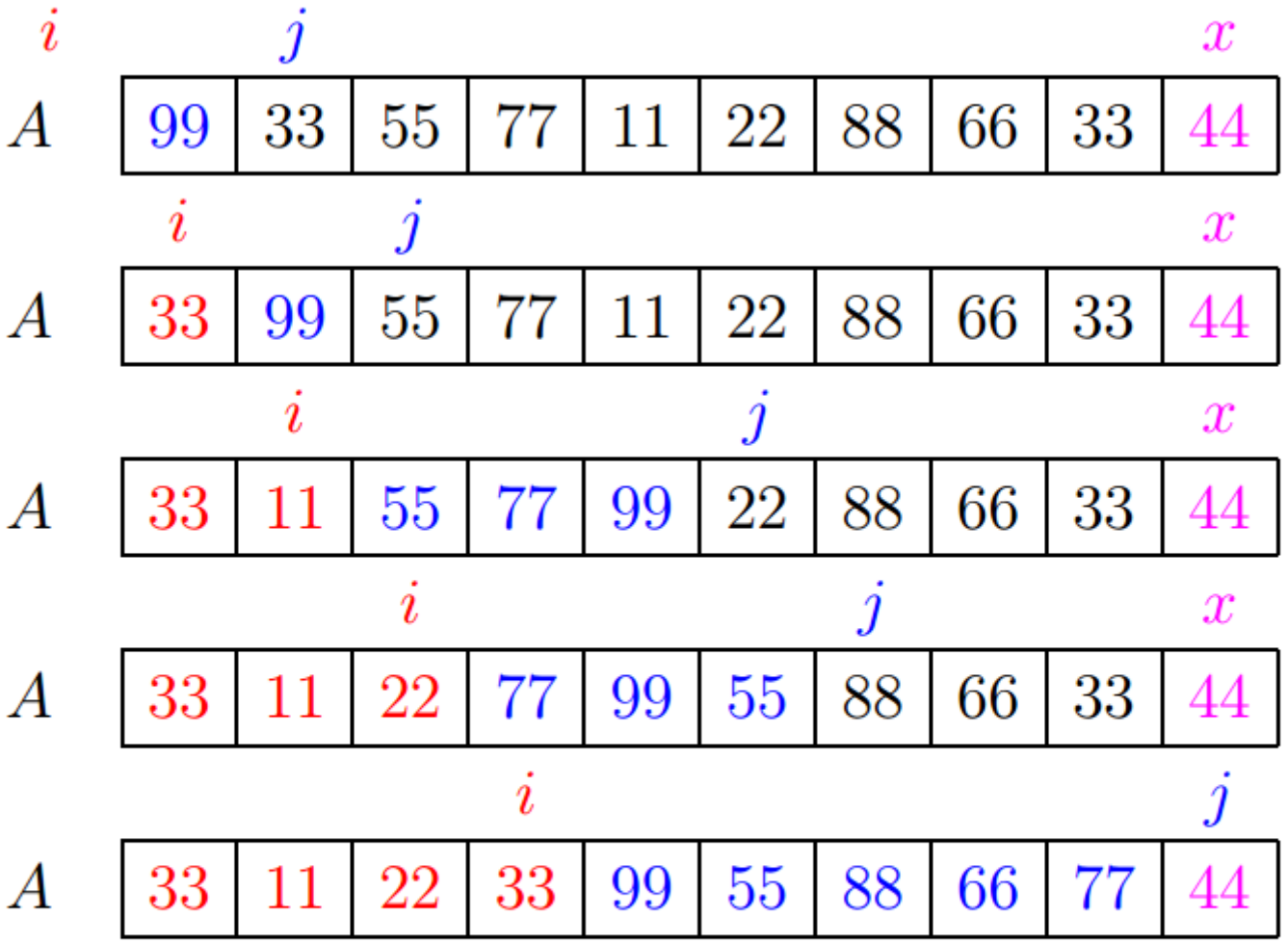

i
j
x
A
99 33 55 77 11 22 88 66 33 44
i
j
x
A
33 99 55 77 11 22 88 66 33 44
i
j
x
A
33 99 55 77 11 22 88 66 33 44

