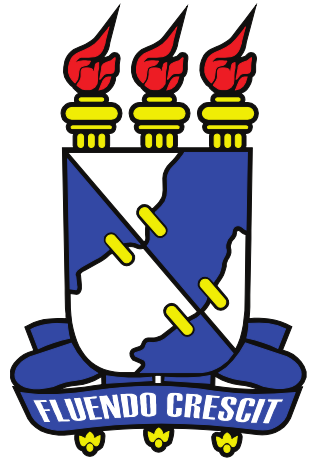


Projeto e Análise de Algoritmos

Quicksort

Bruno Prado

Departamento de Computação / UFS

Introdução

- O que é Quicksort?
  - Criado por Tony Hoare em 1960
  - Estratégia de Divisão e Conquista
  - Funciona particionando o conjunto de dados

Introdução

- Três passos para divisão e conquista em algoritmos
  - Dividir o problema em subproblemas
    - Instâncias menores e mais simples
  - Resolver os subproblemas
    - Mais simples de serem resolvidos
  - Combinar as soluções parciais obtidas para gerar a solução completa
    - Etapa de conquista

Introdução

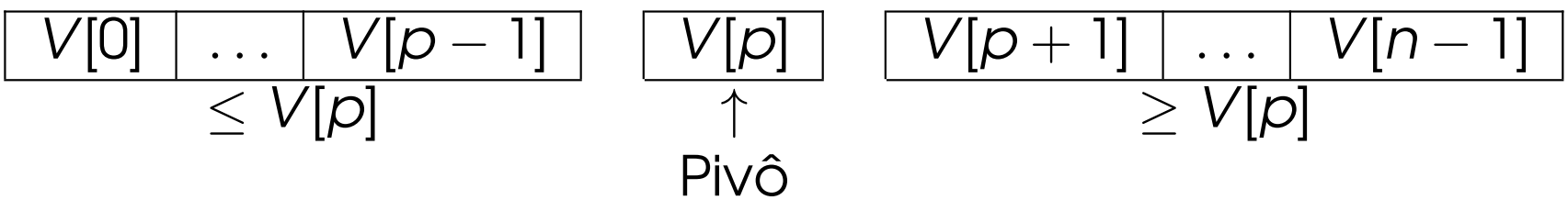
- Vantagens
  - Paralelismo
    - Problema é dividido em partes que podem ser resolvidas separadamente
  - Eficiência algorítmica
    - Complexidade $O(n \log n)$
  - Acesso a memória mais eficiente
    - Dados cabem na memória cache
  - Controle de arredondamento mais preciso
    - Os resultados são combinados ao invés de iterados

Introdução

- Desvantagens
  - Recursão
    - Utilização de pilha que é limitada
    - Menor desempenho por conta do acesso constante a memória
  - Escolha dos casos base
    - Boas escolhas evitam processamento desnecessário para entradas pequenas
  - Subproblemas repetidos
    - É possível obter subproblemas idênticos que vão se calculados repetidamente

Quicksort

- Conceitos chave
  - Estratégia de Divisão e Conquista
  - Particionamento do vetor
  - Escolha do elemento pivô



Quicksort

- Implementação em C
  - Procedimento quicksort

```
void quicksort(int V(), int inicio, int fim) {
        if(inicio < fim) {
                int pivo = particionar(V, inicio, fim);
                quicksort(V, inicio, pivo - 1);
                quicksort(V, pivo + 1, fim);
        }
}
```

Quicksort

- Implementação em C
  - Função particionar

```
int particionar(int V(), int inicio, int fim) {
        int pivo = V(fim);
        int i = inicio - 1, j;
        for(j = inicio; j < fim; j++) {
                if(V(j) <= pivo) {
                        i = i + 1;
                        trocar(&V(i), &V(j));
                }
        }
        trocar(&V(i + 1), &V(fim));
        return i + 1;
}
```

