Algoritmos e Estruturas de Dados 2

Lab 06

Nome: Vitor de Meira Gomes

Matrícula: 800643

Considerações:

Para a realização desse trabalho criei uma classe Custom array que possui metodos de cria o array em ordem crescente, possui metodo para shuffle, parcial shuffle e printArray.

Também para a realização de questões como; contar o número de comparações de arrays completamente bagunçados, farei a contagem em 1000 vezes com arrays aleatórios e calcularei a média para um resultado mais preciso.

Array parcialmente ordenado com 10 elementos:

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

Array parcialmente ordenado com 100 elementos:

|2|1|4|3|6|5|8|7|10|9|12|11|14|13|16|15|18|17|20|19|22|21|24|23|26|25|28|27|30|29|32|31|34|33|36|35|38|37|40|39|42|41|44|43|46|45|48|47|50|49|52|51|54|53|56|55|58|57|60|59|62|61|64|63|66|65|68|67|70|69|72|71|74|73|76|75|78|77|80|79|82|81|84|83|86|85|88|87|90|89|92|91|94|93|96|95|98|97|100|99|

```java
class CustomArray{

    private int tam;
    public int array[];

    public int getTam()
    {
        return this.tam;
    }

    public CustomArray(int tam)
    {
        this.tam = tam;
        this.array = new int[tam];
        fillArray();
    }

    public void fillArray()
    {
        for(int i = 0; i < tam; i++)
        {
            this.array[i] = i + 1;
        }
    }
```

```java
    public void customShuffle()
    {
        Random random = new Random();

        for(int i = tam - 1; i > 0; i--)
        {
            int j = random.nextInt(i + 1);

            int temp = array[i];
            array[i] = array[j];
            array[j] = temp;
        }
    }

    public void partialShuffle()
    {
        for(int i = 0; i < tam; i+=2)
        {
            int temp = array[i];
            array[i] = array[i+1];
            array[i+1] = temp;
        }
    }
```

```java
    public void printArray()
    {
        System.out.print("|");
        for(int i = 0; i < tam; i++)
        {
            System.out.print(array[i] + "|");
        }
        System.out.println();
    }
```

Desordenado

Tamanho do Array	Pivô	Comparações	Movimentações	Tempo (ms)
100	Primeiro	994	563	0.094
	Último	993	564	0.089
	Random Pivot	918	556	0.16
	Mediana de 3	825	568	0.078
1.000	Primeiro	15140	7918	0.809
	Último	15144	7919	0.827
	Random Pivot	13789	7874	1.234
	Mediana de 3	12354	8026	0.823
10.000	Primeiro	203967	102181	9.694
	Último	204080	102162	9.821
	Random Pivot	184009	101775	13.854
	Mediana de 3	164665	103750	9.594

Ordenado

Tamanho do Array	Pivô	Comparações	Movimentações	Tempo (ms)
100	Primeiro	5148	297	0.0
	Último	5148	297	0.0
	Random Pivot	729	201	0.0
	Mediana de 3	606	189	0.0
1.000	Primeiro	501498	2997	0.0
	Último	501498	2997	0.0
	Random Pivot	12454	2001	0.0
	Mediana de 3	9009	1533	0.0
10.000	Primeiro	50014998	29997	54.0
	Último	50014998	29997	58.0
	Random Pivot	156817	19959	0.0
	Mediana de 3	125439	17712	0.0

Funcionamento de Cada Estratégia de Escolha do Pivô

Primeiro Elemento: A estratégia utiliza o primeiro elemento do array como pivô. Quando o array está ordenado ou quase ordenado, essa escolha pode resultar em divisões muito desbalanceadas, o que prejudica o desempenho, pois o algoritmo pode se aproximar da complexidade $O(n^2)$.

Último Elemento: Similar à estratégia do primeiro elemento, mas utiliza o último elemento como pivô. Ela também é vulnerável ao problema de divisão desbalanceada em arrays ordenados ou parcialmente ordenados, resultando em muitas comparações desnecessárias.

Random Pivot: Um elemento aleatório é escolhido como pivô. Essa estratégia reduz a chance de divisões desbalanceadas, oferecendo uma performance mais estável na maioria dos casos, evitando cenários de pior caso frequentes.

