EDB-I

Estrutura de Dados Básicas I

Aula 15
Algoritmos de Ordenação
Quick Sort

(material baseado nas notas de aula do Prof. César Rennó Costa e Prof. Eiji Adachi)

UFRN
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

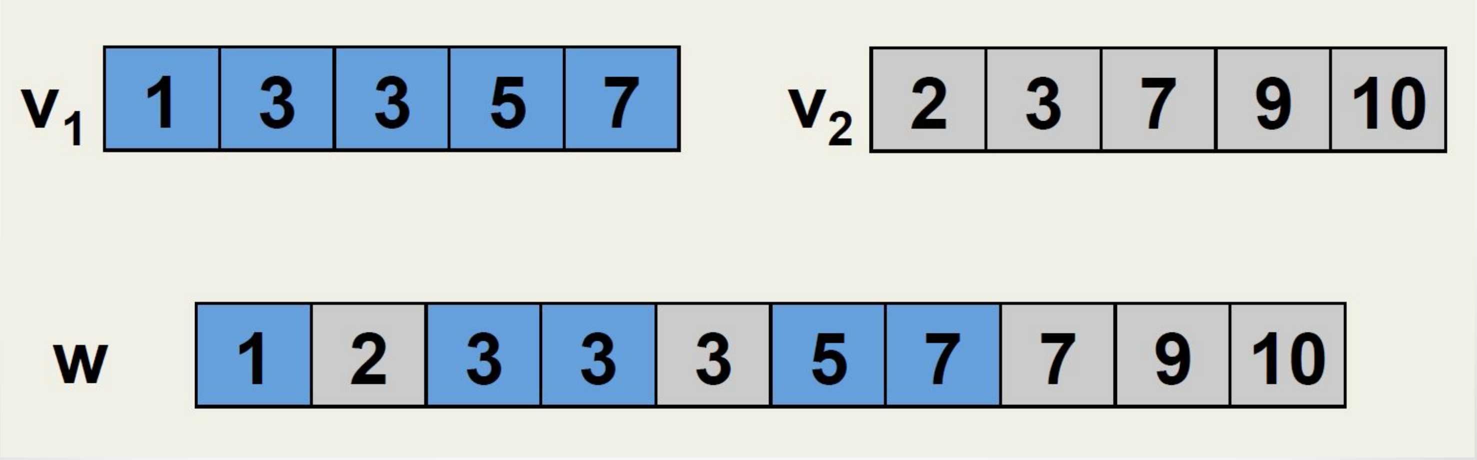
IMD
INSTITUTO METRÓPOLE DIGITAL

Merge Sort

ordenação por intercalação

Idéia Geral

A partir de dois vetores ordenados, posso construir um outro também ordenado

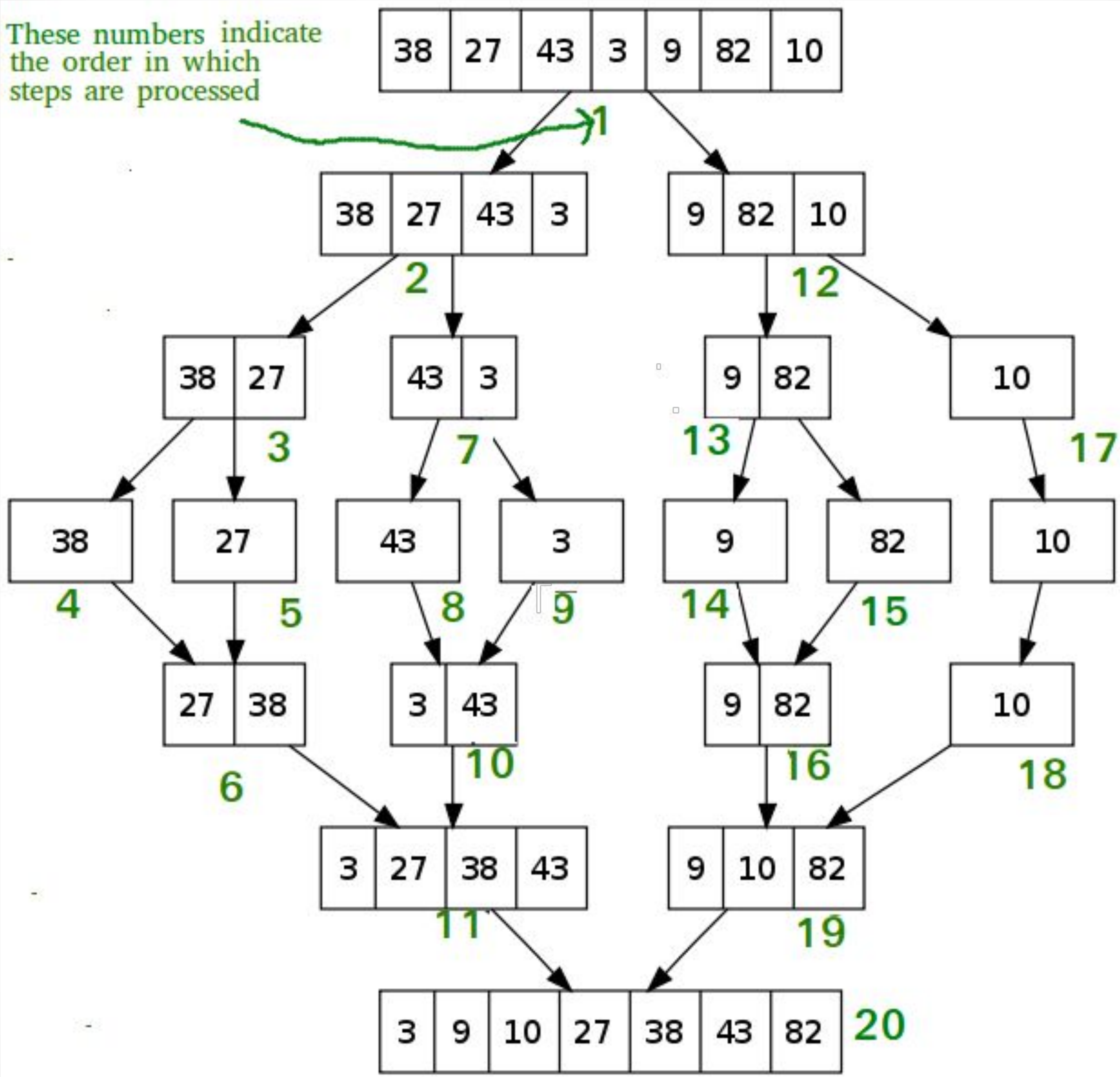


```c
void merge(int *aux, int *v, int inicio, int meio, int fim){
    int i, j, k;
    i = inicio;
    j = meio + 1;
    k = inicio;
    while(i <= meio && j <= fim){
        if(v[i] < v[j]){
            aux[k] = v[i];
            i++;
        }
        else{
            aux[k] = v[j];
            j++;
        }
        k++;
    }

    while(i <= meio){
        aux[k] = v[i];
        i++;
        k++;
    }

    while(j <= fim){
        aux[k] = v[j];
        j++;
        k++;
    }
    //Copia os elementos que foram ordenados para o auxiliar
    for(int p = inicio; p <= fim; p++)
        v[p] = aux[p];
}

void mergeSort(int *aux, int *v, int inicio, int fim){
    if(inicio < fim){
        int meio = (inicio + fim) / 2;
        mergeSort(aux, v, inicio, meio);
        mergeSort(aux, v, meio + 1, fim);
        merge(aux, v, inicio, meio, fim);
    }
}
```