Quicksort

- Simulação do particionamento

```
int particionar(int V(), int inicio, int fim) {
        int pivo = V(fim);
        int i = inicio - 1, j;
        for(j = inicio; j < fim; j++) {
                if(V(j) <= pivo) {
                        i = i + 1;
                        trocar(&V(i), &V(j));
                }
        }
        trocar(&V(i + 1), &V(fim));
        return i + 1;
}
```

i j

2	8	7	3	5	**4**
0	1	2	3	4	5

Quicksort

- Simulação do particionamento

```
int particionar(int V(), int inicio, int fim) {
        int pivo = V(fim);
        int i = inicio - 1, j;
        for(j = inicio; j < fim; j++) {
                if(V(j) <= pivo) {
                        i = i + 1;
                        trocar(&V(i), &V(j));
                }
        }
        trocar(&V(i + 1), &V(fim));
        return i + 1;
}
```

i, j

2	8	7	3	5	**4**
0	1	2	3	4	5

Quicksort

- Simulação do particionamento

```
int particionar(int V(), int inicio, int fim) {
        int pivo = V(fim);
        int i = inicio - 1, j;
        for(j = inicio; j < fim; j++) {
                if(V(j) <= pivo) {
                        i = i + 1;
                        trocar(&V(i), &V(j));
                }
        }
        trocar(&V(i + 1), &V(fim));
        return i + 1;
}
```

	i		j			
	2	8	7	3	5	**4**
	0	1	2	3	4	5

Quicksort

- Simulação do particionamento

```
int particionar(int V(), int inicio, int fim) {
        int pivo = V(fim);
        int i = inicio - 1, j;
        for(j = inicio; j < fim; j++) {
                if(V(j) <= pivo) {
                        i = i + 1;
                        trocar(&V(i), &V(j));
                }
        }
        trocar(&V(i + 1), &V(fim));
        return i + 1;
}
```

i		j			
2	8	7	3	5	**4**
0	1	2	3	4	5

Quicksort

- Simulação do particionamento

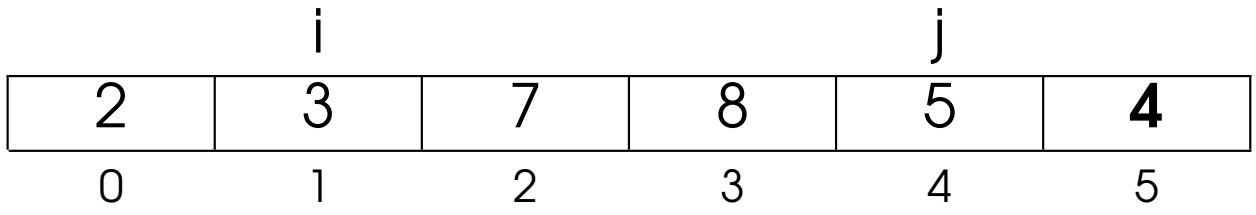
```
int particionar(int V(), int inicio, int fim) {
        int pivo = V(fim);
        int i = inicio - 1, j;
        for(j = inicio; j < fim; j++) {
                if(V(j) <= pivo) {
                        i = i + 1;
                        trocar(&V(i), &V(j));
                }
        }
        trocar(&V(i + 1), &V(fim));
        return i + 1;
}
```

		i		j	
2	3	7	8	5	**4**
0	1	2	3	4	5

Quicksort

- Simulação do particionamento

```
int particionar(int V(), int inicio, int fim) {
        int pivo = V(fim);
        int i = inicio - 1, j;
        for(j = inicio; j < fim; j++) {
                if(V(j) <= pivo) {
                        i = i + 1;
                        trocar(&V(i), &V(j));
                }
        }
        trocar(&V(i + 1), &V(fim));
        return i + 1;
}
```