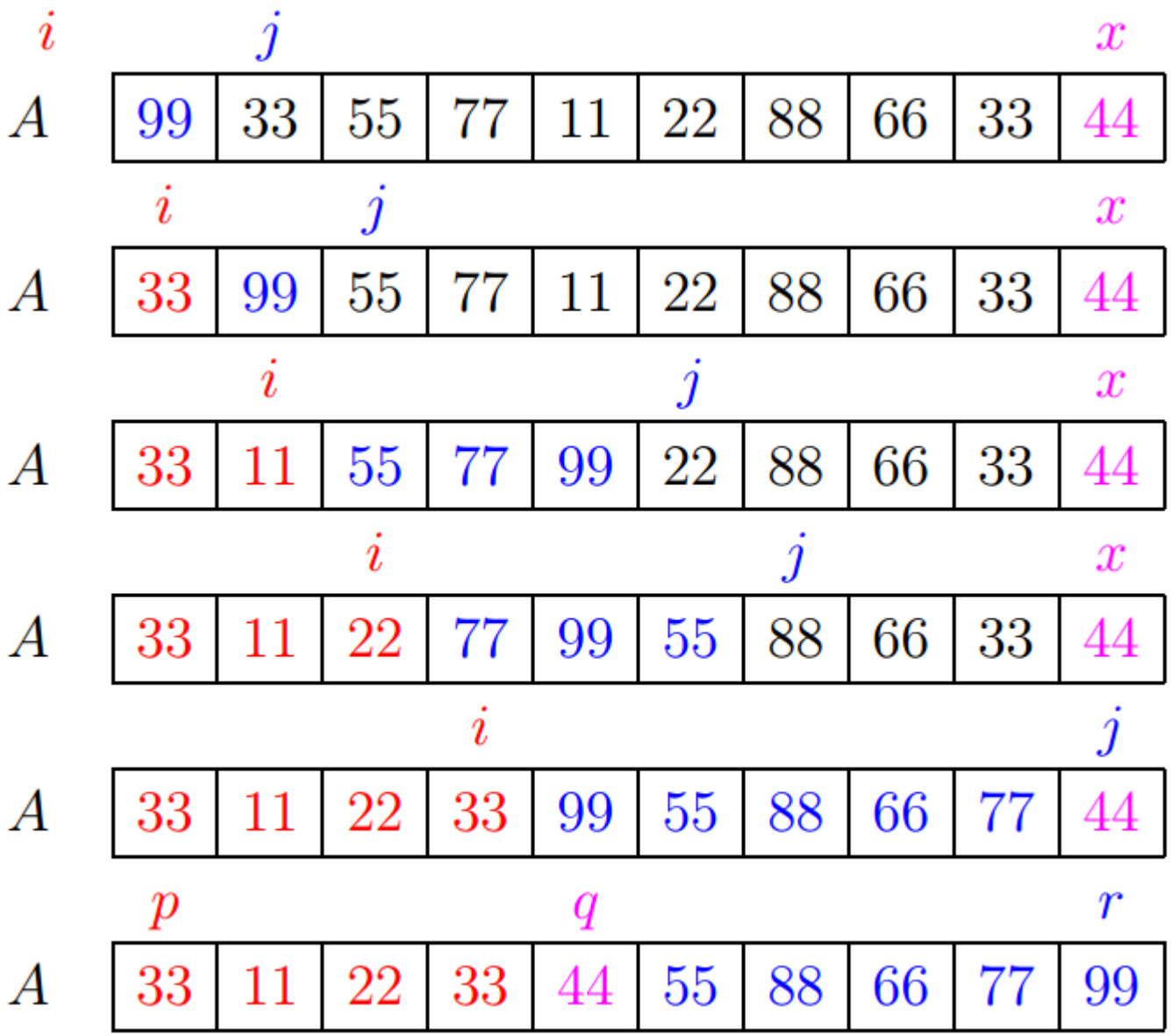

i
j
x
A
99 33 55 77 11 22 88 66 33 44
i
j
x
A
33 99 55 77 11 22 88 66 33 44
i
j
x
A
33 11 55 77 99 22 88 66 33 44


i
j
x
A
99 33 55 77 11 22 88 66 33 44
i
j
x
A
33 99 55 77 11 22 88 66 33 44
i
j
x
A
33 11 55 77 99 22 88 66 33 44
i
j
x
A
33 11 22 77 99 55 88 66 33 44


i
j
x
A
99 33 55 77 11 22 88 66 33 44
i
j
x
A
33 99 55 77 11 22 88 66 33 44
i
j
x
A
33 11 55 77 99 22 88 66 33 44
i
j
x
A
33 11 22 77 99 55 88 66 33 44
i
j
x
A
33 11 22 77 99 55 88 66 33 44


i
j
x
A
99 33 55 77 11 22 88 66 33 44
i
j
x
A
33 99 55 77 11 22 88 66 33 44
i
j
x
A
33 11 55 77 99 22 88 66 33 44
i
j
x
A
33 11 22 77 99 55 88 66 33 44
i
j
x
A
33 11 22 77 99 55 88 66 33 44


i
j
x
A
99 33 55 77 11 22 88 66 33 44
i
j
x
A
33 99 55 77 11 22 88 66 33 44
i
j
x
A
33 11 55 77 99 22 88 66 33 44
i
j
x
A
33 11 22 77 99 55 88 66 33 44
i
j
A
33 11 22 33 99 55 88 66 77 44


i
j
x
A
99 33 55 77 11 22 88 66 33 44
i
j
x
A
33 99 55 77 11 22 88 66 33 44
i
j
x
A
33 11 55 77 99 22 88 66 33 44
i
j
x
A
33 11 22 77 99 55 88 66 33 44
i
j
A
33 11 22 33 99 55 88 66 77 44
p
q
r
A
33 11 22 33 44 55 88 66 77 99