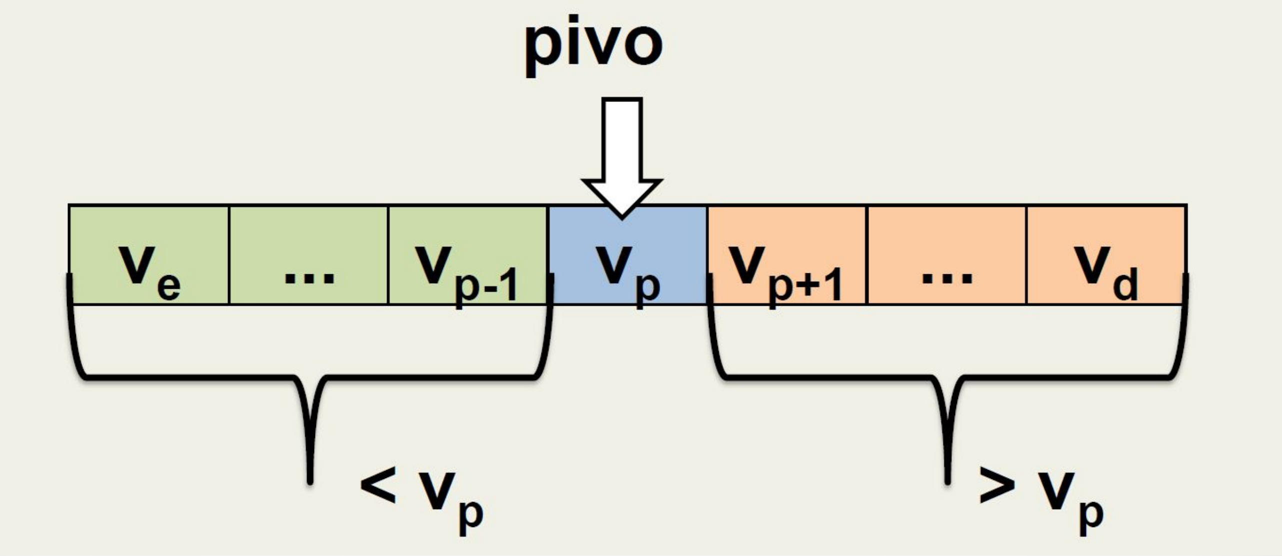
These numbers indicate the order in which steps are processed
38 27 43 3 9 82 10
1
38 27 43 3
9 82 10
2
12
38 27
43 3
9 82
10
3
7
13
17
38
27
43
3
9
82
10
4
5
8
9
14
15
27 38
3 43
9 82
10
6
10
16
18
3 27 38 43
9 10 82
11
19
3 9 10 27 38 43 82
20

Quick Sort

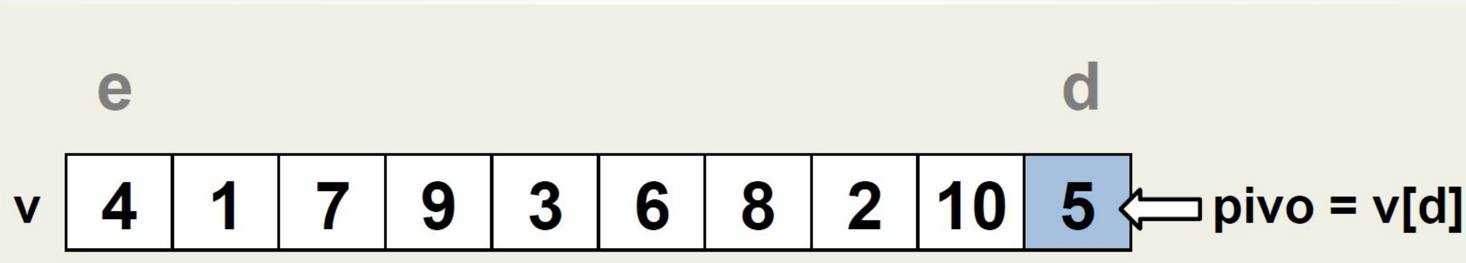
ordenação por partição

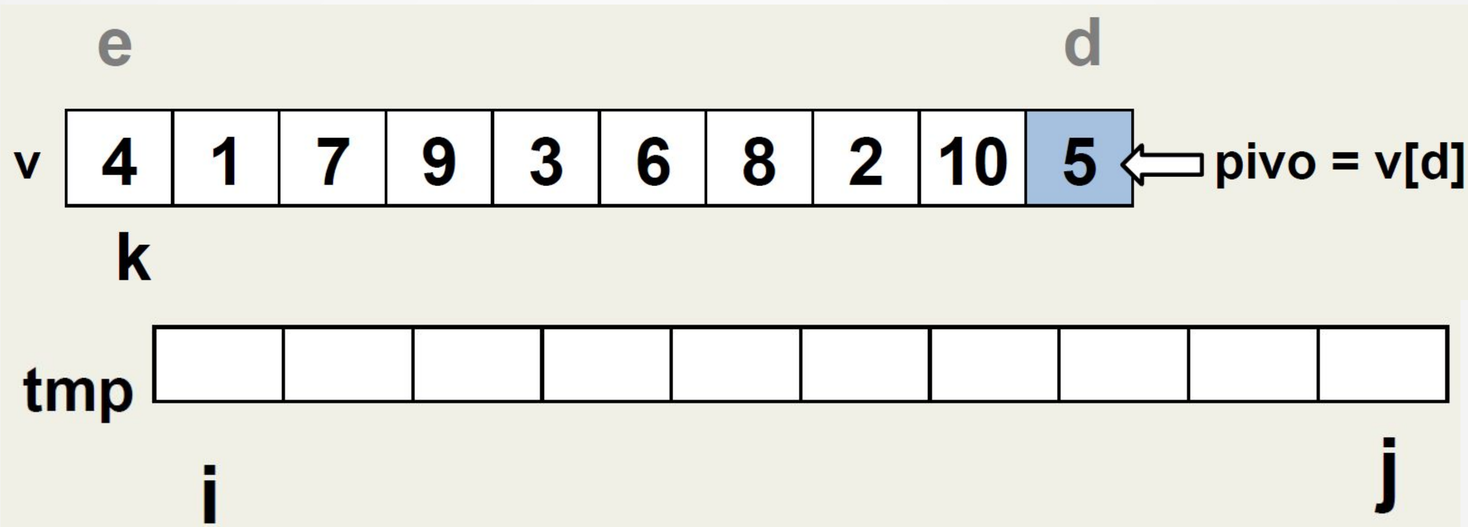
Idéia Geral

particionar e ordenar
(dividir e conquistar)

