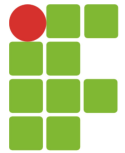
Ordenação: Quicksort

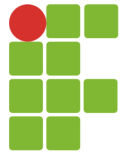
Estrutura de Dados e Algoritmos – Ciência da Computação

Prof. Daniel Saad Nogueira Nunes

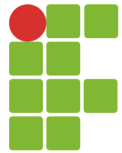
IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário



Sumário



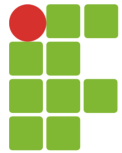
Quicksort

Quicksort

O Quicksort se baseia na escolha de um pivô. Após escolhido este pivô, a sequência original é particionada em três partes:

1. Elementos menores do que o pivô;
2. Pivô;
3. Elementos maiores ou iguais do que o pivô;

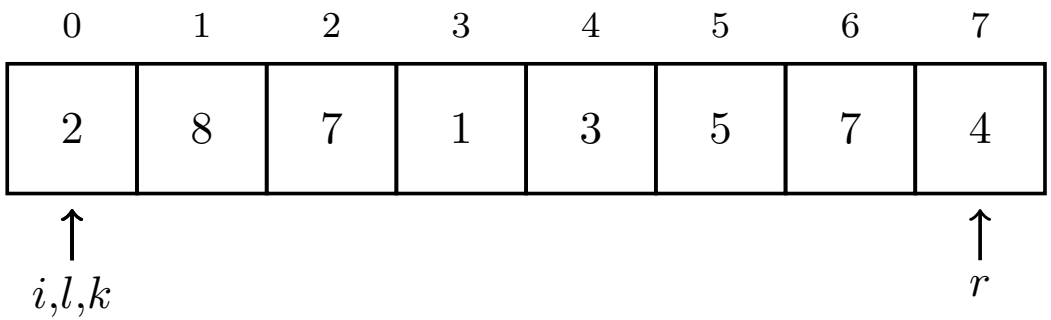
O procedimento é aplicado recursivamente na primeira e última partes.

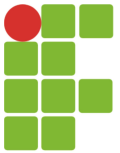


Quicksort

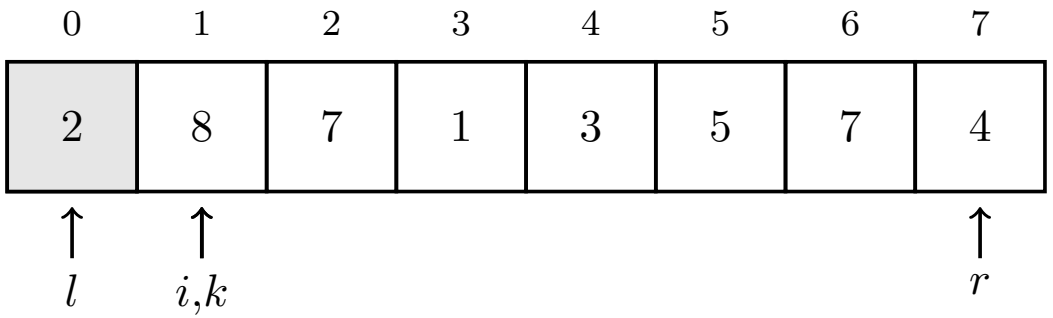
- Tome o pivô como o último elemento do vetor.
- Para fazer esse particionamento, não é necessário ter vetores extras. É possível realizar *in-place*.
- Mantemos um índice k para o final da primeira parte, os elementos menores ou iguais ao pivô.
- Sempre que encontramos um elemento $V[i]$ menor do que o pivô, trocamos $V[i]$ com $V[k]$ e aumentamos k. Isto é, o elemento é colocado imediatamente ao final da primeira partição e o índice k é incrementado, aumentando o tamanho da primeira parte.
- Ao final da classificação, trocamos o elemento que está logo após o término da primeira parte com o pivô.

