Processamento da Informação

Exercícios de programação

Prof. Jesús P. Mena-Chalco
CMCC/UFABC

Q2/2018

Exercício 1: Número de bits

Crie um método que, para um número **n** inteiro positivo, devolva o número de bits necessários para armazenar **n**.

Assinatura:

```
static int numeroBits( int n )
```

Exemplos:

```
n = 1                          (1 = 0001)
Resposta = 1

n = 8
Resposta = 4                   (8 = 1000)

n = 10
Resposta = 4                   (10 = 1010)

n = 138
Resposta = 8                   (138 = 10001010)
```

Exercício 1: Número de bits

```
static int numeroBits( int n ) {
    int cont = 1;

    while (n/2>0) {
        n = n/2;
        cont = cont+1;
    }

    return cont;
}
```

Exercício 2: Encaixe

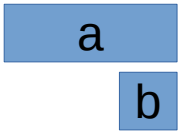
Escreva um método que, recebendo dois números inteiros **a** e **b** como parâmetros, verifica se **b** corresponde aos últimos dígitos de **a**.

Assinatura:

```
static boolean encaixa( int a, int b )
```

Exemplos:

- a = 12345 , b=45 → Resposta = true
- a = 12345 , b=5 → Resposta = true
- a = 12345 , b=12 → Resposta = false
- a = 12 , b=12 → Resposta = true
- a = 12 , b=1212 → Resposta = false

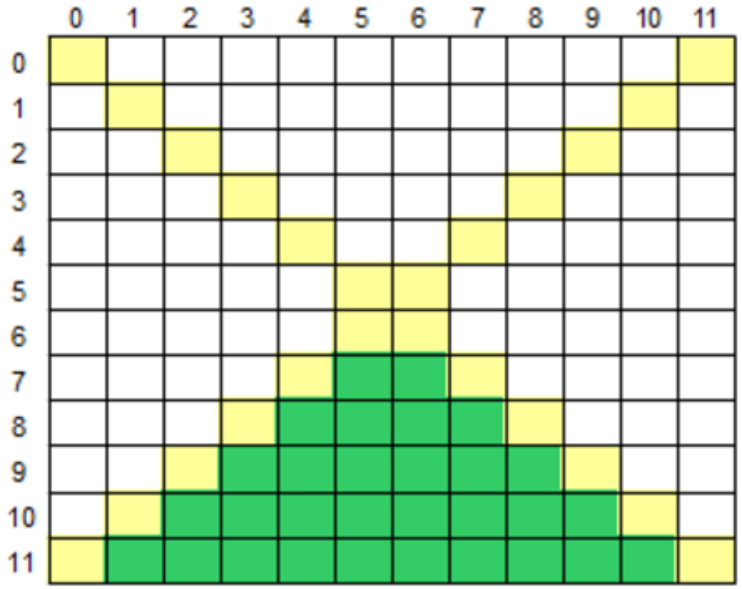


Exercício 2: Encaixe

```
static boolean encaixa( int a, int b ) {

    while ( a%10==b%10 && a>=b ) {
        a = a/10;
        b = b/10;

        if (b==0) {
            return true;
        }
    }

    return false;
}
```

Sobre dois exercícios de matrizes

Área inferior (Problema 1188)



	0	1	2	3	4	5	6	7	8	9	10	11
0	(0 , 0)	(0 , 1)	(0 , 2)	(0 , 3)	(0 , 4)	(0 , 5)	(0 , 6)	(0 , 7)	(0 , 8)	(0 , 9)	(0 , 10)	(0 , 11)
1	(1 , 0)	(1 , 1)	(1 , 2)	(1 , 3)	(1 , 4)	(1 , 5)	(1 , 6)	(1 , 7)	(1 , 8)	(1 , 9)	(1 , 10)	(1 , 11)
2	(2 , 0)	(2 , 1)	(2 , 2)	(2 , 3)	(2 , 4)	(2 , 5)	(2 , 6)	(2 , 7)	(2 , 8)	(2 , 9)	(2 , 10)	(2 , 11)
3	(3 , 0)	(3 , 1)	(3 , 2)	(3 , 3)	(3 , 4)	(3 , 5)	(3 , 6)	(3 , 7)	(3 , 8)	(3 , 9)	(3 , 10)	(3 , 11)
4	(4 , 0)	(4 , 1)	(4 , 2)	(4 , 3)	(4 , 4)	(4 , 5)	(4 , 6)	(4 , 7)	(4 , 8)	(4 , 9)	(4 , 10)	(4 , 11)
5	(5 , 0)	(5 , 1)	(5 , 2)	(5 , 3)	(5 , 4)	(5 , 5)	(5 , 6)	(5 , 7)	(5 , 8)	(5 , 9)	(5 , 10)	(5 , 11)
6	(6 , 0)	(6 , 1)	(6 , 2)	(6 , 3)	(6 , 4)	(6 , 5)	(6 , 6)	(6 , 7)	(6 , 8)	(6 , 9)	(6 , 10)	(6 , 11)
7	(7 , 0)	(7 , 1)	(7 , 2)	(7 , 3)	(7 , 4)	(7 , 5)	(7 , 6)	(7 , 7)	(7 , 8)	(7 , 9)	(7 , 10)	(7 , 11)
8	(8 , 0)	(8 , 1)	(8 , 2)	(8 , 3)	(8 , 4)	(8 , 5)	(8 , 6)	(8 , 7)	(8 , 8)	(8 , 9)	(8 , 10)	(8 , 11)
9	(9 , 0)	(9 , 1)	(9 , 2)	(9 , 3)	(9 , 4)	(9 , 5)	(9 , 6)	(9 , 7)	(9 , 8)	(9 , 9)	(9 , 10)	(9 , 11)
10	(10 , 0)	(10 , 1)	(10 , 2)	(10 , 3)	(10 , 4)	(10 , 5)	(10 , 6)	(10 , 7)	(10 , 8)	(10 , 9)	(10 , 10)	(10 , 11)
11	(11 , 0)	(11 , 1)	(11 , 2)	(11 , 3)	(11 , 4)	(11 , 5)	(11 , 6)	(11 , 7)	(11 , 8)	(11 , 9)	(11 , 10)	(11 , 11)