Mediana de 3: A mediana entre o primeiro, o meio e o último elemento é utilizada como pivô. Esta estratégia geralmente evita divisões muito desbalanceadas, sendo considerada uma das mais eficientes para QuickSort, especialmente quando o array já está ordenado ou parcialmente ordenado.

Desempenho Observado em Cada Cenário

Arrays Desordenados

Os tempos de execução para os arrays desordenados mostraram que as estratégias "Primeiro Elemento" e "Último Elemento" apresentam um desempenho similar, com tempos crescentes conforme o tamanho do array aumenta. A estratégia "Mediana de 3" foi a mais eficiente para arrays menores, enquanto a "Random Pivot" apresentou um bom desempenho para tamanhos intermediários.

Arrays Ordenados

Nos arrays já ordenados, as estratégias "Primeiro Elemento" e "Último Elemento" tiveram o pior desempenho, devido ao problema de divisões desbalanceadas. A estratégia "Mediana de 3" se destacou como a mais eficiente, com tempos de execução significativamente menores. "Random Pivot" também teve um bom desempenho, mas ligeiramente inferior à mediana.

Arrays Parcialmente Ordenados

Nos arrays parcialmente ordenados, o comportamento das estratégias foi semelhante ao cenário dos arrays ordenados. "Mediana de 3" e "Random Pivot" apresentaram os melhores resultados. A estratégia de escolher o primeiro ou último elemento levou a muitos casos de divisão desbalanceada, resultando em tempos de execução maiores.

Qual Estratégia Foi Mais Eficiente?

A análise dos dados demonstra que a estratégia mais eficiente depende do estado do array e do tamanho da entrada:

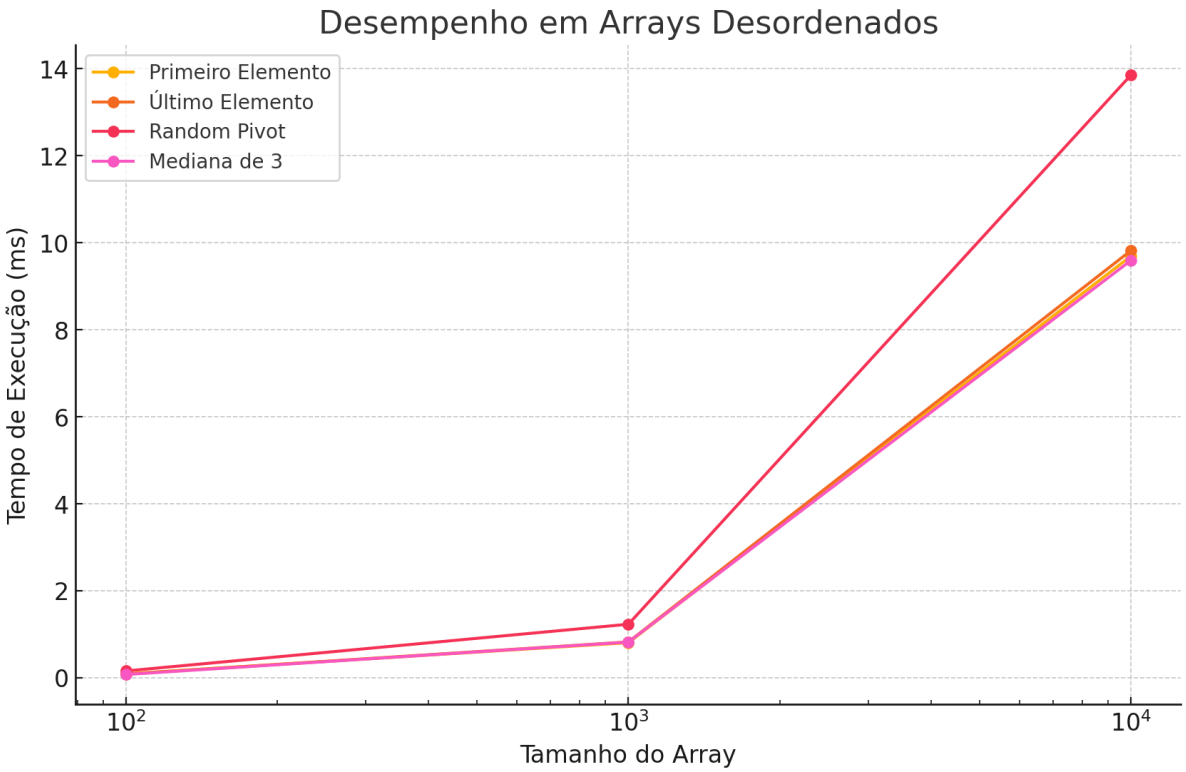
Desordenado: A estratégia "Mediana de 3" foi a mais eficiente para arrays menores, enquanto a "Random Pivot" apresentou um desempenho competitivo para arrays maiores. Isso se deve ao fato de que essas estratégias minimizam a chance de divisões desbalanceadas.