Quicksort

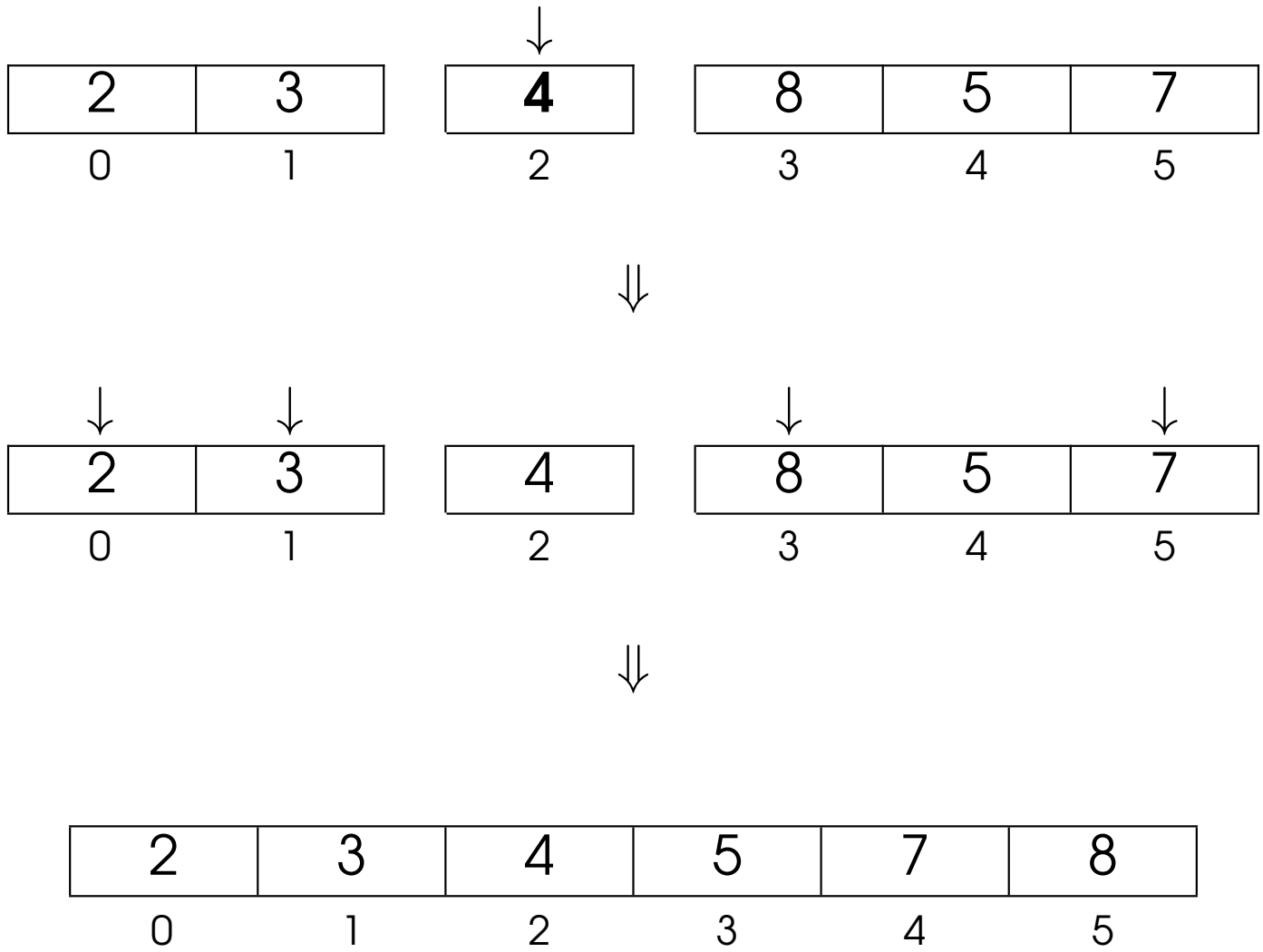
- Simulação do particionamento

```
int particionar(int V(), int inicio, int fim) {
        int pivo = V(fim);
        int i = inicio - 1, j;
        for(j = inicio; j < fim; j++) {
                if(V(j) <= pivo) {
                        i = i + 1;
                        trocar(&V(i), &V(j));
                }
        }
        trocar(&V(i + 1), &V(fim));
        return i + 1;
}
```

	i	i + 1		j	fim
2	3	**4**	8	5	7
0	1	2	3	4	5

Quicksort

- Papel do particionamento no Quicksort



Quicksort

- Papel do particionamento no Quicksort
  - É a própria ordenação
  - Qual a melhor forma de particionar? E qual a pior?
  - Várias estratégias

Quicksort

- Estratégias de escolha de pivô
  - Randômico

```
int randomico(int V(), int inicio, int fim) {
        int i = inicio + (rand() % (fim - inicio + 1));
        trocar(&V(fim), &V(i));
        return particionar(V, inicio, fim);
}
```

Quicksort

- Estratégias de escolha de pivô
  - Pivô Mediana de 3
    - Escolher três elementos aplicando alguma heurística
    - Pivô é a mediana destes 3 elementos
    - Implementação livre

Quicksort

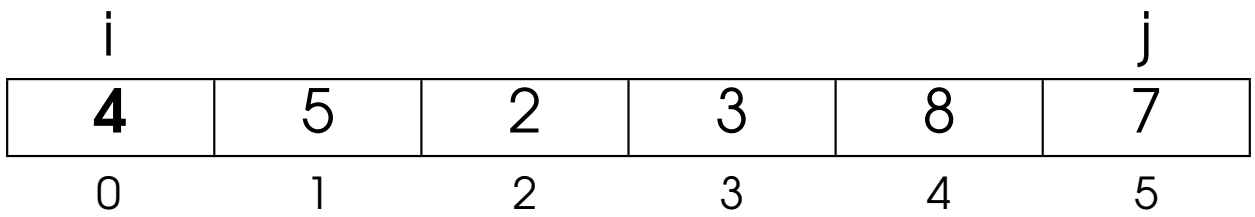
- Particionamento Hoare

```
int hoare(int V(), int inicio, int fim) {
        int pivo = V(inicio);
        int i = inicio;
        int j = fim;
        while(i < j) {
                while(j > i && V(j) >= pivo) j–;
                while(i < j && V(i) < pivo) i++;
                if(i < j) trocar(&V(i), &V(j));
        }
        return j;
}
```