	0	1	2	3	4	5	6	7	8	9	10	11
0	(0 , 0)	(0 , 1)	(0 , 2)	(0 , 3)	(0 , 4)	(0 , 5)	(0 , 6)	(0 , 7)	(0 , 8)	(0 , 9)	(0 , 10)	(0 , 11)
1	(1 , 0)	(1 , 1)	(1 , 2)	(1 , 3)	(1 , 4)	(1 , 5)	(1 , 6)	(1 , 7)	(1 , 8)	(1 , 9)	(1 , 10)	(1 , 11)
2	(2 , 0)	(2 , 1)	(2 , 2)	(2 , 3)	(2 , 4)	(2 , 5)	(2 , 6)	(2 , 7)	(2 , 8)	(2 , 9)	(2 , 10)	(2 , 11)
3	(3 , 0)	(3 , 1)	(3 , 2)	(3 , 3)	(3 , 4)	(3 , 5)	(3 , 6)	(3 , 7)	(3 , 8)	(3 , 9)	(3 , 10)	(3 , 11)
4	(4 , 0)	(4 , 1)	(4 , 2)	(4 , 3)	(4 , 4)	(4 , 5)	(4 , 6)	(4 , 7)	(4 , 8)	(4 , 9)	(4 , 10)	(4 , 11)
5	(5 , 0)	(5 , 1)	(5 , 2)	(5 , 3)	(5 , 4)	(5 , 5)	(5 , 6)	(5 , 7)	(5 , 8)	(5 , 9)	(5 , 10)	(5 , 11)
6	(6 , 0)	(6 , 1)	(6 , 2)	(6 , 3)	(6 , 4)	(6 , 5)	(6 , 6)	(6 , 7)	(6 , 8)	(6 , 9)	(6 , 10)	(6 , 11)
7	(7 , 0)	(7 , 1)	(7 , 2)	(7 , 3)	(7 , 4)	(7 , 5)	(7 , 6)	(7 , 7)	(7 , 8)	(7 , 9)	(7 , 10)	(7 , 11)
8	(8 , 0)	(8 , 1)	(8 , 2)	(8 , 3)	(8 , 4)	(8 , 5)	(8 , 6)	(8 , 7)	(8 , 8)	(8 , 9)	(8 , 10)	(8 , 11)
9	(9 , 0)	(9 , 1)	(9 , 2)	(9 , 3)	(9 , 4)	(9 , 5)	(9 , 6)	(9 , 7)	(9 , 8)	(9 , 9)	(9 , 10)	(9 , 11)
10	(10 , 0)	(10 , 1)	(10 , 2)	(10 , 3)	(10 , 4)	(10 , 5)	(10 , 6)	(10 , 7)	(10 , 8)	(10 , 9)	(10 , 10)	(10 , 11)
11	(11 , 0)	(11 , 1)	(11 , 2)	(11 , 3)	(11 , 4)	(11 , 5)	(11 , 6)	(11 , 7)	(11 , 8)	(11 , 9)	(11 , 10)	(11 , 11)