Quicksort

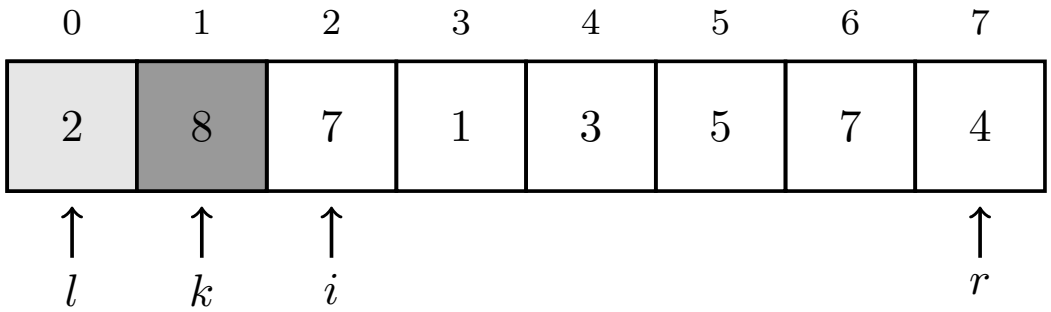




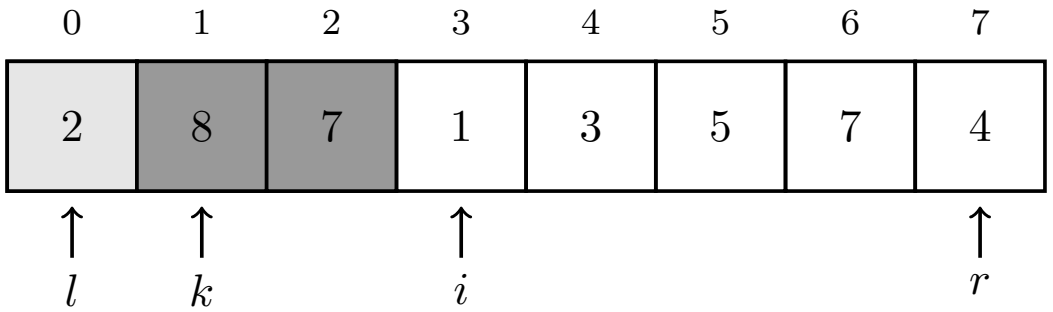
Quicksort



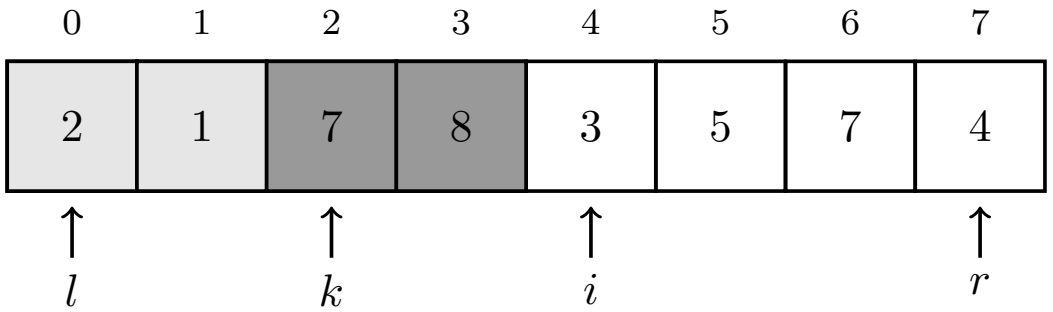
Quicksort



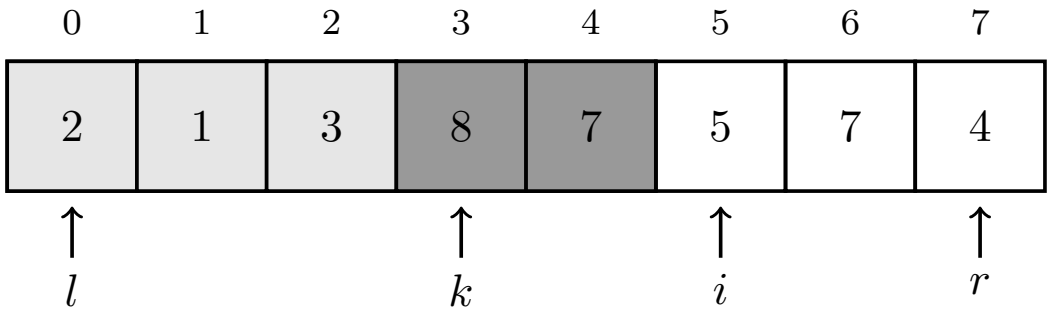
Quicksort



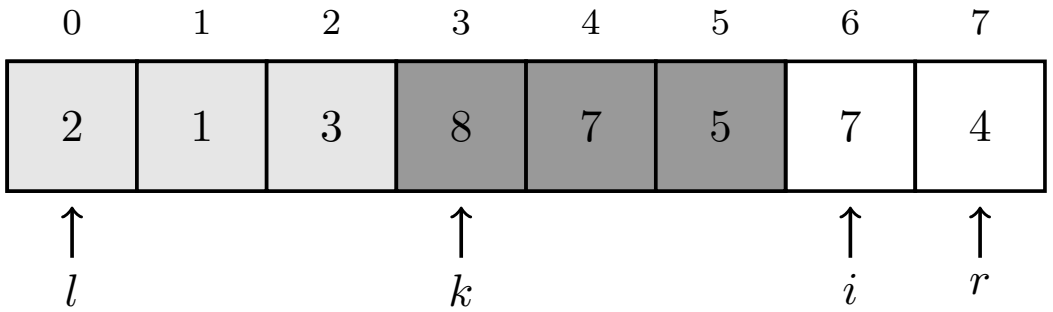
Quicksort



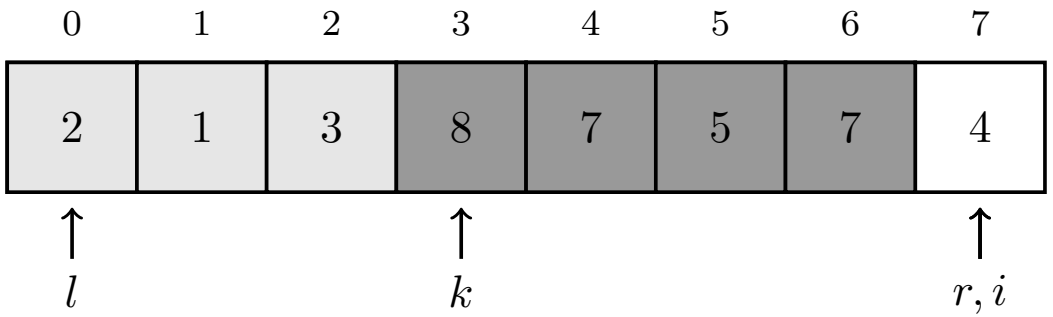
Quicksort