Quicksort

- Particionamento Hoare

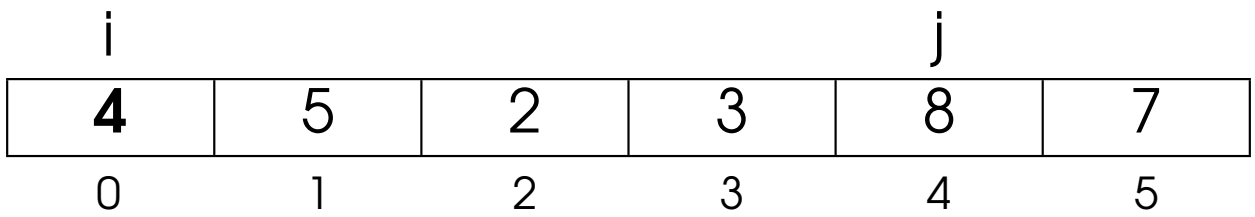
```
int hoare(int V(), int inicio, int fim) {
        int pivo = V(inicio);
        int i = inicio;
        int j = fim;
        while(i < j) {
                while(j > i && V(j) >= pivo) j–;
                while(i < j && V(i) < pivo) i++;
                if(i < j) trocar(&V(i), &V(j));
        }
        return j;
}
```



Quicksort

- Particionamento Hoare

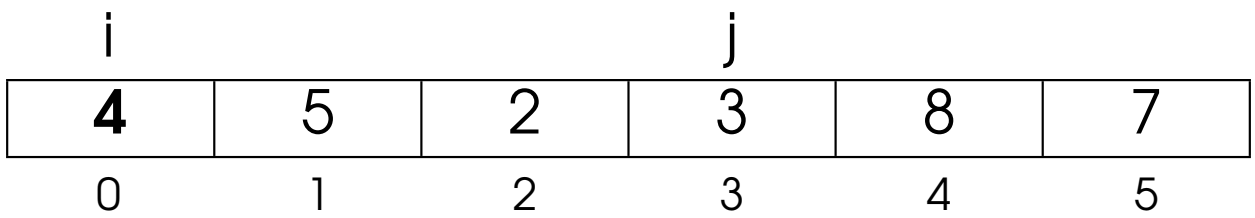
```
int hoare(int V(), int inicio, int fim) {
        int pivo = V(inicio);
        int i = inicio;
        int j = fim;
        while(i < j) {
                while(j > i && V(j) >= pivo) j–;
                while(i < j && V(i) < pivo) i++;
                if(i < j) trocar(&V(i), &V(j));
        }
        return j;
}
```



Quicksort

- Particionamento Hoare

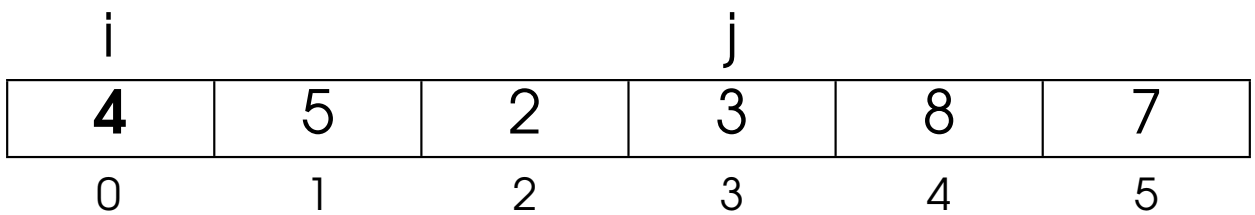
```
int hoare(int V(), int inicio, int fim) {
        int pivo = V(inicio);
        int i = inicio;
        int j = fim;
        while(i < j) {
                while(j > i && V(j) >= pivo) j–;
                while(i < j && V(i) < pivo) i++;
                if(i < j) trocar(&V(i), &V(j));
        }
        return j;
}
```



