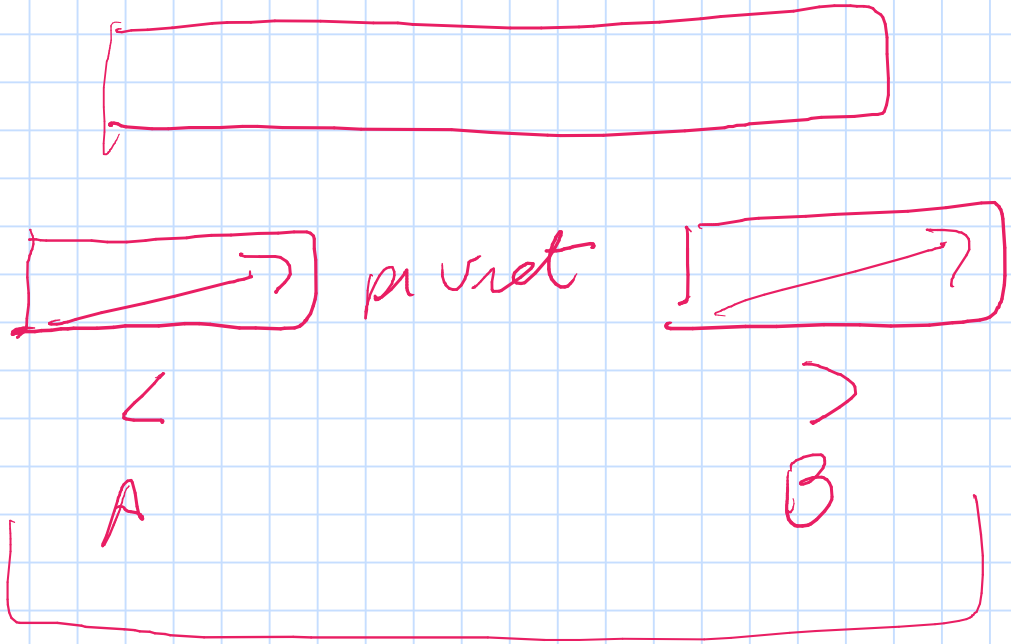
Quicksort (ordenação de A[p..r])

- Escolhe 1 elemento (pivot)
- **Divisão**: Particiona A de forma que
  - A[p..q-1] possui todos os elementos menores que o pivot
  - A[q+1..r] possui todos os elementos maiores que o pivot
    - ↳ Índice q é dado pelo procedimento de particionamento
    - ↳ Subvetor pode ser vazio.
- **Conquista**: Ordena recursivamente cada um dos subvetores obtidos
- **Combina**: Uma vez que um subvetor está ordenado com relação ao outro, não há necessidade de combinar



```
QuickSort(A[p..r])
1 Se (r ≤ p) Retorna
2 pivot ← A[r]
3 q ← Particiona(A,p,r,pivot)
4 QuickSort(A,p,q-1)
5 QuickSort(A,q+1,r)
```

o vetor A vai ser reorganizado da seguinte forma:

A[p..q-1] estarão os elementos menores que o pivot. Em A[q] estará o pivot e em A[q+1..r] estarão os elementos maiores que o pivot.

No linha 4, QuickSort é chamado recursivamente para o subvetor A[p..q-1].

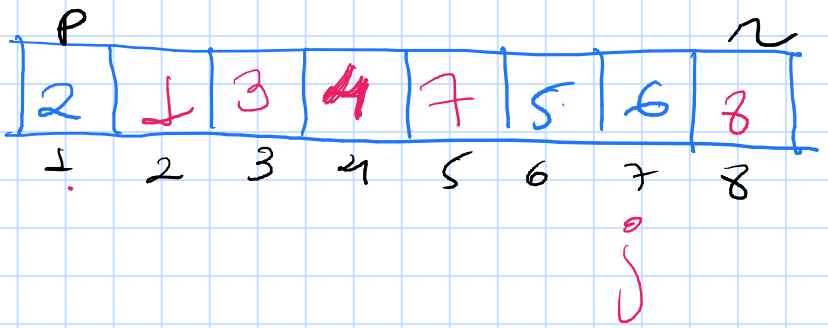
Pela hipótese de indução, A[p..q-1] é ordenado corretamente. O mesmo ocorre p/ A[q+1..r].

Como os elementos de A[p..q-1] são menores que A[q], eles também são menores que todos os elementos de A[q+1..r]. Logo, eles estão nas posições corretas para que o vetor inteiro fique ordenado.

Teorema: O algoritmo QuickSort ordena corretamente um vetor de n elementos recebido como entrada.

Sem vetor auxiliar:

```
Particiona (A, p, r, pivot)
    i ← p-1
    Para j ← p até r-1
        Se A[j] ≤ pivot
            i ← i+1
            A[i] ↔ A[j]
    A[i+1] ↔ A[r]
    Retorna i+1
```



complexidade? (vetor de n elementos) Θ(n)

Complexidade do Quicksort:

- Depende do balanceamento do particionamento (pivot)

- Pior caso?

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

$$= 0 + \dots + n-1 + n$$

$$= \frac{n \cdot (n+1)}{2} = O(n^2)$$

$$\Theta(n^2)$$

- Melhor caso? As partições têm sempre o mesmo tamanho.

$$T(n) = 2\,T\left(\frac{n}{2}\right) + \Theta(n)$$

$$\Theta(n \log n)$$

- caso médio

```
QuickSort (A, p, r)
    Se (r ≤ p) Retorna
    i ← Random (p, r)
    A[r] ↔ A[i]
    pivot ← A[r]
    q ← Particiona (A, p, r, pivot)
    QuickSort (A, p, q-1)
    QuickSort (A, q+1, r)
```

```
Particiona (A, p, r, pivot)
    i ← p-1
    Para j ← p até r-1
        Se A[j] ≤ pivot
            i ← i+1
            A[i] ↔ A[j]
    A[i+1] ↔ A[r]
    Retorna i+1
```

↳ Número de comparações do laço é suficiente para calcular o tempo de execução do algoritmo

↳ Contar a quantidade total de comparações

- Sejam $z_1, \dots, z_n$ os elementos de A, do menor para o maior

- Z_{ij} = conjunto dos elementos de z_i a z_j

Quando z_i e z_j são comparados?

b) Elementos são comparados apenas com o pivot e, para cada pivot, só ocorre 1 vez.

- X_{ij} = 1 se z_i é comparado com z_j em algum momento do algoritmo

↳ variável indicadora = 1 se ocorre, 0 c.c.

Número total de comparações:

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

Calcular o valor esperado dos dois lados:

Número médio de comparações.

$$E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right]$$

Pela linearidade do valor esperado:

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]$$

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} P\{z_i \text{ ser comparado com } z_j\}$$

OBS:

1) Todo mundo é comparado com o pivot

2) Uma vez que o pivot separa os elementos em dois conjuntos, nenhum elemento do primeiro é comparado com nenhum do segundo

3) Em cada conjunto, dois elementos serão comparados $\Leftrightarrow$ algum deles for pivot.

$P\{z_i \text{ ser comparado com } z_j\} =$

$P\{z_i \text{ ou } z_j \text{ serem (o primeiro) pivot em } z_{ij}$

$= P\{z_i \text{ ser o primeiro pivot em } z_{ij}\}$
$+$
$P\{z_j \text{ ser o primeiro pivot em } z_{ij}\}$

$= \frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1}$

$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$

$k = j - i$

$E[X] = \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k+1}$

$< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}$

Série harmônica

$= \sum_{i=1}^{n-1} O(\log n)$

$= O(n \log n)$