Partição: Separar elementos em um vetor

Problema: Rearranjar um dado vetor `A[p..r]` e devolver um índice `q`, p≤q≤r`,` tais que

$$A[p..q-1] \le A[q] \le A[q+1..r]$$

```
PARTICIONE (A, p, r)
1   x ← A[r]          ▷ x é o "pivô"
2   i ← p−1
3   para j ← p até r − 1 faça
4       se A[j] ≤ x
5           então i ← i + 1
6                 A[i] ↔ A[j]
7   A[i+1] ↔ A[r]
8   devolva i + 1
```

```
PARTICIONE (A, p, r)
1   x ← A[r]          ▷ x é o "pivô"
2   i ← p−1
3   para j ← p até r − 1 faça
4         se A[j] ≤ x
5               então i ← i + 1
6                     A[i] ↔ A[j]
7   A[i+1] ↔ A[r]
8   devolva i + 1
```

```c
int Particione(int A[], int p, int r) {
    int i, j, x, aux;

    x = A[r];
    i = p-1;

    for (j=p; j<=r-1; j++) {
        if (A[j]<=x) {
            i = i+1;
            aux = A[i];
            A[i] = A[j];
            A[j] = aux;
        }
        ImprimeVetor(A, r-p+1);
    }

    aux = A[i+1];
    A[i+1] = A[r];
    A[r] = aux;

    return i+1;
}
```

```c
int main()
{
    int A[] = {-1,14,13,34,17,15,10,46,23,12,41,30,21};
    int m=sizeof(A)/sizeof(A[0]);

    ImprimeVetor(A, m);
    printf("\nq=%d", Particione(A, 0, m-1));
    ImprimeVetor(A, m);
}
```

```
-1 14 13 34 17 15 10 46 23 12 41 30 21
q=7
-1 14 13 17 15 10 12 21 23 34 41 30 46
```

```
-1 14 13 34 17 15 10 46 23 12 41 30 21
-1 14 13 34 17 15 10 46 23 12 41 30 21
-1 14 13 34 17 15 10 46 23 12 41 30 21
-1 14 13 34 17 15 10 46 23 12 41 30 21
-1 14 13 17 34 15 10 46 23 12 41 30 21
-1 14 13 17 15 34 10 46 23 12 41 30 21
-1 14 13 17 15 10 34 46 23 12 41 30 21
-1 14 13 17 15 10 34 46 23 12 41 30 21
-1 14 13 17 15 10 34 46 23 12 41 30 21
-1 14 13 17 15 10 12 46 23 34 41 30 21
-1 14 13 17 15 10 12 46 23 34 41 30 21
q=7
-1 14 13 17 15 10 12 21 23 34 41 30 46
```

Alguns exemplos de execução

```
-1 14 13 34 17 15 10 46 23 12 41 30 99
-1 14 13 34 17 15 10 46 23 12 41 30 99
-1 14 13 34 17 15 10 46 23 12 41 30 99
-1 14 13 34 17 15 10 46 23 12 41 30 99
-1 14 13 34 17 15 10 46 23 12 41 30 99
-1 14 13 34 17 15 10 46 23 12 41 30 99
-1 14 13 34 17 15 10 46 23 12 41 30 99
-1 14 13 34 17 15 10 46 23 12 41 30 99
-1 14 13 34 17 15 10 46 23 12 41 30 99
-1 14 13 34 17 15 10 46 23 12 41 30 99
-1 14 13 34 17 15 10 46 23 12 41 30 99
-1 14 13 34 17 15 10 46 23 12 41 30 99
-1 14 13 34 17 15 10 46 23 12 41 30 99
q=12
-1 14 13 34 17 15 10 46 23 12 41 30 99
```

```
-1 14 13 34 17 15 10 46 23 12 41 30 -99
-1 14 13 34 17 15 10 46 23 12 41 30 -99
-1 14 13 34 17 15 10 46 23 12 41 30 -99
-1 14 13 34 17 15 10 46 23 12 41 30 -99
-1 14 13 34 17 15 10 46 23 12 41 30 -99
-1 14 13 34 17 15 10 46 23 12 41 30 -99
-1 14 13 34 17 15 10 46 23 12 41 30 -99
-1 14 13 34 17 15 10 46 23 12 41 30 -99
-1 14 13 34 17 15 10 46 23 12 41 30 -99
-1 14 13 34 17 15 10 46 23 12 41 30 -99
-1 14 13 34 17 15 10 46 23 12 41 30 -99
-1 14 13 34 17 15 10 46 23 12 41 30 -99
-1 14 13 34 17 15 10 46 23 12 41 30 -99
q=0
-99 14 13 34 17 15 10 46 23 12 41 30 -1
```

```
-1 14 13 34 17 15 10 46 23 12 41 30 21
-1 14 13 34 17 15 10 46 23 12 41 30 21
-1 14 13 34 17 15 10 46 23 12 41 30 21
-1 14 13 34 17 15 10 46 23 12 41 30 21
-1 14 13 34 17 15 10 46 23 12 41 30 21
-1 14 13 17 34 15 10 46 23 12 41 30 21
-1 14 13 17 15 34 10 46 23 12 41 30 21
-1 14 13 17 15 10 34 46 23 12 41 30 21
-1 14 13 17 15 10 34 46 23 12 41 30 21
-1 14 13 17 15 10 34 46 23 12 41 30 21
-1 14 13 17 15 10 12 46 23 34 41 30 21
-1 14 13 17 15 10 12 46 23 34 41 30 21
-1 14 13 17 15 10 12 46 23 34 41 30 21
q=7
-1 14 13 17 15 10 12 21 23 34 41 30 46
```