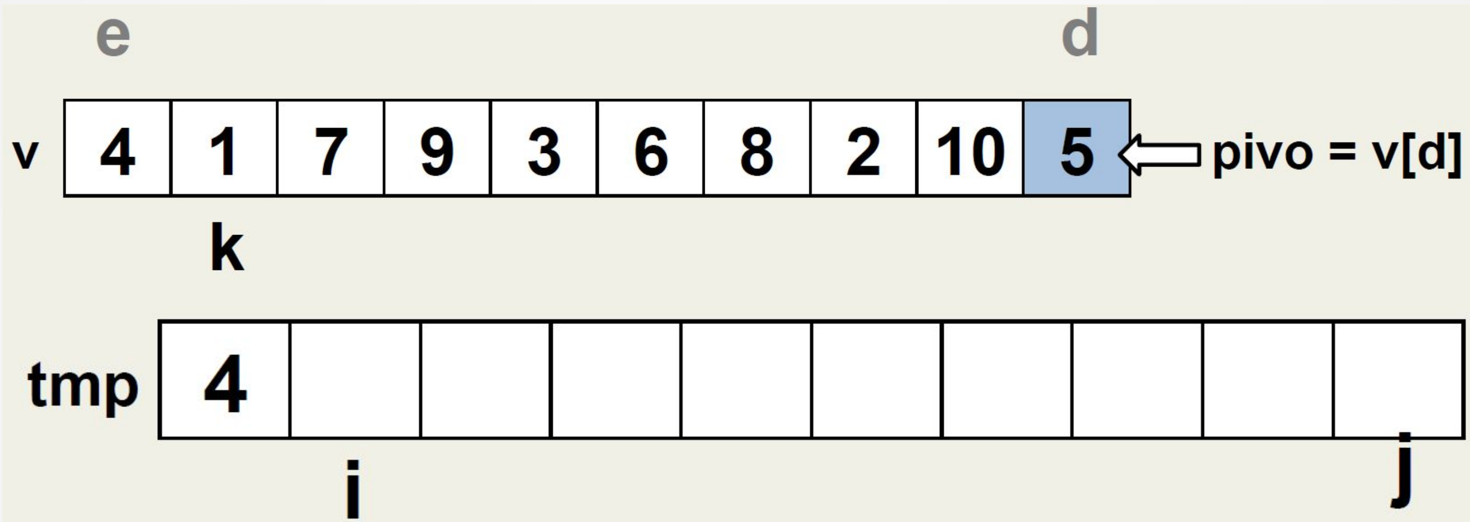


	e									d
v	4	1	7	9	3	6	8	2	10	5

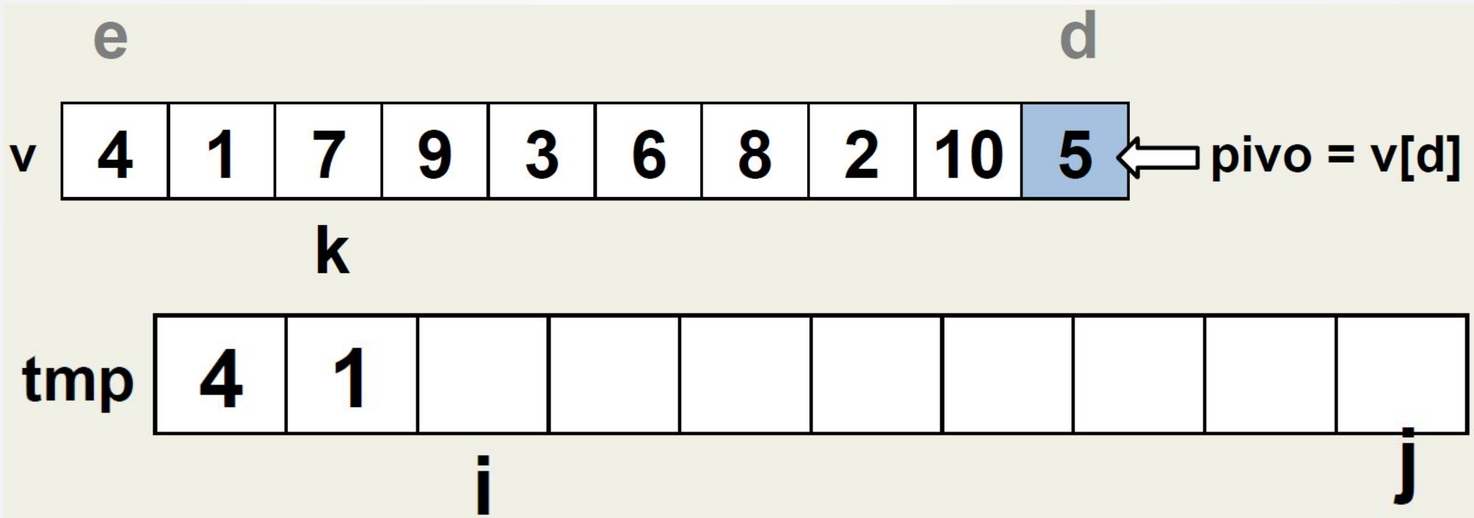

e
d
v
4
1
7
9
3
6
8
2
10
5
pivo = v[d]



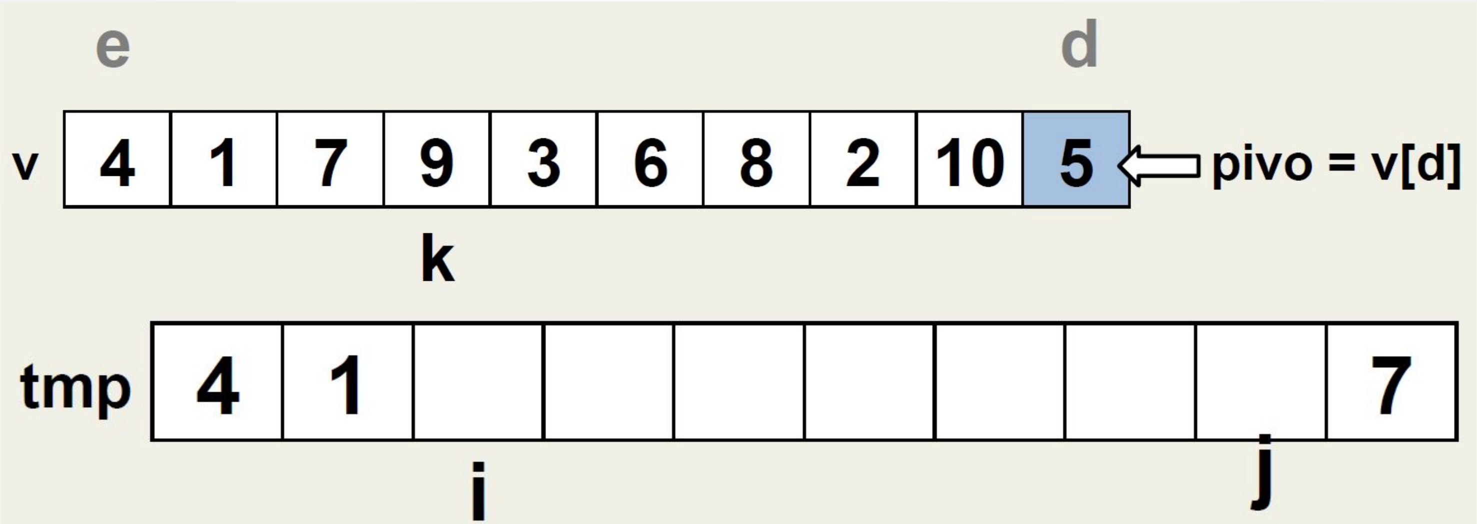

```
SE v[k] < pivo ENTÃO
    tmp[i] = v[k], i = i+1, k = k+1
SENÃO
    tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```



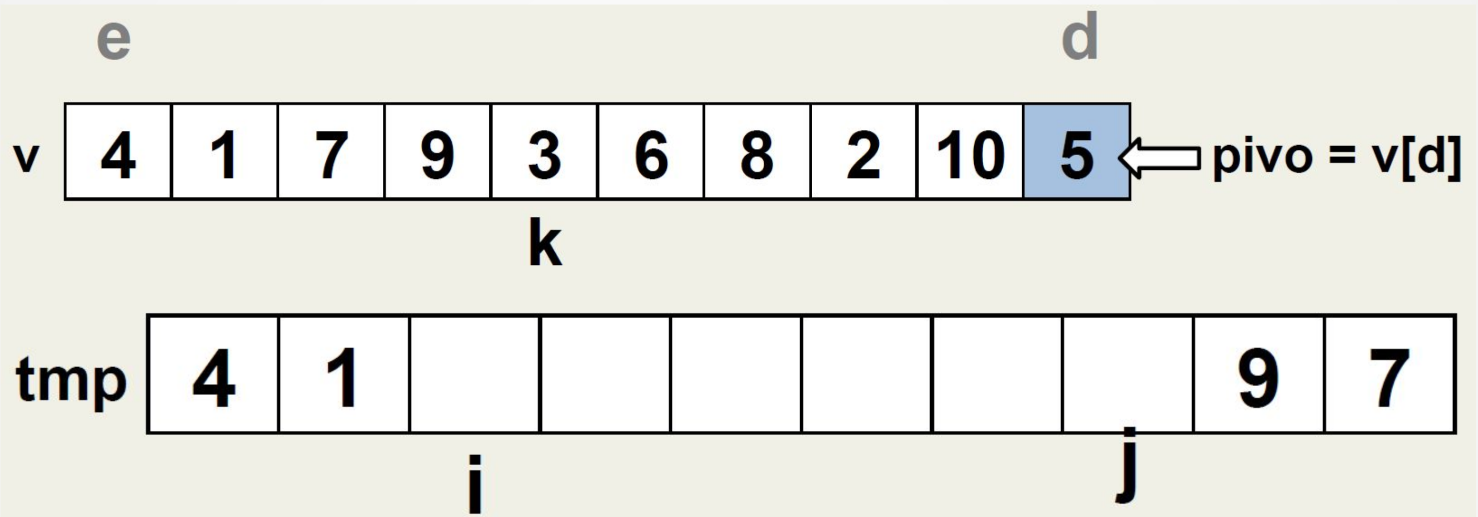