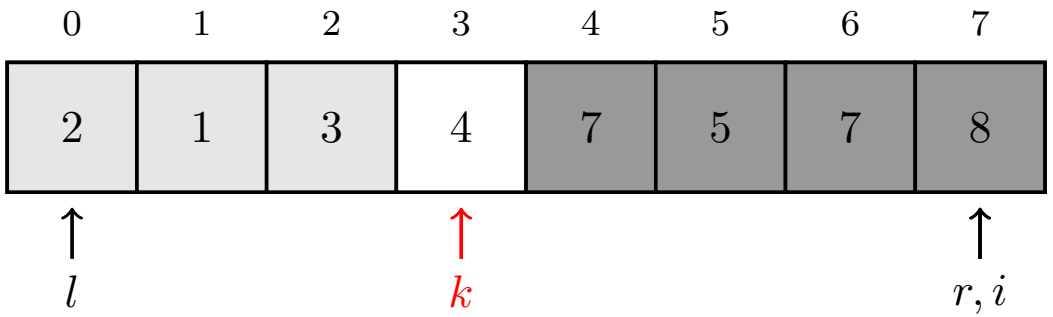
Quicksort

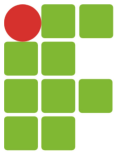


Quicksort

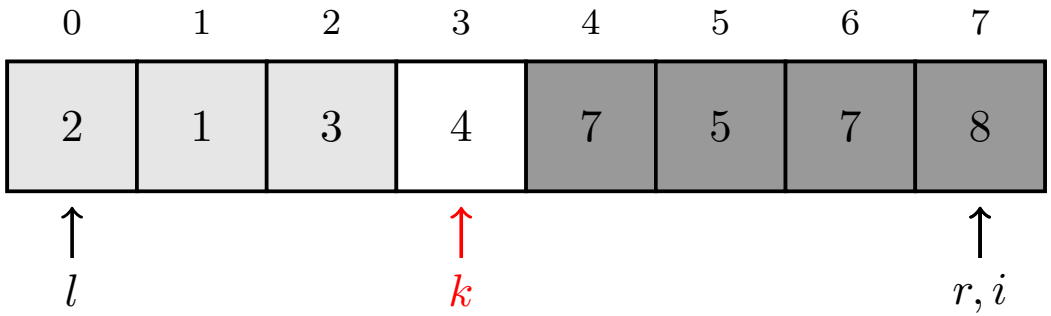


Quicksort

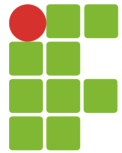




Quicksort



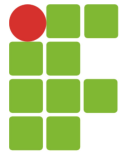
Recursivamente ordenamos V no intervalo $[0, 2]$ e no intervalo $[4, 7]$.