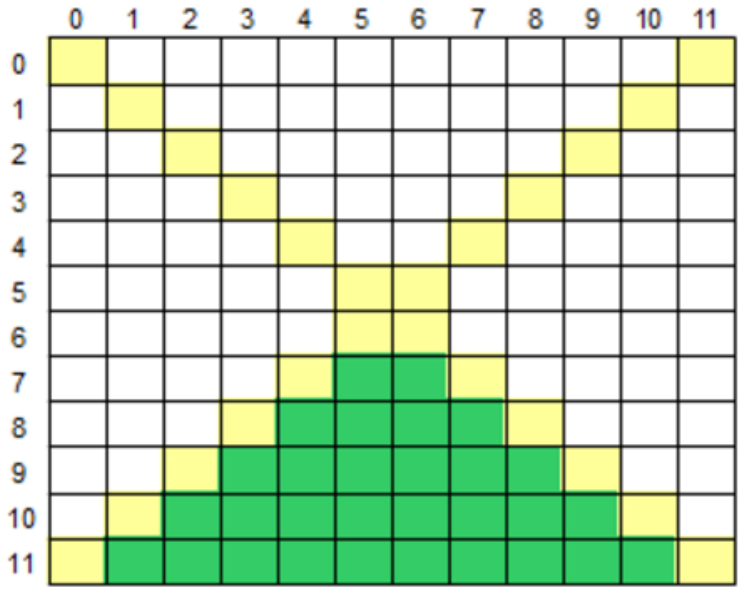
i > j

	0	1	2	3	4	5	6	7	8	9	10	11
0	(0 , 0)	(0 , 1)	(0 , 2)	(0 , 3)	(0 , 4)	(0 , 5)	(0 , 6)	(0 , 7)	(0 , 8)	(0 , 9)	(0 , 10)	(0 , 11)
1	(1 , 0)	(1 , 1)	(1 , 2)	(1 , 3)	(1 , 4)	(1 , 5)	(1 , 6)	(1 , 7)	(1 , 8)	(1 , 9)	(1 , 10)	(1 , 11)
2	(2 , 0)	(2 , 1)	(2 , 2)	(2 , 3)	(2 , 4)	(2 , 5)	(2 , 6)	(2 , 7)	(2 , 8)	(2 , 9)	(2 , 10)	(2 , 11)
3	(3 , 0)	(3 , 1)	(3 , 2)	(3 , 3)	(3 , 4)	(3 , 5)	(3 , 6)	(3 , 7)	(3 , 8)	(3 , 9)	(3 , 10)	(3 , 11)
4	(4 , 0)	(4 , 1)	(4 , 2)	(4 , 3)	(4 , 4)	(4 , 5)	(4 , 6)	(4 , 7)	(4 , 8)	(4 , 9)	(4 , 10)	(4 , 11)
5	(5 , 0)	(5 , 1)	(5 , 2)	(5 , 3)	(5 , 4)	(5 , 5)	(5 , 6)	(5 , 7)	(5 , 8)	(5 , 9)	(5 , 10)	(5 , 11)
6	(6 , 0)	(6 , 1)	(6 , 2)	(6 , 3)	(6 , 4)	(6 , 5)	(6 , 6)	(6 , 7)	(6 , 8)	(6 , 9)	(6 , 10)	(6 , 11)
7	(7 , 0)	(7 , 1)	(7 , 2)	(7 , 3)	(7 , 4)	(7 , 5)	(7 , 6)	(7 , 7)	(7 , 8)	(7 , 9)	(7 , 10)	(7 , 11)
8	(8 , 0)	(8 , 1)	(8 , 2)	(8 , 3)	(8 , 4)	(8 , 5)	(8 , 6)	(8 , 7)	(8 , 8)	(8 , 9)	(8 , 10)	(8 , 11)
9	(9 , 0)	(9 , 1)	(9 , 2)	(9 , 3)	(9 , 4)	(9 , 5)	(9 , 6)	(9 , 7)	(9 , 8)	(9 , 9)	(9 , 10)	(9 , 11)
10	(10 , 0)	(10 , 1)	(10 , 2)	(10 , 3)	(10 , 4)	(10 , 5)	(10 , 6)	(10 , 7)	(10 , 8)	(10 , 9)	(10 , 10)	(10 , 11)
11	(11 , 0)	(11 , 1)	(11 , 2)	(11 , 3)	(11 , 4)	(11 , 5)	(11 , 6)	(11 , 7)	(11 , 8)	(11 , 9)	(11 , 10)	(11 , 11)

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	1	2	3	4	5	6	7	8	9	10	11
1	1	2	3	4	5	6	7	8	9	10	11	12
2	2	3	4	5	6	7	8	9	10	11	12	13
3	3	4	5	6	7	8	9	10	11	12	13	14
4	4	5	6	7	8	9	10	11	12	13	14	15
5	5	6	7	8	9	10	11	12	13	14	15	16
6	6	7	8	9	10	11	12	13	14	15	16	17
7	7	8	9	10	11	12	13	14	15	16	17	18
8	8	9	10	11	12	13	14	15	16	17	18	19
9	9	10	11	12	13	14	15	16	17	18	19	20
10	10	11	12	13	14	15	16	17	18	19	20	21
11	11	12	13	14	15	16	17	18	19	20	21	22

i + j >=12

Área inferior (Problema 1188)



(i > j) && (i + j >=12)