```
SE v[k] < pivo ENTÃO
    tmp[i] = v[k], i = i+1, k = k+1
SENÃO
    tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```



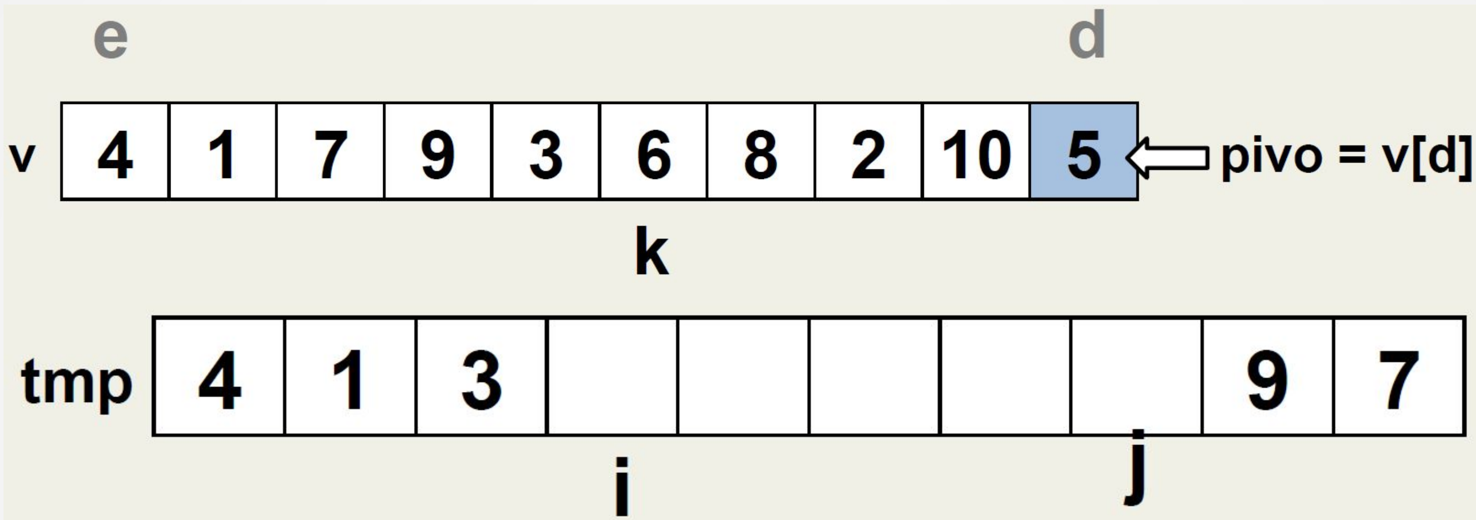

```
SE v[k] < pivo ENTÃO
      tmp[i] = v[k], i = i+1, k = k+1
SENÃO
      tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```



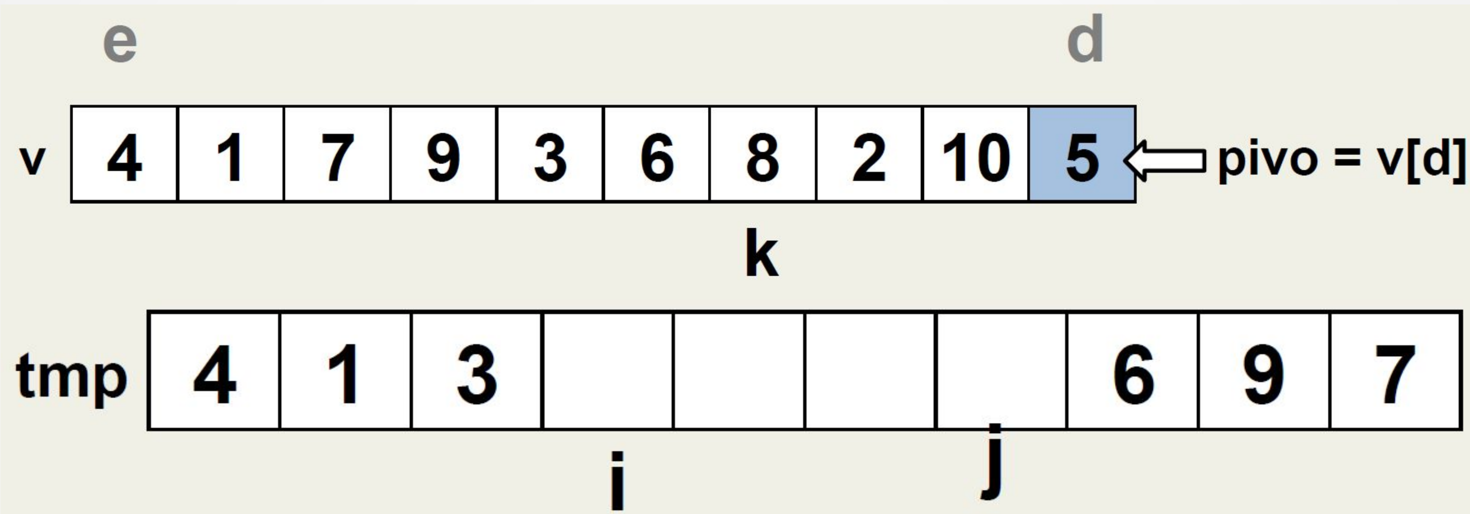