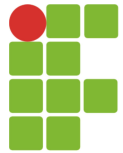
Quicksort

Escolha do pivô

- A priori qualquer elemento pode ser escolhido como pivô.
- Na implementação a ser apresentada, o pivô é o último elemento do vetor.
- Isso pode ser alterado simplesmente trocando as posições do pivô de escolha com o último elemento.

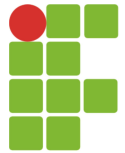


Sumário



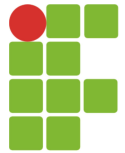
Quicksort

```
23 void quick_sort(int *v, size_t size){
24     quick_sort_helper(v,0,size-1);
25 }
```



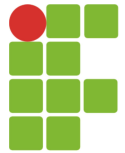
Quicksort

```
15 void quick_sort_helper(int*v, int l,int r){
16     if (l < r){
17         size_t pos = partition(v,l,r,r);
18         quick_sort_helper(v, l, pos - 1);
19         quick_sort_helper(v, pos + 1, r);
20     }
21 }
```

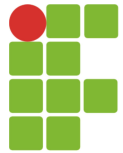


Quicksort

```
1  static size_t partition(int *vet, int esq,int dir,int pivot){
2      size_t pos, i;
3      swap(vet, pivot, dir);
4      pos = esq;
5      for(i = esq; i < dir; i++){
6          if (vet[i] < vet[dir]){
7              swap(vet, i, pos);
8              pos++;
9          }
10     }
11     swap(vet, dir, pos);
12     return pos;
13 }
```



Sumário



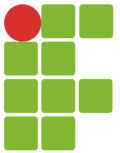
Quicksort: análise

Análise: pior caso

O procedimento de particionamento custa tempo $\Theta(n)$, logo, a relação de recorrência do Quicksort, no pior caso, corresponde à:

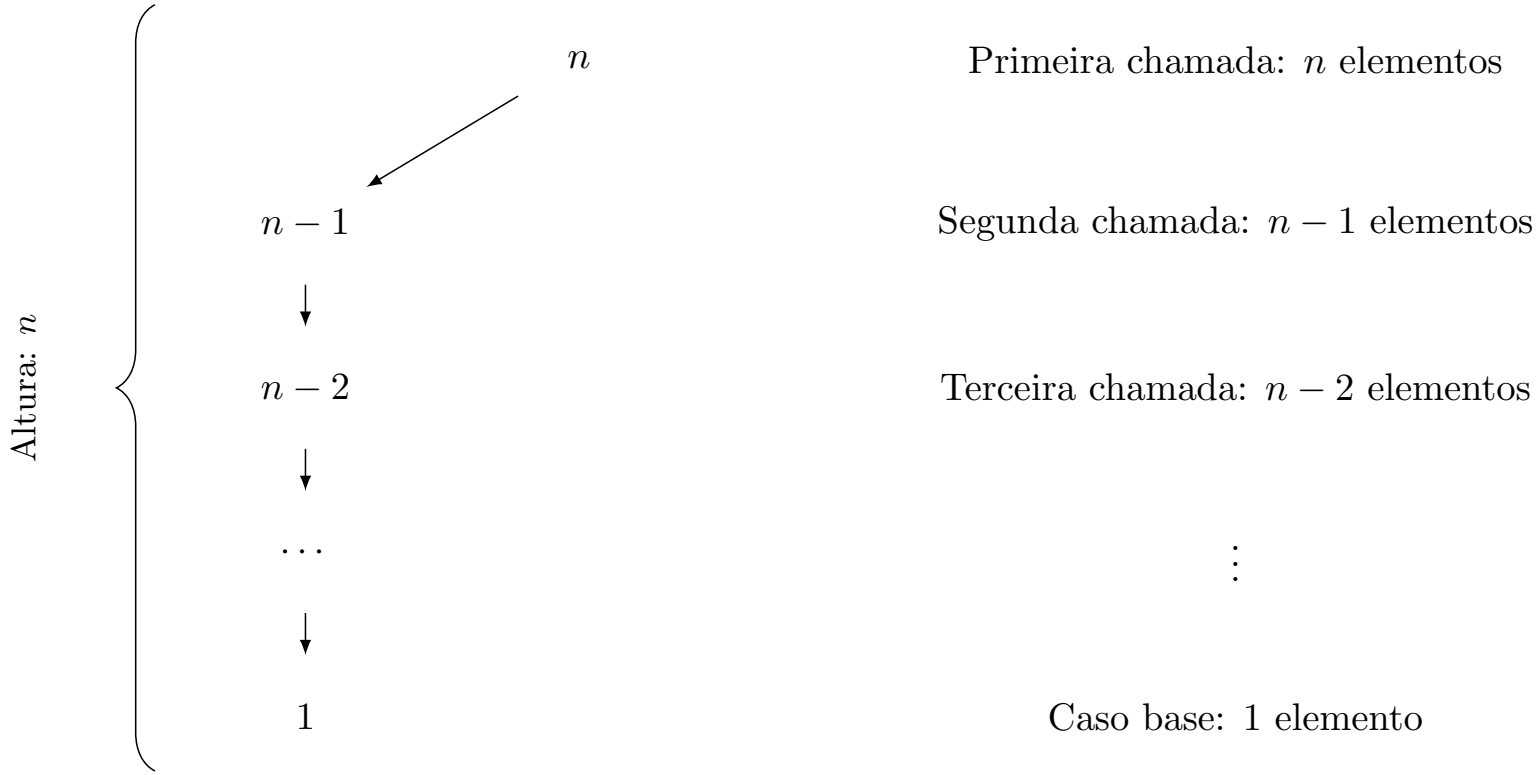
$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(n-1) + \Theta(n), & n > 1 \end{cases}$$