Quicksort

- Particionamento Hoare

```
int hoare(int V(), int inicio, int fim) {
        int pivo = V(inicio);
        int i = inicio;
        int j = fim;
        while(i < j) {
                while(j > i && V(j) >= pivo) j–;
                while(i < j && V(i) < pivo) i++;
                if(i < j) trocar(&V(i), &V(j));
        }
        return j;
}
```



Quicksort

- Particionamento Hoare

```
int hoare(int V(), int inicio, int fim) {
        int pivo = V(inicio);
        int i = inicio;
        int j = fim;
        while(i < j) {
                while(j > i && V(j) >= pivo) j–;
                while(i < j && V(i) < pivo) i++;
                if(i < j) trocar(&V(i), &V(j));
        }
        return j;
}
```

	i			j		
	3	5	2	**4**	8	7
	0	1	2	3	4	5

Quicksort

- Particionamento Hoare

```
int hoare(int V(), int inicio, int fim) {
        int pivo = V(inicio);
        int i = inicio;
        int j = fim;
        while(i < j) {
                while(j > i && V(j) >= pivo) j–;
                while(i < j && V(i) < pivo) i++;
                if(i < j) trocar(&V(i), &V(j));
        }
        return j;
}
```

i			j		
3	5	2	**4**	8	7
0	1	2	3	4	5

Quicksort

- Particionamento Hoare

```
int hoare(int V(), int inicio, int fim) {
        int pivo = V(inicio);
        int i = inicio;
        int j = fim;
        while(i < j) {
                while(j > i && V(j) >= pivo) j–;
                while(i < j && V(i) < pivo) i++;
                if(i < j) trocar(&V(i), &V(j));
        }
        return j;
}
```

	i	j			
3	5	2	**4**	8	7
0	1	2	3	4	5

Quicksort

- Particionamento Hoare

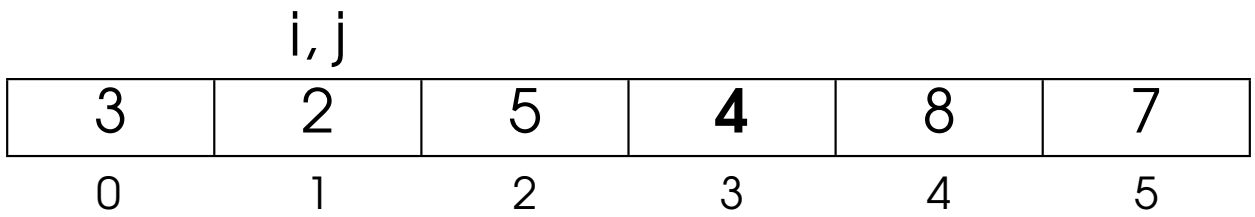
```
int hoare(int V(), int inicio, int fim) {
        int pivo = V(inicio);
        int i = inicio;
        int j = fim;
        while(i < j) {
                while(j > i && V(j) >= pivo) j–;
                while(i < j && V(i) < pivo) i++;
                if(i < j) trocar(&V(i), &V(j));
        }
        return j;
}
```

	i	j			
3	2	5	**4**	8	7
0	1	2	3	4	5

Quicksort

- Particionamento Hoare

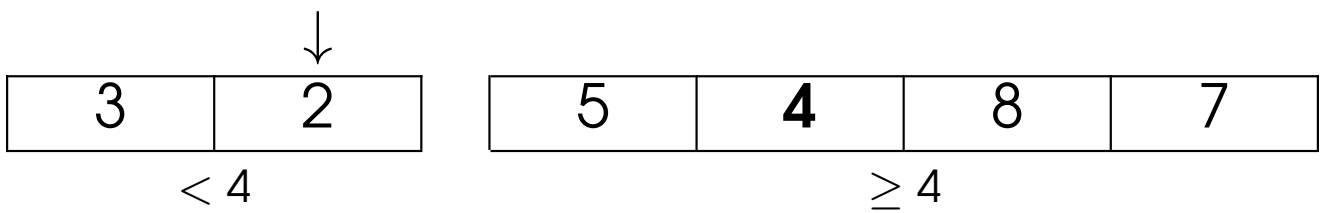
```
int hoare(int V(), int inicio, int fim) {
        int pivo = V(inicio);
        int i = inicio;
        int j = fim;
        while(i < j) {
                while(j > i && V(j) >= pivo) j–;
                while(i < j && V(i) < pivo) i++;
                if(i < j) trocar(&V(i), &V(j));
        }
        return j;
}
```