```
SE v[k] < pivo ENTÃO
    tmp[i] = v[k], i = i+1, k = k+1
SENÃO
    tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```



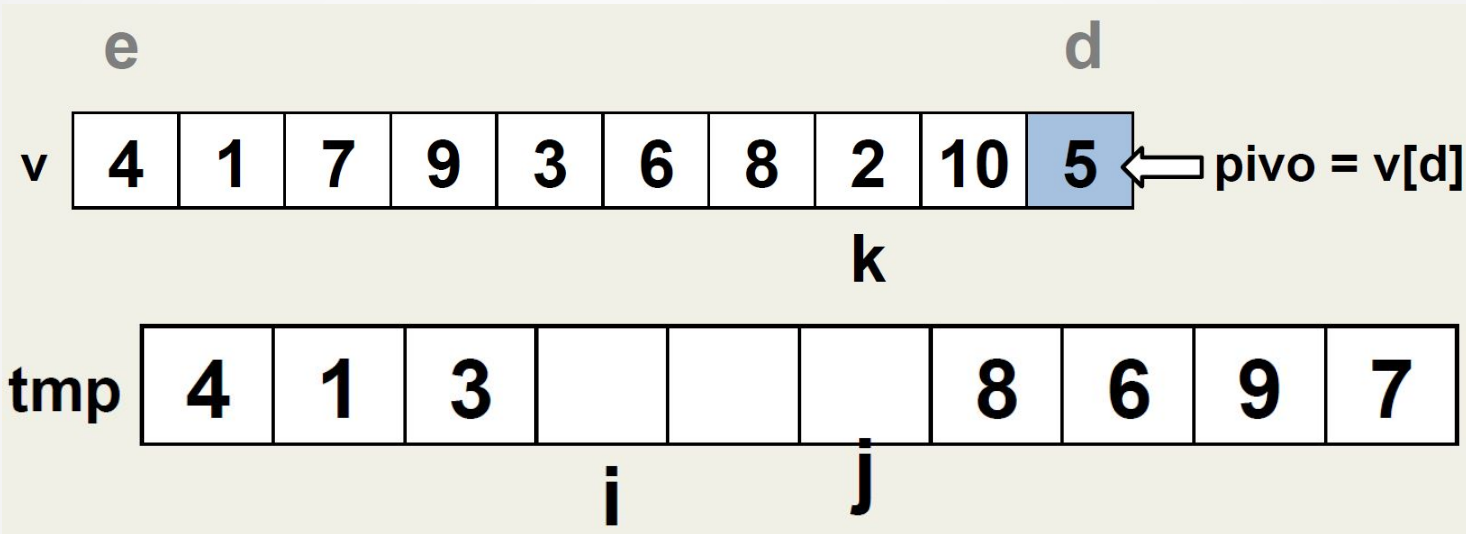
SE v[k] < pivo ENTÃO
    tmp[i] = v[k], i = i+1, k = k+1
SENÃO
    tmp[j] = v[k], j = j-1, k = k+1
FIM_SE



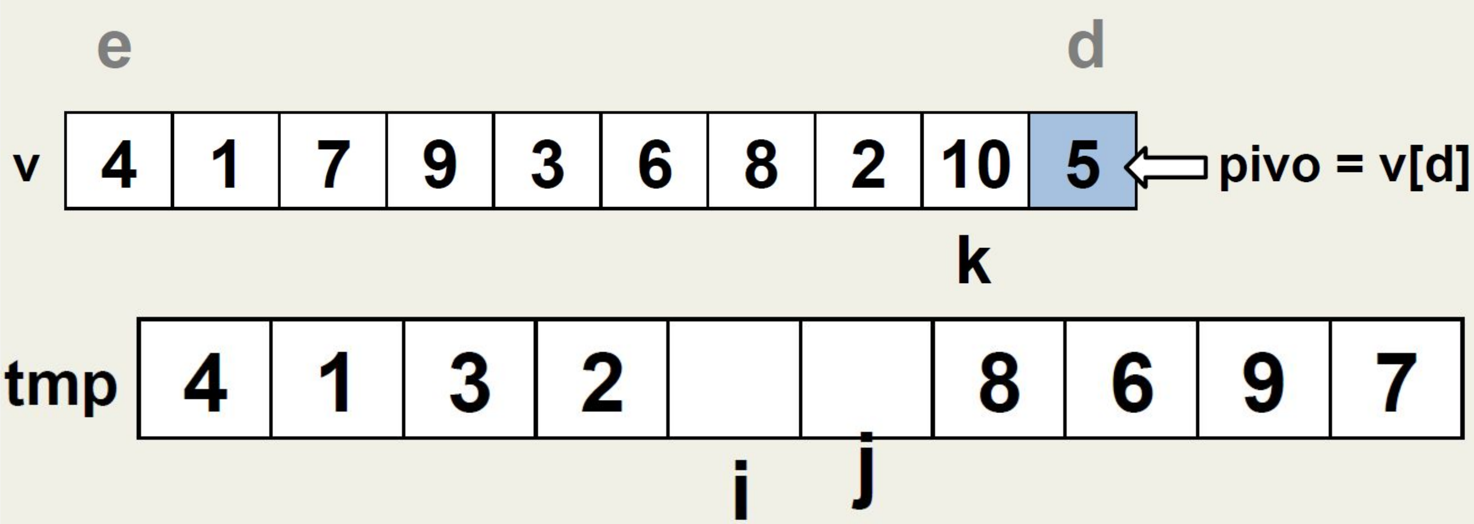

```
SE v[k] < pivo ENTÃO
     tmp[i] = v[k], i = i+1, k = k+1
SENÃO
     tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```



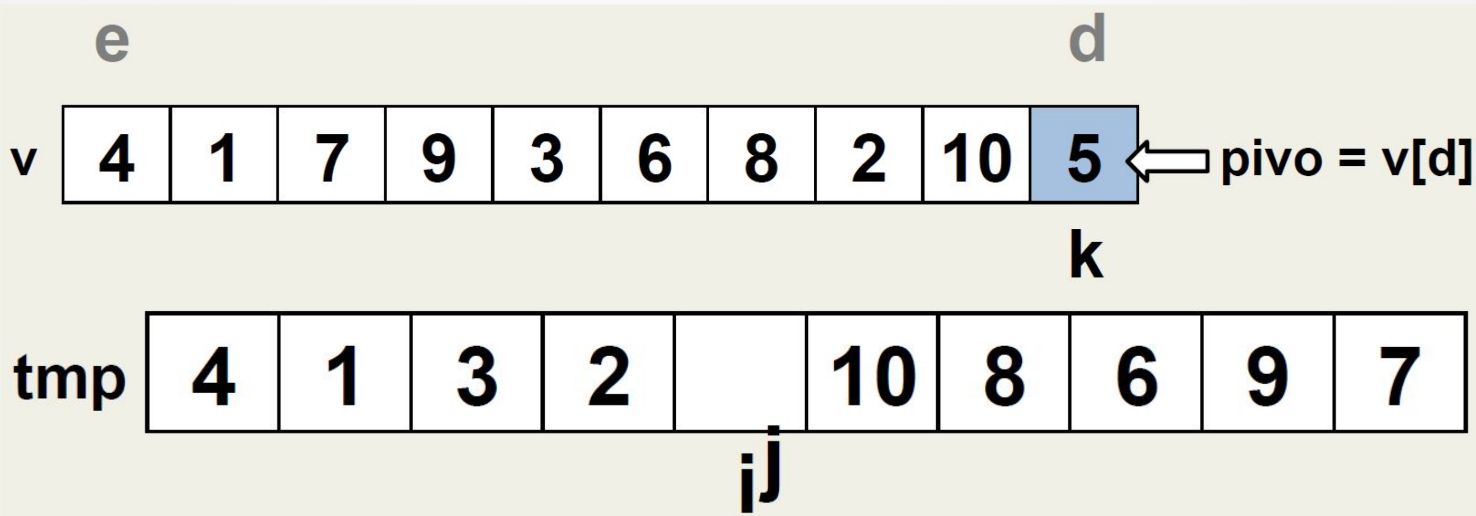