Como escolher um bom pivô?

O que é um bom pivô?

```
PARTICIONE (A, p, r)
1   x ← A[r]          ▷ x é o "pivô"
2   i ← p−1
3   para j ← p até r − 1 faça
4         se A[j] ≤ x
5               então i ← i + 1
6                     A[i] ↔ A[j]
7   A[i+1] ↔ A[r]
8   devolva i + 1
```

```c
int Particione(int A[], int p, int r) {
    int i, j, x, aux;

    x = A[r];
    i = p-1;

    for (j=p; j<=r-1; j++) {
        if (A[j]<=x) {
            i = i+1;
            aux = A[i];
            A[i] = A[j];
            A[j] = aux;
        }

    }

    aux = A[i+1];
    A[i+1] = A[r];
    A[r] = aux;

    return i+1;
}
```

Custo de tempo = O(n)
Considerando comparações ou swaps

Quick Sort

Rearranja $A[p\,..\,r]$ em ordem crescente.

```
QUICKSORT (A, p, r)
1    se p < r
2         então q ← PARTICIONE (A, p, r)
3               QUICKSORT (A, p, q − 1)
4               QUICKSORT (A, q + 1, r)
```

Rearranja $A[p\,..\,r]$ em ordem crescente.

```
QUICKSORT (A, p, r)
1   se p < r
2       então q ← PARTICIONE (A, p, r)
3             QUICKSORT (A, p, q − 1)
4             QUICKSORT (A, q + 1, r)
```

```
void QuickSort(int A[], int p, int r) {
    if (p<r) {
        int q = Particione(A, p, r);
        QuickSort(A, p, q-1);
        QuickSort(A, q+1, r);
    }
}
```

Questões importantes:

- Desempenho do algoritmo:
  - Melhor caso: O(n.lg(n))
  - Pior caso: $O(n^2)$

- Altura da pilha de execução do Quicksort (**P2**)

$T(n)$ é $\Theta(n^2)$.

O consumo de tempo do QUICKSORT no pior caso é $\mathrm{O}(n^2)$.

O consumo de tempo do QUICKSORT é $\mathrm{O}(n^2)$.

$M(n)$ é $\Theta(n \lg n)$.

O consumo de tempo do QUICKSORT no melhor caso é $\Omega(n \log n)$.

Na verdade . . .

O consumo de tempo do QUICKSORT no melhor caso é $\Theta(n \log n)$.

Questões importantes:

- Desempenho do algoritmo:
  - Melhor caso: O(n.lg(n))
  - Pior caso: O(n²)

- Altura da pilha de execução do Quicksort (**P2**)

Considere a seguinte variante do algoritmo Quicksort:

```
QUICKSORT′ (A, p, r)
    enquanto p < r faça
        q ← PARTICIONE (A, p, r)
        QUICKSORT′ (A, p, q − 1)
        p ← q + 1
```

Mostre que a pilha de recursão pode atingir altura proporcional a n, onde $n := r - p + 1$. Modifique o código de modo que a pilha de recursão tenha altura $\mathrm{O}(\lg n)$. (Veja enunciado completo em CLRS p.162.)

Ordenação baseada em comparações

Ordenação

- **Algoritmos basedos em Comparações**
  - Insertion sort
  - Selection sort
  - Bubble sort
  - Merge sort
  - Quick sort