Quicksort

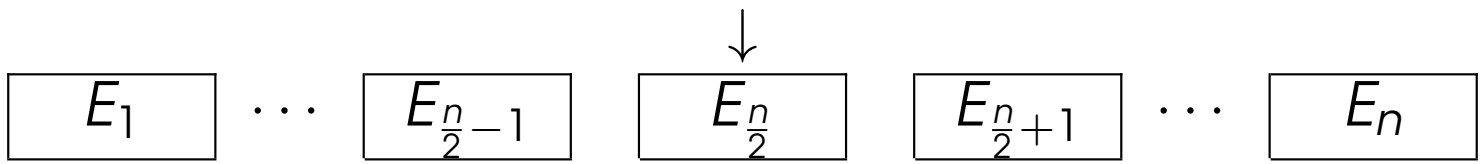
- Particionamento Hoare

```
int hoare(int V(), int inicio, int fim) {
        int pivo = V(inicio);
        int i = inicio;
        int j = fim;
        while(i < j) {
                while(j > i && V(j) >= pivo) j–;
                while(i < j && V(i) < pivo) i++;
                if(i < j) trocar(&V(i), &V(j));
        }
        return j;
}
```



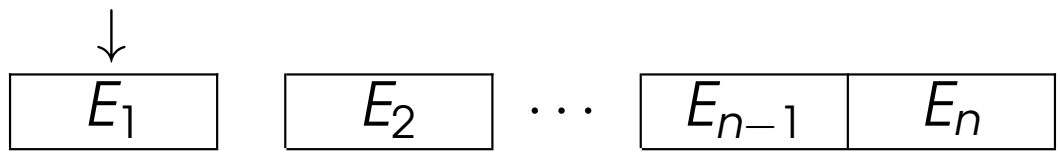
Quicksort

- Análise de complexidade
  - Melhor caso $\Omega(n \log n)$
  - Particionamento no meio do vetor
  - Dividindo em subvetores de tamanho próximos



Quicksort

- Análise de complexidade
  - Pior caso $O(n^2)$
  - Particionamento na borda do vetor
  - Subvetor com tamanho próximo a vetor inteiro



Quicksort

- Análise de complexidade
  - Ordenação in-place
  - Espaço entre $\Omega(\log n)$ e $O(n)$
  - Não estável

Exemplo

- Considerando o algoritmo Quicksort, realize a ordenação decrescente do vetor
  - Sequência 32, 54, 92, 74, 23, 3, 43, 63
  - Aplique o particionamento de Hoare
  - Execute o algoritmo passo a passo, indicando os índices e as trocas

Exercício

- A empresa de desenvolvimento de sistemas Poxim Tech está realizando um experimento para determinar qual variante do algoritmo de ordenação crescente do Quicksort apresenta o melhor resultado para um determinado conjunto de sequências numéricas
  - Neste experimento foram utilizadas as seguintes variantes: particionar padrão (PP), particionar por mediana de 3 (PM), particionar por pivô aleatório (PA), hoare padrão (HP), hoare por mediana de 3 (HM) e hoare por pivô aleatório (HA).
  - Técnicas de escolha do pivô
    - Mediana de 3: $V_1 = V\left[\frac{n}{4}\right], V_2 = V\left[\frac{n}{2}\right], V_3 = \left[\frac{3n}{4}\right]$
    - Aleatório: $V_a = V\left[ini + |V\left[ini\right]| \bmod n\right]$

Exercício

- Formato de arquivo de entrada
  - [#*n total de vetores*]
  - [#*N*1 *números do vetor* 1]
  - $[E_1] \cdots [E_{N1}]$
  - ...
  - [#*Nn números do vetor n*]
  - $[E_1] \cdots [E_{Nn}]$

```
4
6
-23 10 7 -34 432 3
4
955 -32 1 9
7
834 27 39 19 3 -1 -33
10
847 38 -183 -13 94 -2 -42 54 28 100
```

Exercício

- Formato de arquivo de saída
  - Para cada vetor é impressa a quantidade total de números N e a sequência com ordenação estável contendo o número de trocas e de chamadas

```
0: N(6) PP(15) HP(16) PM(19) HM(19) HA(20) PA(22)
1: N(4) PP(10) HP(10) PM(11) PA(11) HM(12) HA(12)
2: N(7) HP(17) PM(18) PP(23) HM(26) HA(27) PA(30)
3: N(10) PM(28) HP(28) PP(33) HA(35) HM(37) PA(38)
```