```
SE v[k] < pivo ENTÃO
      tmp[i] = v[k], i = i+1, k = k+1
SENÃO
      tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```




```
SE v[k] < pivo ENTÃO
    tmp[i] = v[k], i = i+1, k = k+1
SENÃO
    tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```

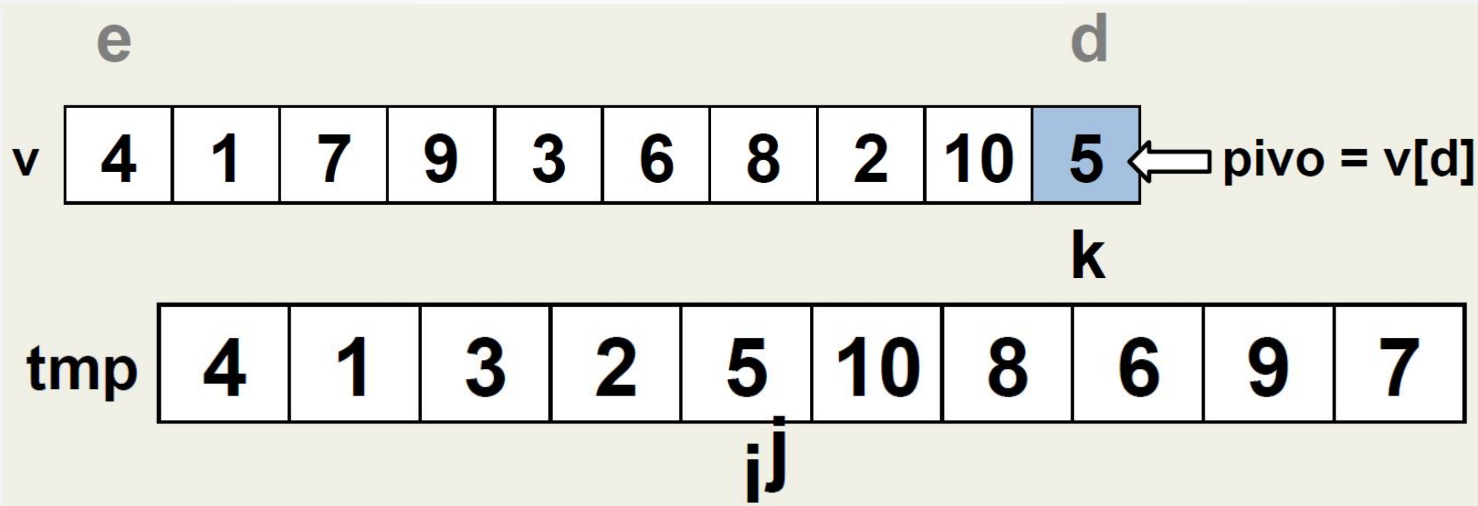



```
SE v[k] < pivo ENTÃO
    tmp[i] = v[k], i = i+1, k = k+1
SENÃO
    tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```




```
SE v[k] < pivo ENTÃO
    tmp[i] = v[k], i = i+1, k = k+1
SENÃO
    tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```

tmp[i] = pivo

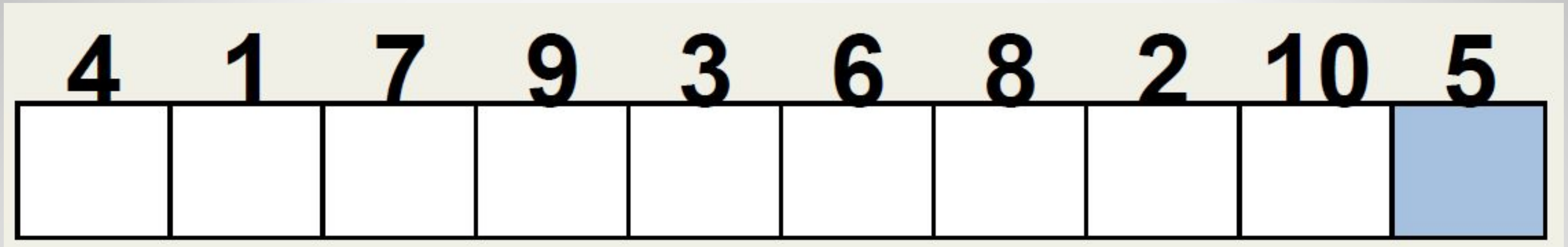



```
SE v[k] < pivo ENTÃO
      tmp[i] = v[k], i = i+1, k = k+1
SENÃO
      tmp[j] = v[k], j = j-1, k = k+1
FIM_SE
```

tmp[i] = pivo