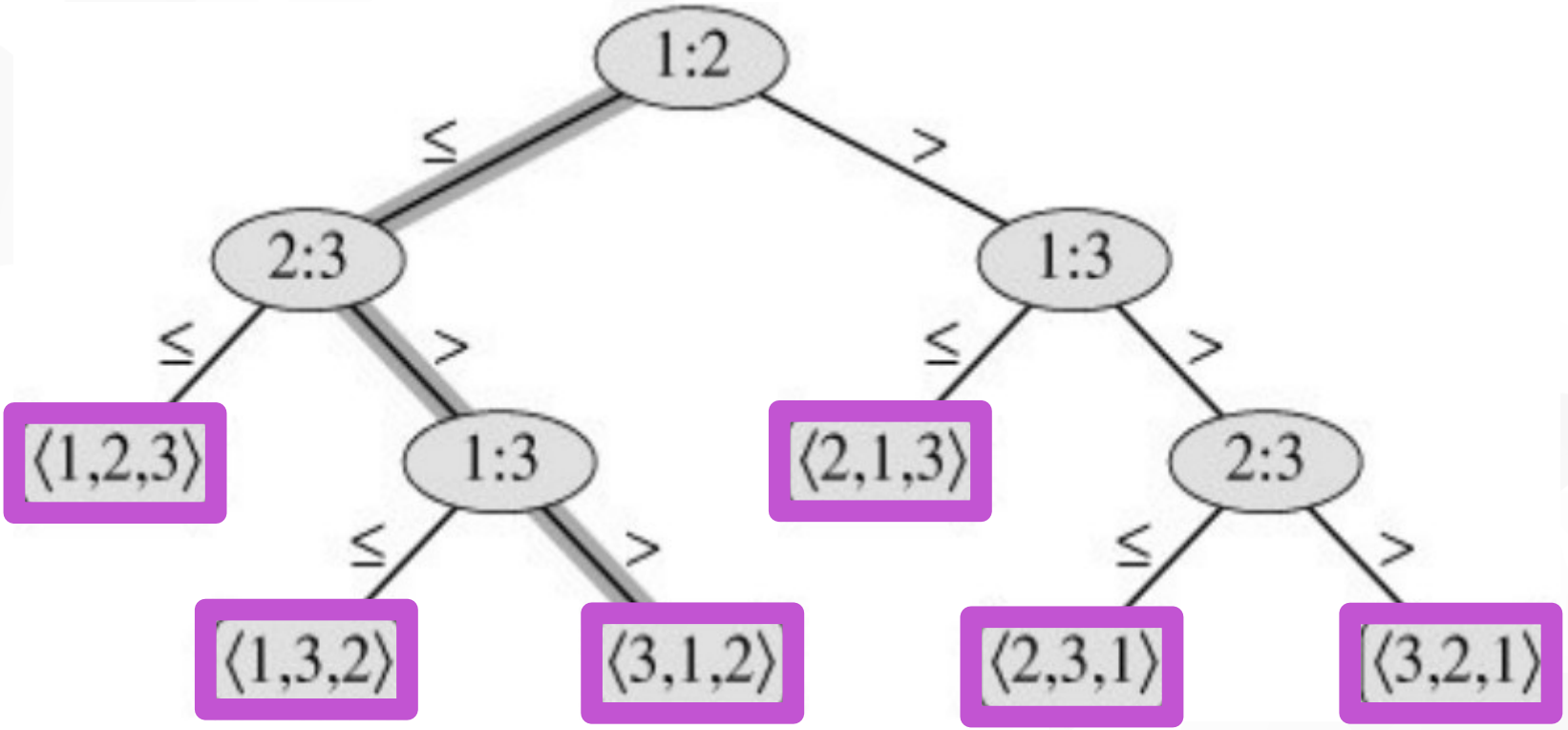
Complexidade computacional $\Omega(n \log(n))$

[limite matemático]
[limite assintótico para a ordenação]

Ordenação baseada em comparações

Sem perda de generalidade suponha que os valores a ser ordenados são sempre distintos