Área inferior (Problema 1188)

```java
class Main {

  public static void main(String[] args) {
    Scanner sc = new Scanner (System.in);

    int i, j, elementos=0;
    double mij, soma=0;
    String T = sc.next();

    for (i=0 ; i<12 ; i++) {
      for (j=0 ; j<12; j++) {
        mij = sc.nextDouble();
        if (i>j && (i+j)>=12) {
          soma += mij;
          elementos++;
        }
      }
    }

    if (T.charAt(0) == 'S')
      System.out.printf("%.1f\n", soma);
    else
      System.out.printf("%.1f\n", soma/elementos);
  }

}
```

Atividade em aula

Particiona

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

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

    for (j=p; j<=r-1; j=j+1) {
        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;
}
```

Particiona

```
int p = 0;
int r = 9;
int A[] = {99, 33, 55, 77, 11, 22, 88, 66, 33, 44};

int x = particiona(A, p, r);
```

(a) Resposta: x = 4

(b) A = {33, 11, 22, 33, **44**, 55, 88, 66, 77, 99}

(c) O método particiona o vetor **A** considerando como elemento pivô o elemento 44.

- Os elementos <= 44 estão no lado esquerdo.
- Os elementos >= 44 estão no lado direito.

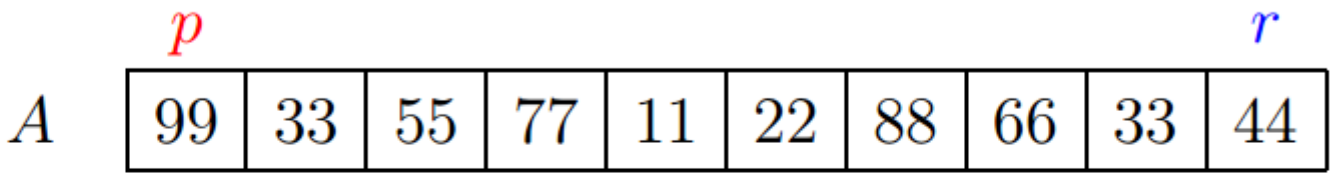
Sobre um algoritmo sofisticado para ordenar um vetor: Quicksort

Partição: Separar elementos em um vetor