```
Particionar( v[n], esquerda, direita ) :
  pivo = v[direita]
  i = 0, j = direita-esquerda, k = esquerda
  WHILE k < direita :
      IF v[k] < pivo :
            tmp[i] = v[k], i=i+1
      ELSE :
            tmp[j] = v[k], j=j-1
      END_IF
      k = k+1
  END_WHILE
  tmp[i] = pivo,
  COPY tmp in v
  RETORNE i
FIM
```

Já sei particionar para obter dois subvetores (com relação ao pivô), mas e ordenar?


4 1 7 9 3 6 8 2 10 5

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7

Considere agora dois subvetores

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2						

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2						

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2						
1	2	3	4						

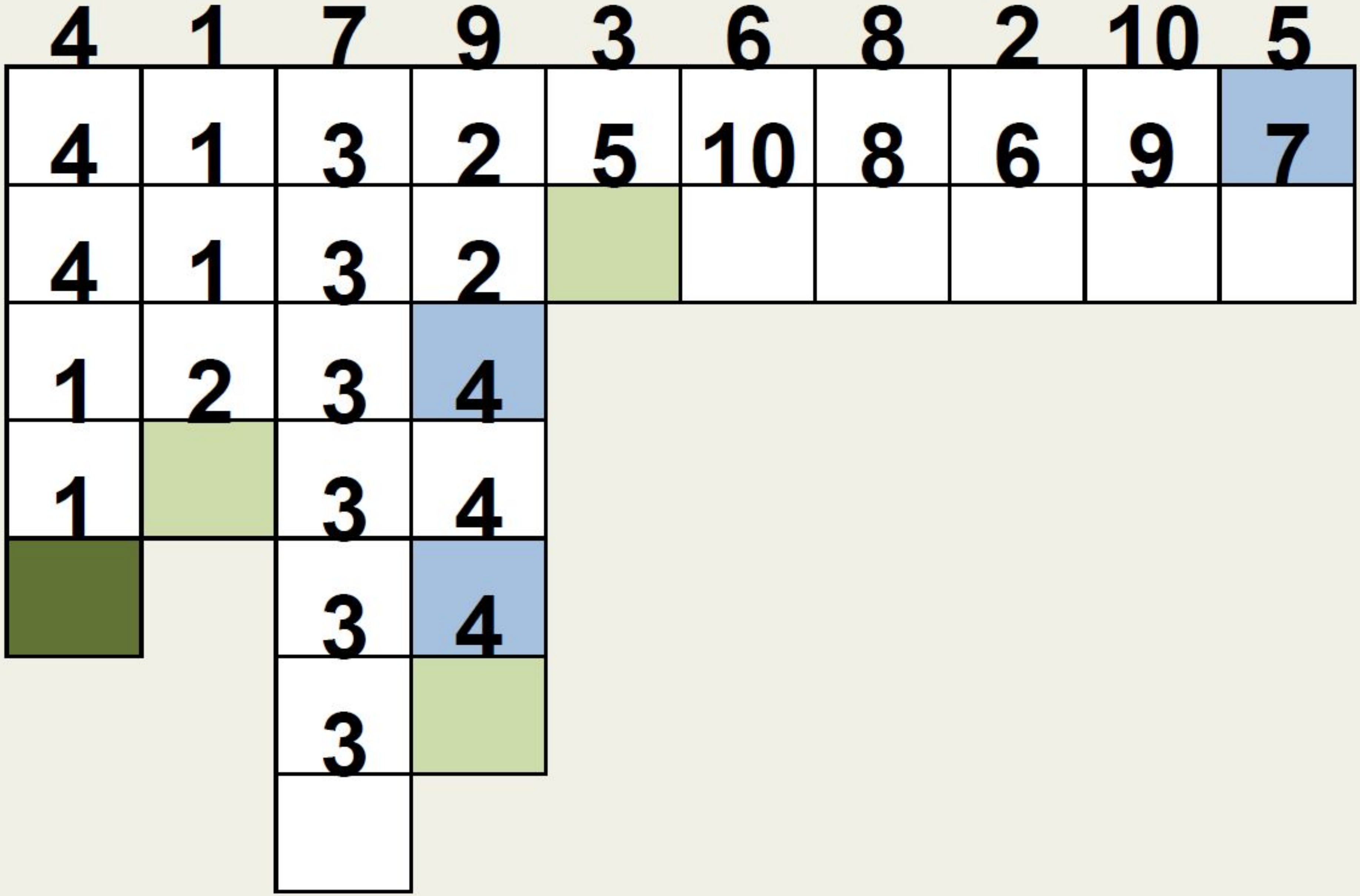
4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2						
1	2	3	4						
1									

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2						
1	2	3	4						
1									

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2						
1	2	3	4						
1		3	4						

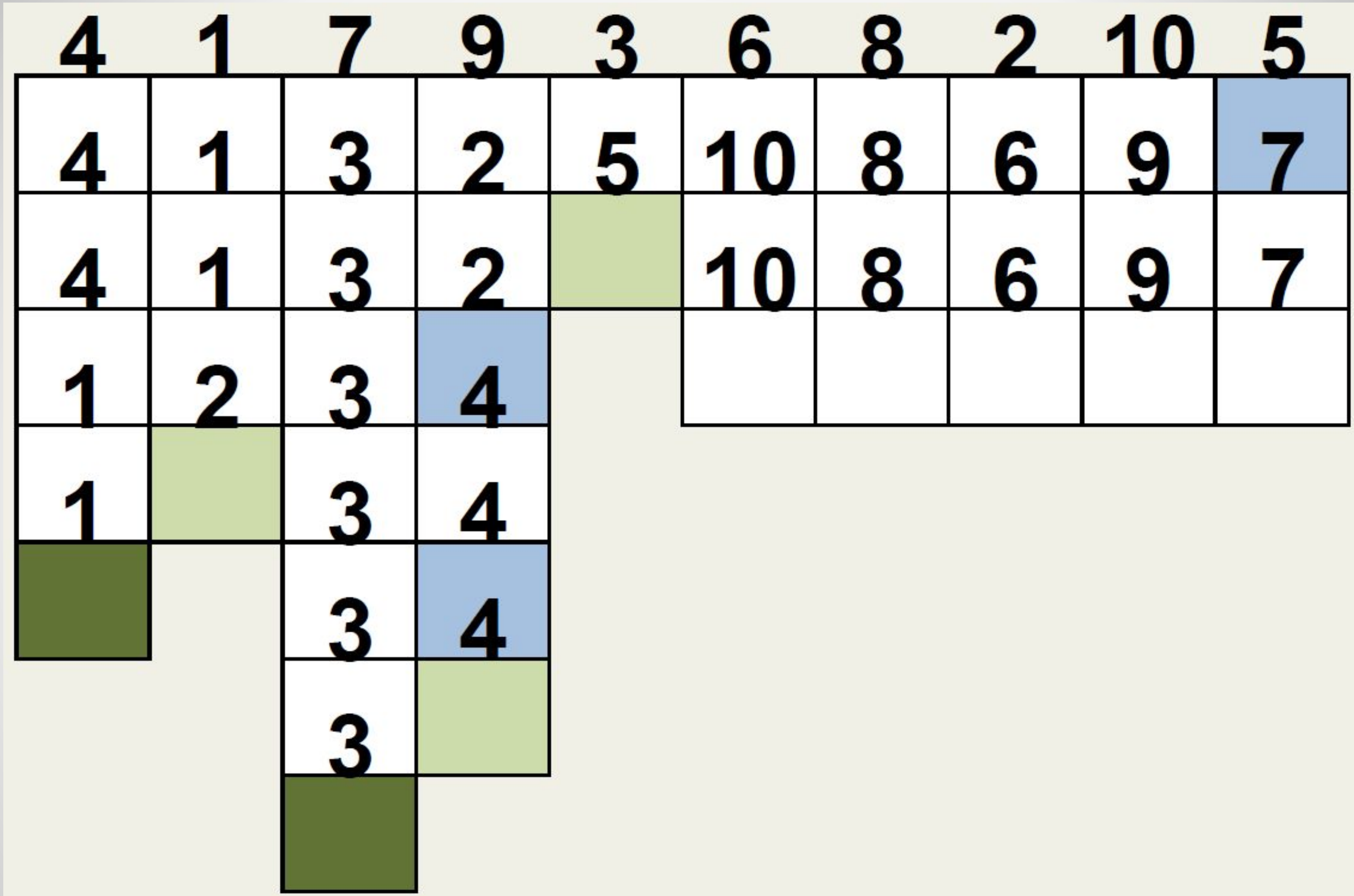
4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2						
1	2	3	4						
1		3	4						

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2						
1	2	3	4						
1		3	4						
		3	4						

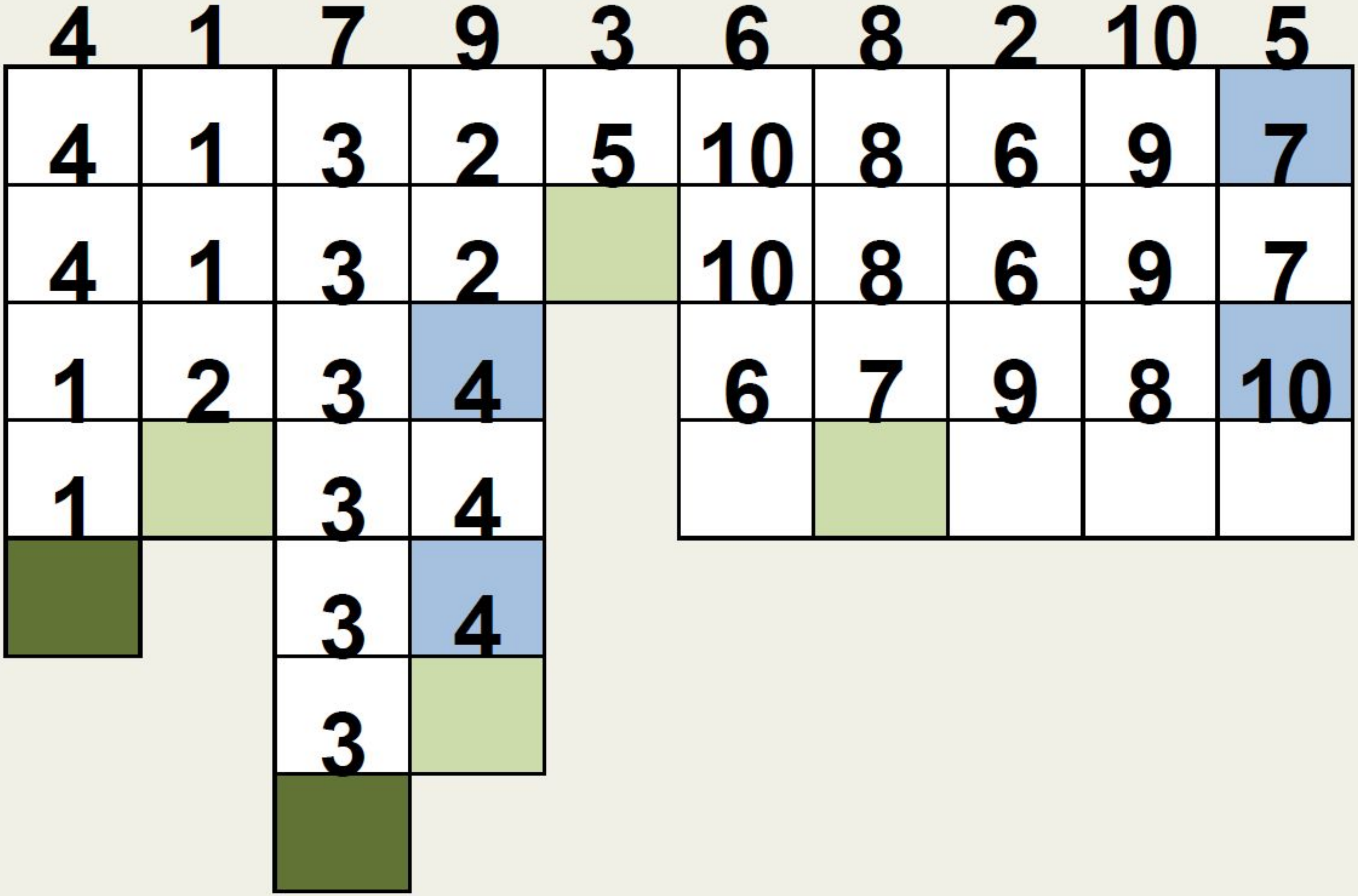


4 1 7 9 3 6 8 2 10 5
4 1 3 2 5 10 8 6 9 7
4 1 3 2
1 2 3 4
1 3 4
3 4
3

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2						
1	2	3	4						
1		3	4						
		3	4						
		3							


4 1 7 9 3 6 8 2 10 5
4 1 3 2 5 10 8 6 9 7
4 1 3 2 10 8 6 9 7
1 2 3 4
1 3 4
3 4
3

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2		10	8	6	9	7
1	2	3	4						
1		3	4						
		3	4						
		3							


4 1 7 9 3 6 8 2 10 5
4 1 3 2 5 10 8 6 9 7
4 1 3 2 10 8 6 9 7
1 2 3 4 6 7 9 8 10
1 3 4
3 4
3

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2		10	8	6	9	7
1	2	3	4		6	7	9	8	10
1		3	4		6				
		3	4						
		3							

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2		10	8	6	9	7
1	2	3	4		6	7	9	8	10
1		3	4		6				
		3	4						
		3							

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2		10	8	6	9	7
1	2	3	4		6	7	9	8	10
1		3	4		6		9	8	10
		3	4						
		3							

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2		10	8	6	9	7
1	2	3	4		6	7	9	8	10
1		3	4		6		9	8	10
		3	4						
		3							

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2		10	8	6	9	7
1	2	3	4		6	7	9	8	10
1		3	4		6		9	8	10
		3	4				9	8	10
		3							

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2		10	8	6	9	7
1	2	3	4		6	7	9	8	10
1		3	4		6		9	8	10
		3	4				9	8	10
		3					9	8	

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2		10	8	6	9	7
1	2	3	4		6	7	9	8	10
1		3	4		6		9	8	10
		3	4				9	8	10
		3					9	8	

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2		10	8	6	9	7
1	2	3	4		6	7	9	8	10
1		3	4		6		9	8	10
		3	4				9	8	10
		3					9	8	
							8	9	

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2		10	8	6	9	7
1	2	3	4		6	7	9	8	10
1		3	4		6		9	8	10
		3	4				9	8	10
		3					9	8	
							8	9	
								9	

4	1	7	9	3	6	8	2	10	5
4	1	3	2	5	10	8	6	9	7
4	1	3	2		10	8	6	9	7
1	2	3	4		6	7	9	8	10
1		3	4		6		9	8	10
		3	4				9	8	10
		3					9	8	
							8	9	
								9	