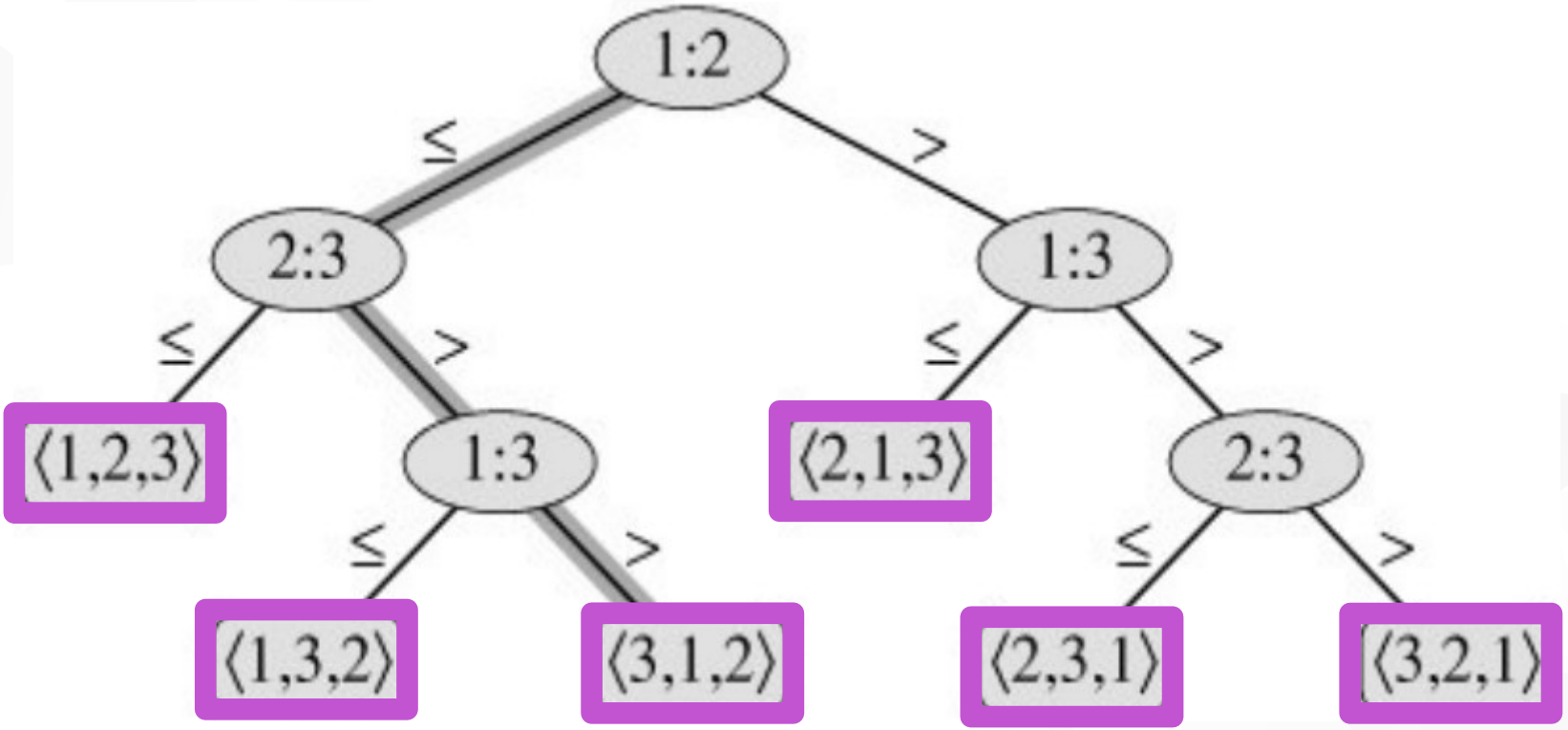
[Árvore de decisão]



Ordenação baseada em comparações

Sem perda de generalidade suponha que os valores a ser ordenados são sempre distintos

[Árvore de decisão]