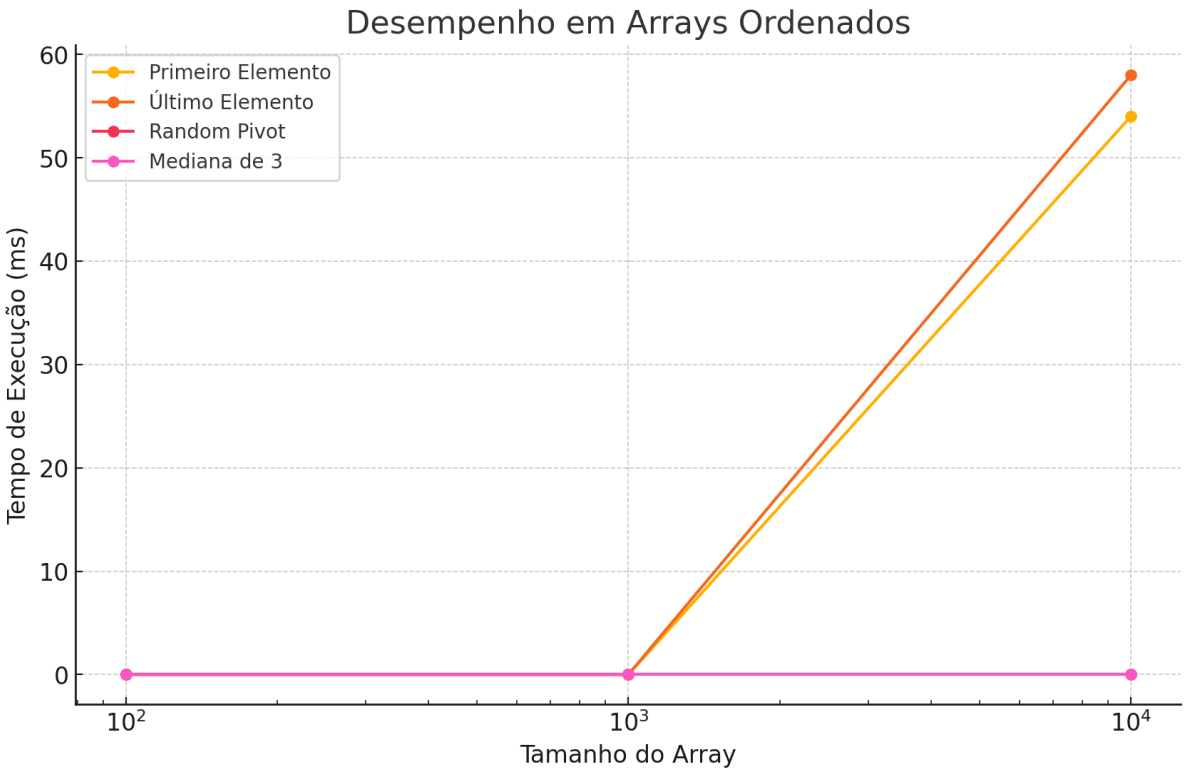
Ordenado: A "Mediana de 3" foi a mais eficiente, seguida pela "Random Pivot". As estratégias "Primeiro Elemento" e "Último Elemento" se mostraram muito ineficazes, já que sempre produzem divisões muito desbalanceadas neste cenário.