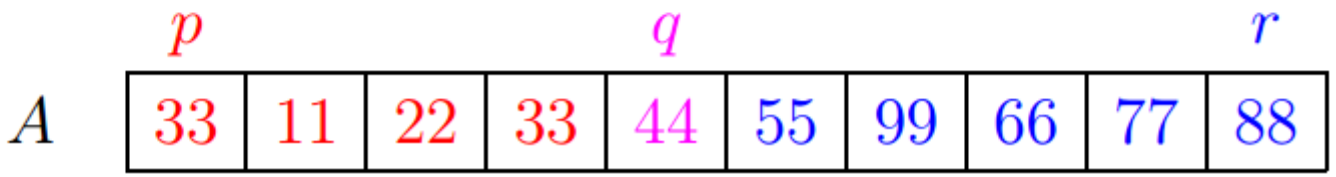
Problema: Rearranjar um dado vetor `A[p..r]` e devolver um índice `q`, p≤q≤r, tais que

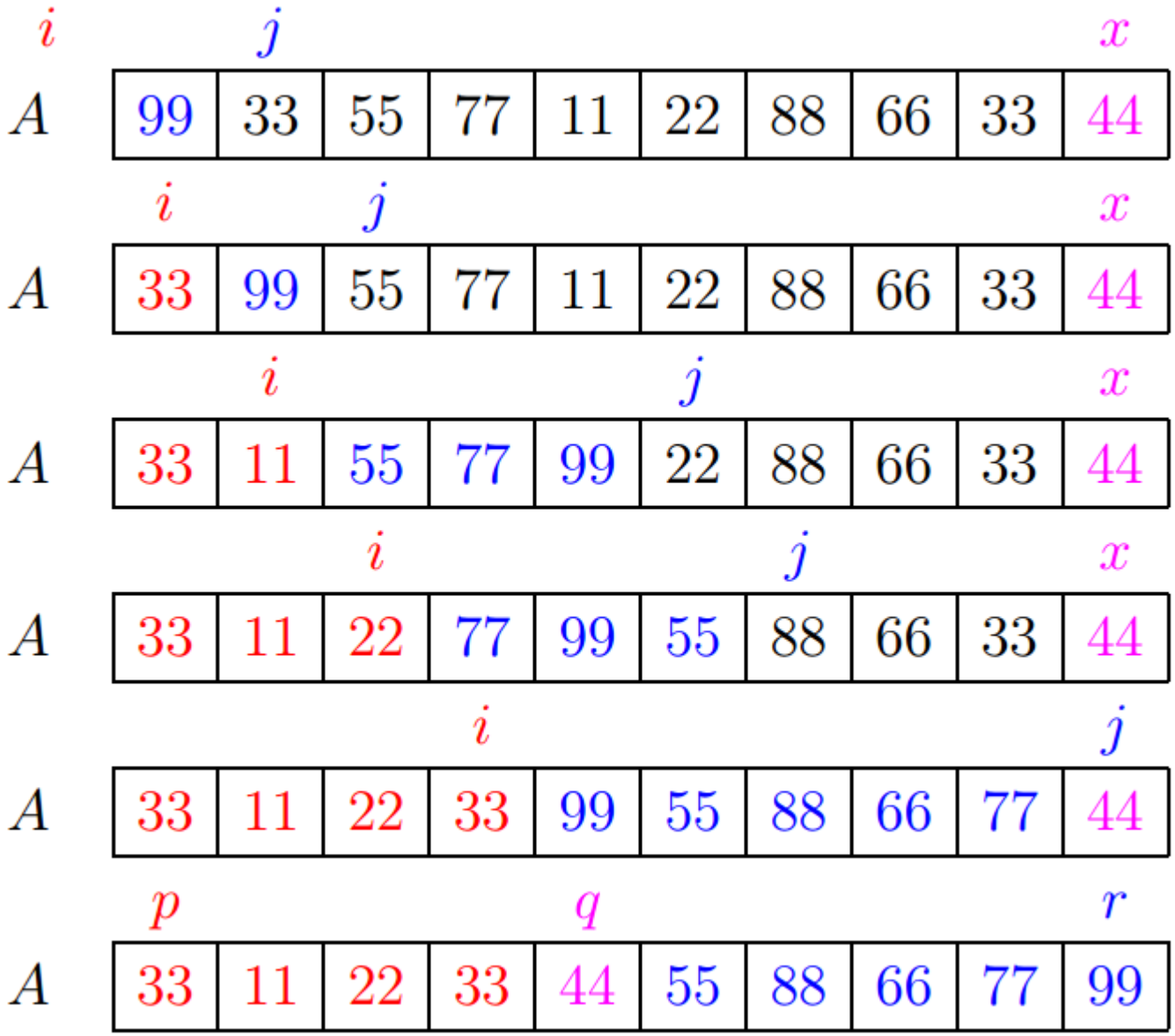
`A[p..q-1]` ≤ `A[q]` ≤ `A[q+1..r]`

Entra:

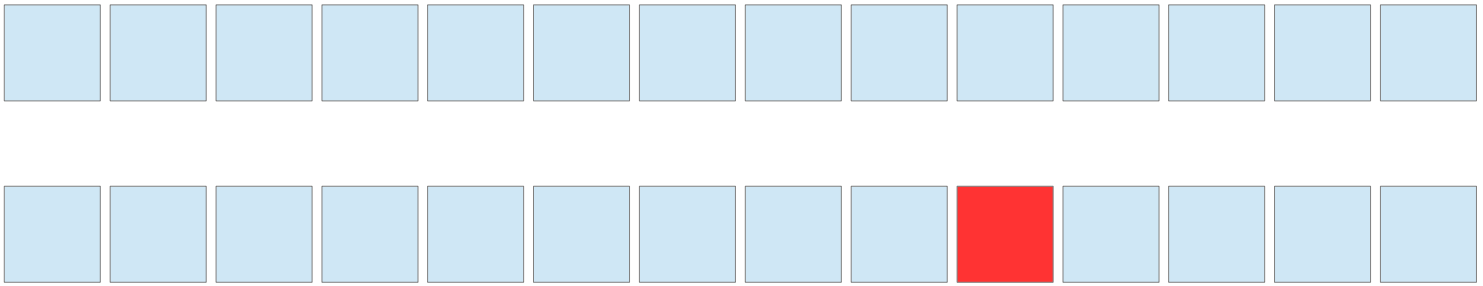


Sai:

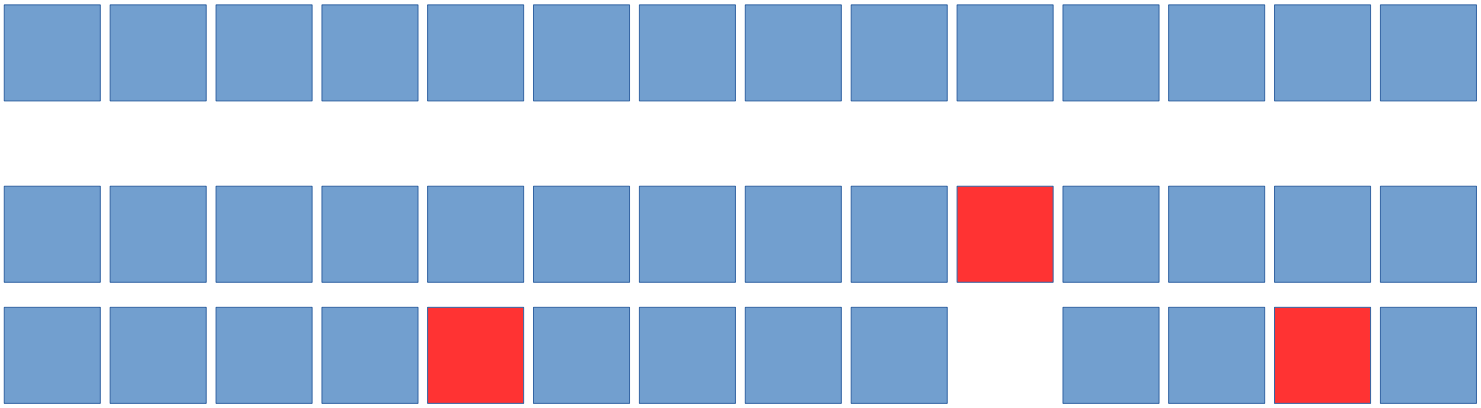



i j x
A 99 33 55 77 11 22 88 66 33 44
i j x
A 33 99 55 77 11 22 88 66 33 44
i j x
A 33 11 55 77 99 22 88 66 33 44
i j x
A 33 11 22 77 99 55 88 66 33 44