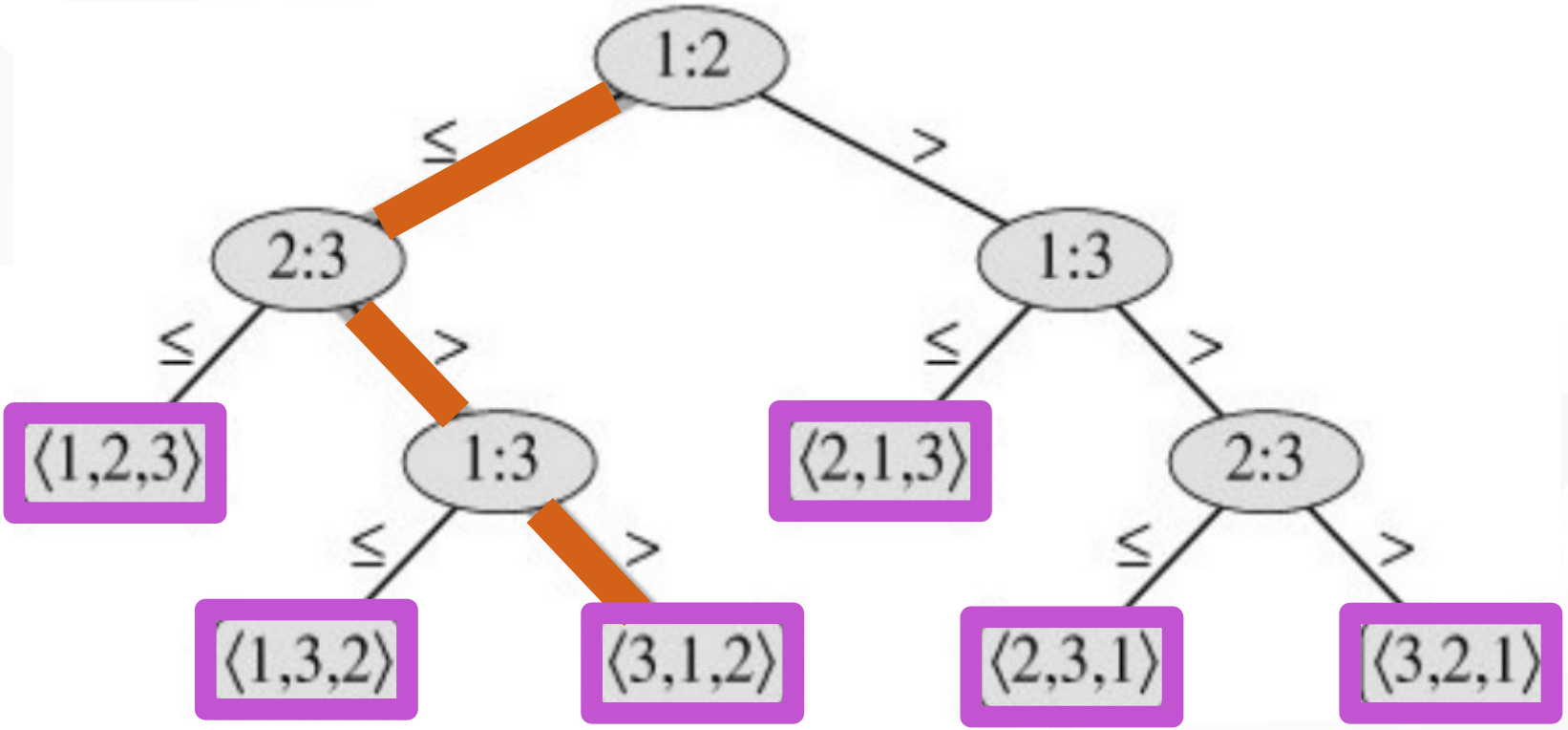


[Cada nó folha está associada a uma permutação dos elementos do vetor]

Ordenação baseada em comparações

Sem perda de generalidade suponha que os valores a ser ordenados são sempre distintos

[Árvore de decisão]



[Qualquer algoritmo de ordenação deverá percorrer um caminho desta árvore]