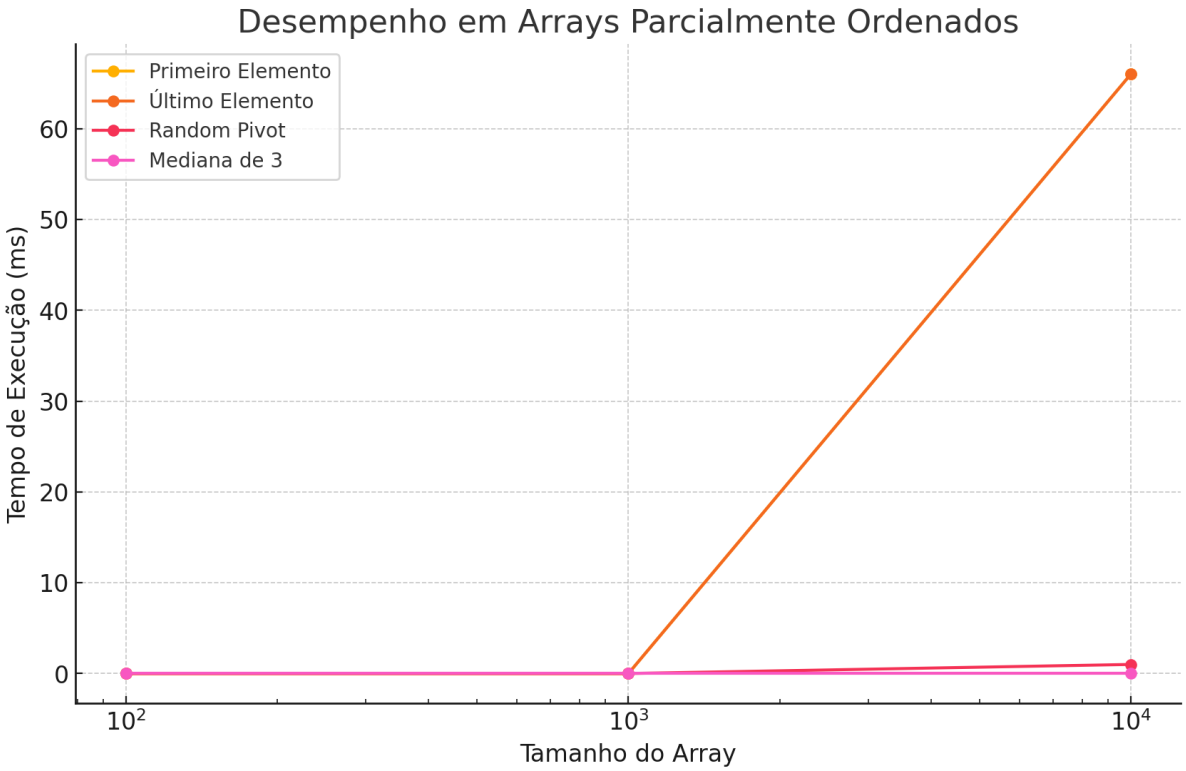
Parcialmente Ordenado: Assim como no cenário ordenado, "Mediana de 3" foi a mais eficiente, seguida de perto por "Random Pivot". As estratégias "Primeiro Elemento" e "Último Elemento" também foram ineficazes, pelos mesmos motivos.

Conclusão

A escolha do pivô influencia significativamente o desempenho do QuickSort. Para arrays ordenados ou parcialmente ordenados, a "Mediana de 3" é a escolha mais eficiente, enquanto "Random Pivot" também oferece um desempenho robusto em diversos cenários. As estratégias que escolhem o primeiro ou o último elemento como pivô devem ser evitadas em casos onde o array pode estar ordenado, para evitar o pior caso do algoritmo.


Desempenho em Arrays Desordenados
Primeiro Elemento
Último Elemento
Random Pivot
Mediana de 3
Tempo de Execução (ms)
0
2
4
6
8
10
12
14
10^2
10^3
10^4
Tamanho do Array


Desempenho em Arrays Ordenados
Primeiro Elemento
Último Elemento
Random Pivot
Mediana de 3
Tempo de Execução (ms)
0
10
20
30
40
50
60
10^2
10^3
10^4
Tamanho do Array


Desempenho em Arrays Parcialmente Ordenados
Primeiro Elemento
Último Elemento
Random Pivot
Mediana de 3
Tempo de Execução (ms)
0
10
20
30
40
50
60
10^2
10^3
10^4
Tamanho do Array