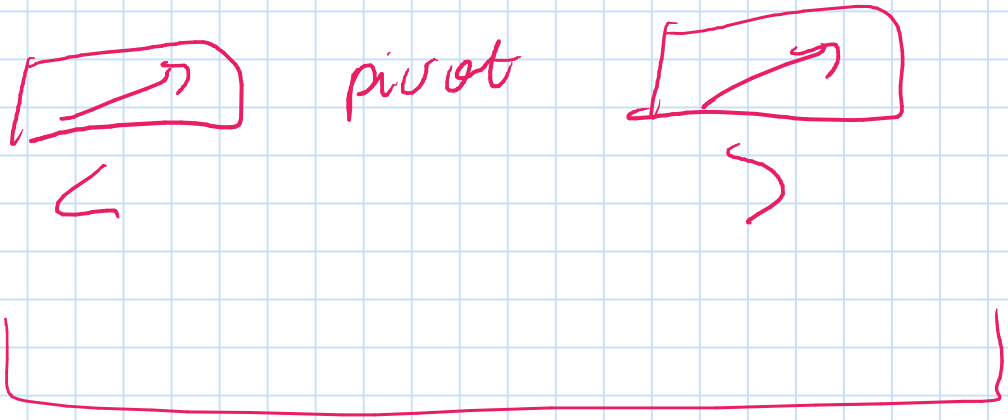
Quicksort (ordenação de A[p..r])

- 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 cada subvetor está ordenado, não há operação extra a realizar para concluir a ordenação.



```
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)
```

↳ Como implementar o procedimento Particiona utilizando vetor auxiliar?

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

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
```

$O(n)$

```
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)
```

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

Indução em n.

Base: n=0, n=1

Trivialmente ordenado, e o algoritmo Retorna.

Hipótese: QuickSort ordena corretamente um vetor com < n elementos.

Passo: Seja A um vetor de n > 1 elementos. Na linha 2, pivot recebe o elemento da última posição do vetor, e o Particiona reorganiza o vetor de forma que A[p..q-1] possui apenas elementos menores que o pivot, A[q+1..r] elementos

```
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)
```

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

Na linha 3, o algoritmo Particiona rearranja o trecho A[p..r] de modo que os elementos maiores que o pivot fiquem à direita do pivot e os demais à esquerda. Ao final, o pivot fica na posição q.

Na linha 4, o algoritmo é chamado recursivamente para ordenar o trecho de p a q-1.

Por hipótese de indução, essa porção do vetor é ordenada corretamente. O mesmo ocorre na linha 5 para A[q+1..r].

Como os elementos de A[p..q-1] são menores que o pivot, também são menores que os elementos de A[q+1..r], e portanto os menores elementos do vetor já se encontram na posição final da ordenação do vetor inteiro.

O argumento análogo pode ser utilizado para mostrar que os elementos de $A[q+1 \ldots r]$ estão posicionados corretamente e A está completamente ordenado. ☐

Complexidade do Quicksort:

→ Depende do balanceamento do particionamento (pivot)

- Pior caso?

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

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

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

- Melhor caso?

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

(H) $n \log n$

- 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 $z_1, \ldots, z_n$ 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?

↳ Elementos são comparados apenas com o pivot e, para cada pivot, apenas 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:

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

Calculando o valor esperado dos dois lados:

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

↳ número médio de comparações.

Pela linearidade do valor esp.

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

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

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

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

$$= O(n \log n)$$