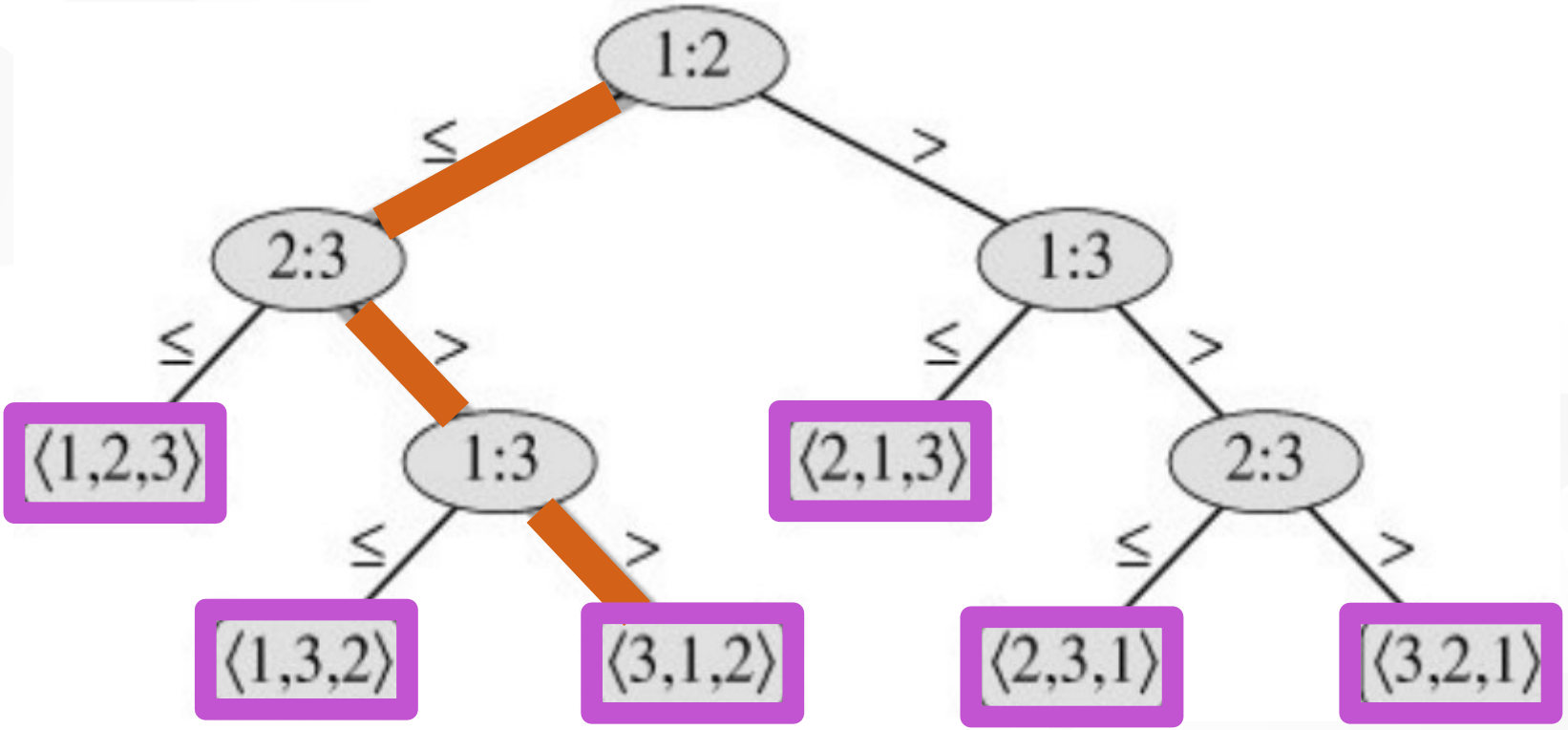
da raiz até a folha

Ordenação baseada em comparações

Sem perda de generalidade suponha que os valores a ser ordenados são sempre distintos

[Árvore de decisão]

Número de folhas = n!



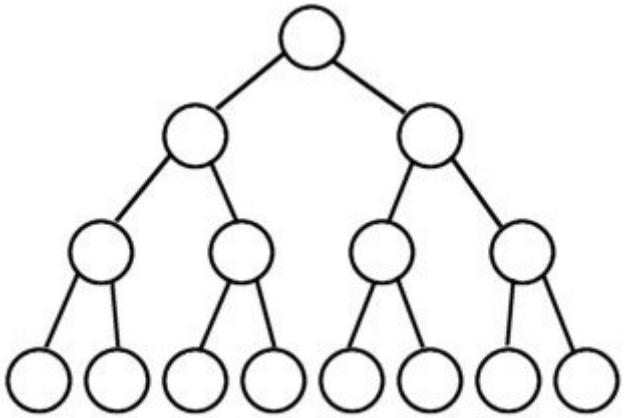
[Qualquer algoritmo de ordenação deverá percorrer um caminho desta árvore]

da raiz até a folha

Ordenação baseada em comparações

Seja ***L*** o número de folhas de uma árvore binária e ***h*** sua altura.

Então $L \leq 2^h$



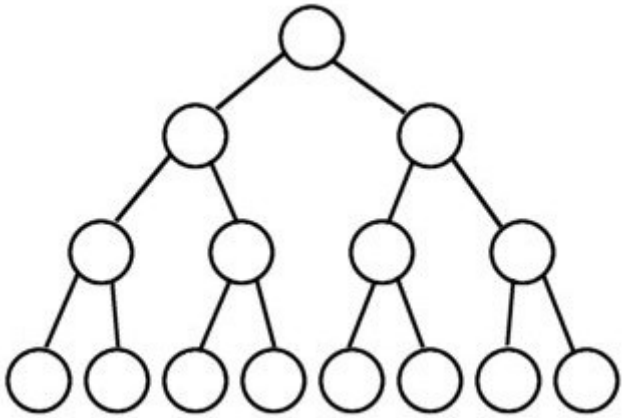
h=3

L=8

Ordenação baseada em comparações

Seja ***L*** o número de folhas de uma árvore binária e ***h*** sua altura.

Então $L \leq 2^h$



h=3

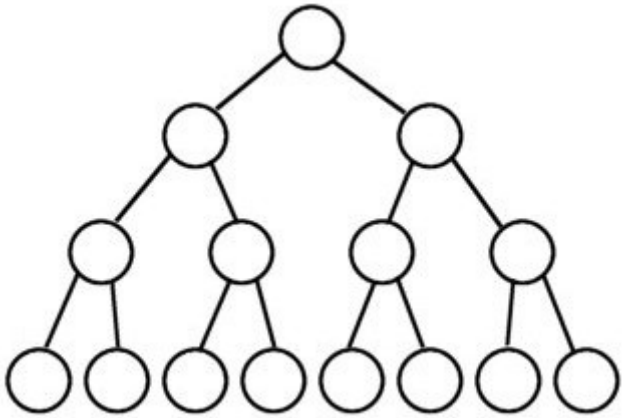
L=8

$h \geq \log(n!)$

Ordenação baseada em comparações

Seja ***L*** o número de folhas de uma árvore binária e ***h*** sua altura.

Então $L \leq 2^h$



h=3

L=8

$$h \geq \log(n!)$$

$$h = \Omega(n \log n)$$

Ordenação baseada em comparações

- **Algoritmos basedo em Comparaçães**
  - Insertion sort
  - Selection sort
  - Bubble sort
  - Merge sort
  - Quick sort

Vários algoritmos aqui listados são ótimos pois a sua complexidade computacional é $O(n \log n)$

Recursos computacionais

- Sortvis: http://sortvis.org/
- Sorting: http://sorting.at/
- Data struture visualizations
  https://www.cs.usfca.edu/~galles/visualization/Algorithms.html
- 15 Sorting Algorithms in 6 Minutes
  https://www.youtube.com/watch?v=kPRA0W1kECg