i j
A 33 11 22 33 99 55 88 66 77 44
p q r
A 33 11 22 33 44 55 88 66 77 99

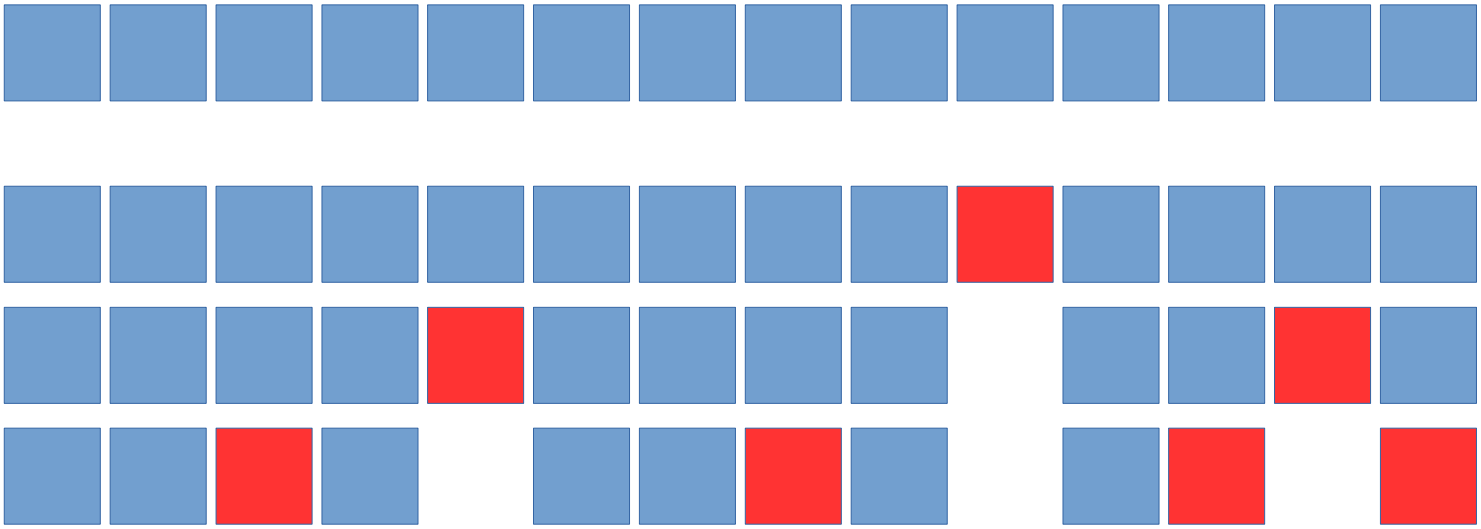
QuickSort



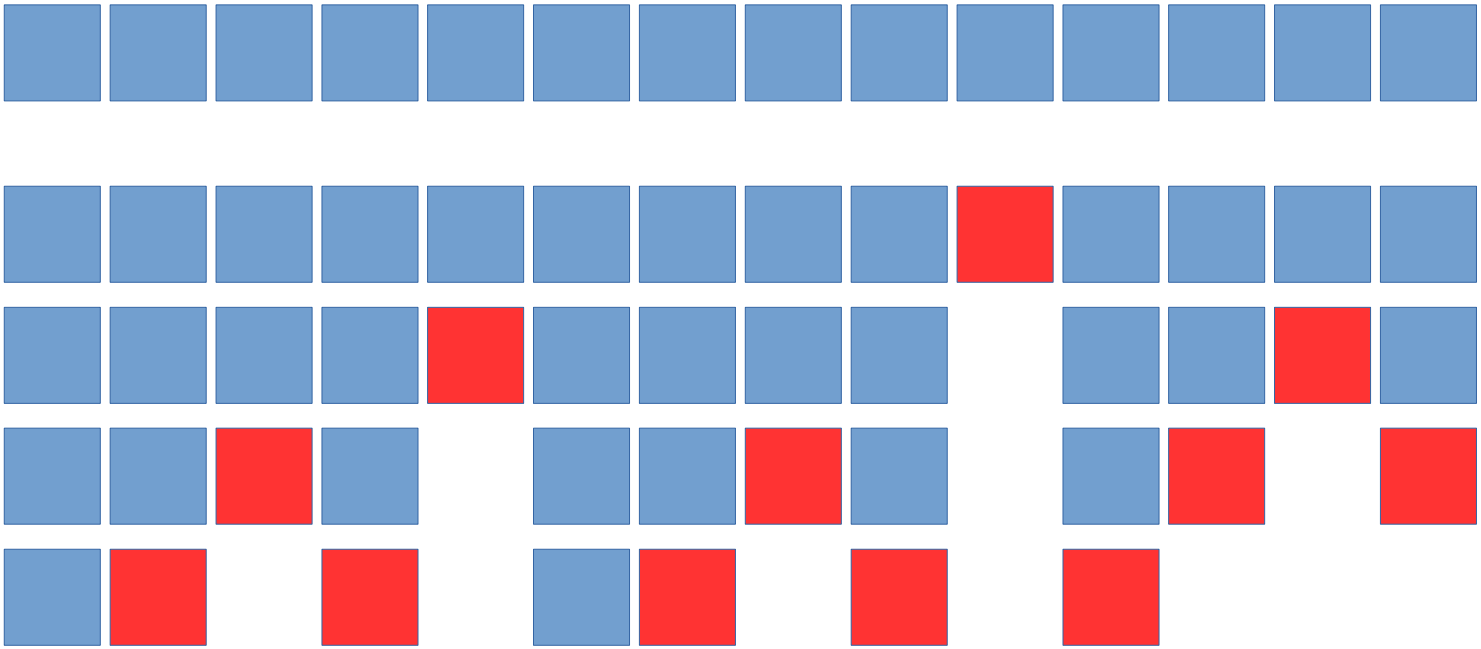
QuickSort



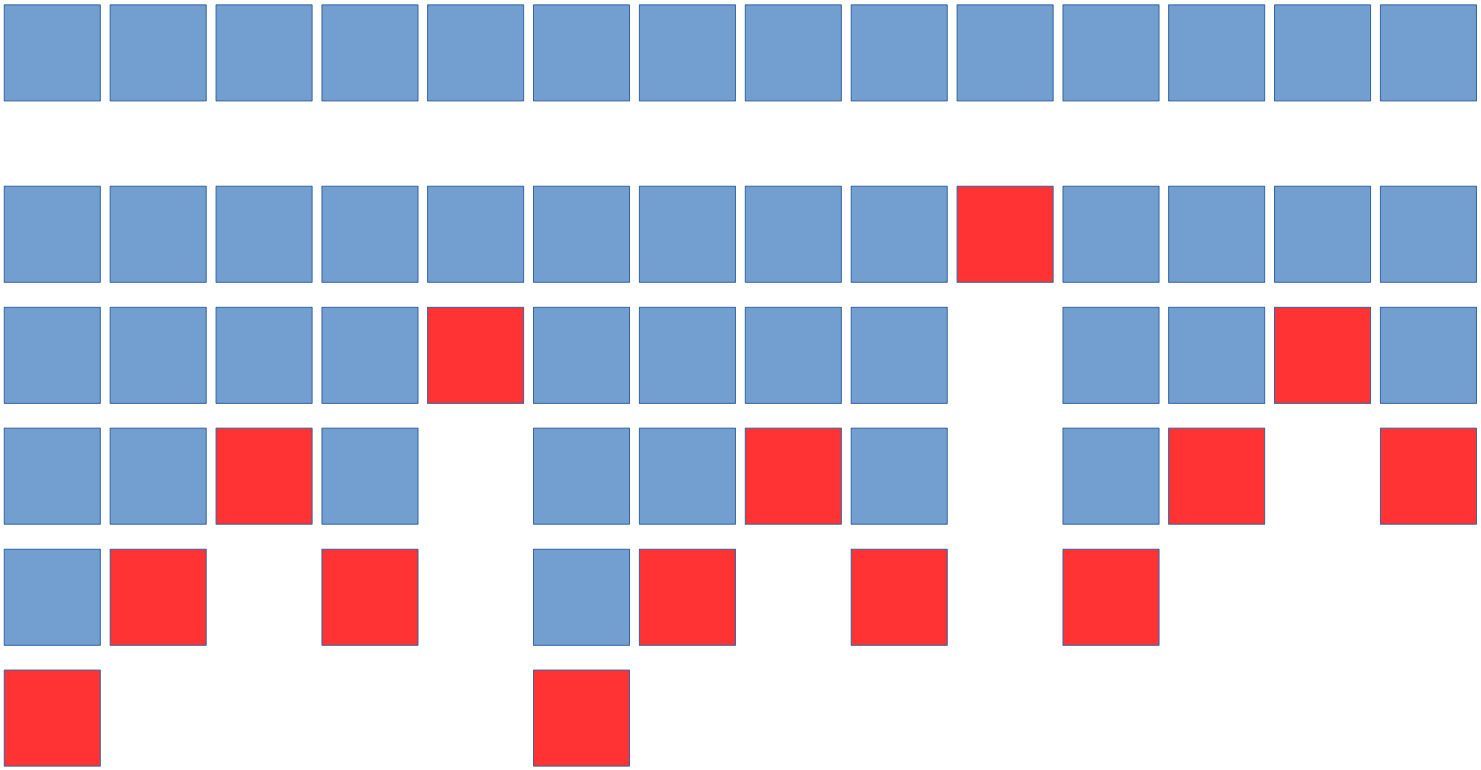
QuickSort



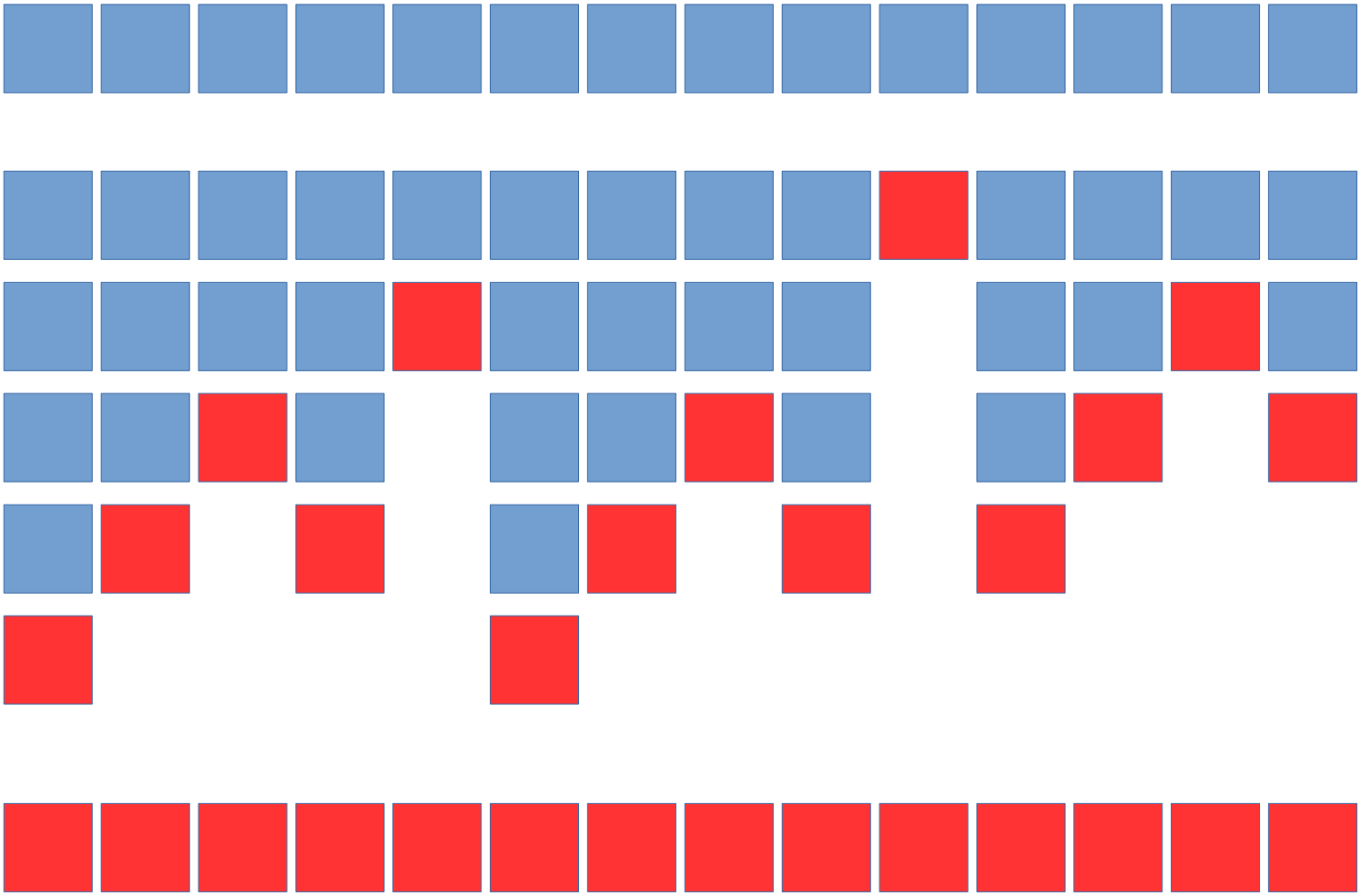
QuickSort



QuickSort



QuickSort



QuickSort

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

https://www.youtube.com/watch?v=vxENKlcs2Tw