```c
void quickSort(int *v, int esquerda, int direita){

    int pivo;

    if (esquerda < direita){
        pivo = particionar(v, esquerda, direita);
        quickSort(v, esquerda, pivo-1);
        quickSort(v, pivo+1, direita);

    }
}
```

Complexidade

$T(n) = T(p) + T(n-p-1) + n$

- Melhor caso: $O(n*\log_2 n)$
- Pior caso: $O(n^2)$

Escolha para o pivô

- primeiro ou último elemento
- aleatório
- mediana

```
Particionar( v[n], esquerda, direita ) :
   i_mediana = AcharMediana( v )
   Swap v[i_mediana], v[direita]
   pivo = v[direita]
   i = esquerda, j = direita-1
   WHILE j ≥ i :
        WHILE v[i] < pivo && j ≥ i :
               i=i+1
        END_WHILE
        WHILE v[j] > pivo && j ≥ i :
               j = j-1
        END_WHILE
        IF j ≥ i :
               Swap v[i], v[j]
        END_IF
   END_WHILE
   Swap v[i], v[direita]
   Return i
END
```

Exercícios

1. Implemente e teste o algoritmo quicksort

2. Dado um vetor com números inteiros entre 0 e 9, encontre os dois números cuja soma é máxima. Esses números devem ser formados utilizando-se todos os elementos do vetor, e a diferença no número de dígitos não deve ser superior a 1.

Exemplos:

entrada: [3, 5, 1, 6, 8, 7]
saída: 863 e 751

entrada: [8, 1, 4, 5, 0, 3]
saída: 841 e 530

1. Repita o exercício 2, porém para encontrar os dois números cuja soma é mínima.