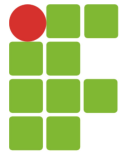
Portanto, o Quicksort leva tempo $\Theta(n^2)$ no pior caso.



Quicksort: análise

Análise: pior caso





Quicksort: análise

Análise: caso-médio

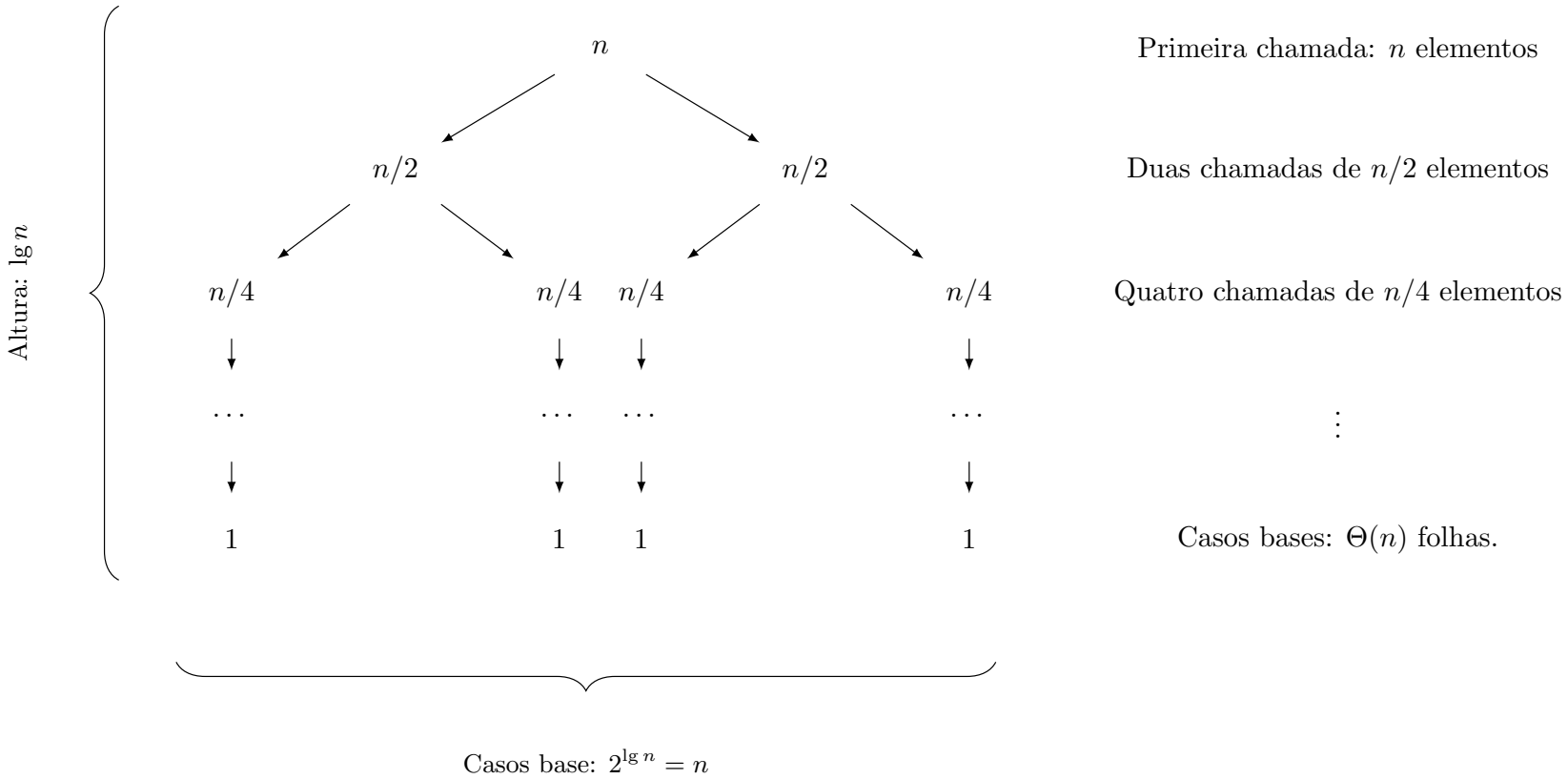
Contanto, no caso médio, o Quicksort divide as partições de modo em que a primeira e a última partição tenham tamanhos similares, o que leva a uma relação de recorrência que se aproxima de:

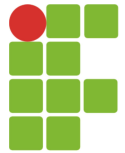
$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ 2T(n/2) + \Theta(n), & n > 1 \end{cases}$$

Nessa ocasião, o Quicksort leva tempo $\Theta(n \lg n)$.

Quicksort: análise

Análise: caso-médio





Quicksort: análise

Estabilidade

- O quicksort não é estável, pois, o procedimento de particionamento, pode trocar a posição relativa de elementos iguais.

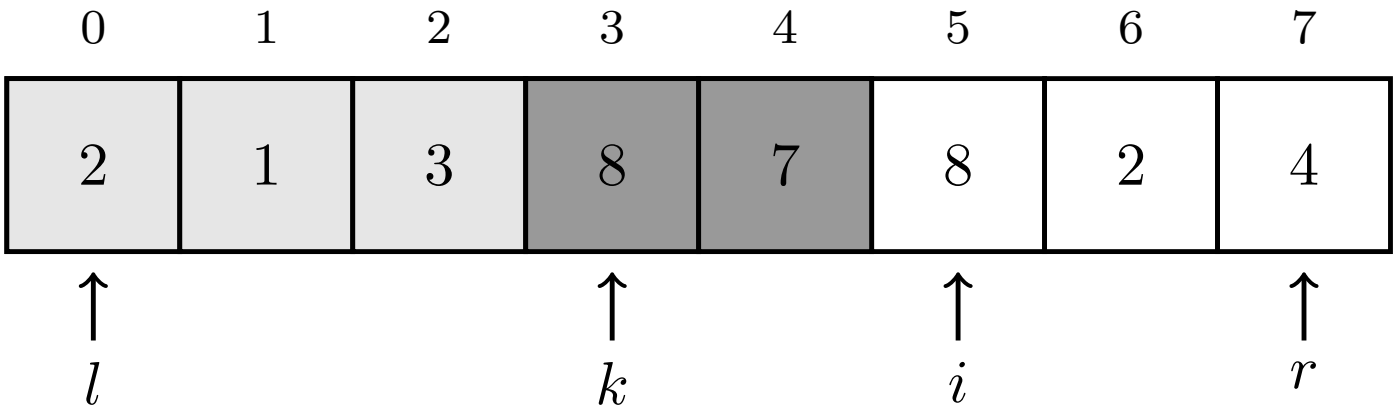
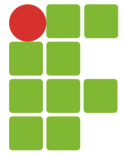


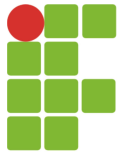
Figura: A posição dos 8s seria invertida.



Quicksort: análise

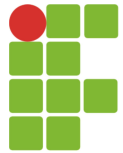
In-place

- O quicksort não é *in-place* por a pilha de chamadas recursivas consome mais de $\Theta(1)$ de memória.



Análise Quicksort

In-place	Estável
✗	✗



Quicksort: análise

Observação

- Pior caso em $\Theta(n^2)$;
- Caso médio em $\Theta(n \lg n)$;
- Recursivo;
- Não é in-place por causa da pilha de chamadas recursivas;
- Eficiente na prática;