O funcionamento de cada estratégia de escolha do pivô:

- Primeiro Elemento:

O algoritmo utiliza o primeiro elemento do array como pivô. Esta é uma estratégia simples, mas nem sempre a mais eficiente. Dependendo da natureza dos dados, pode levar a uma pior performance.

```java
public static void QuickSortFirstPivot(int esq, int dir, int[] array){
    int i = esq, j = dir;
    int pivo = array[esq];

    while (i <= j){
        while(array[i] < pivo){
            i++;
        }
        while(array[j] > pivo){
            j--;
        }

        if(i <= j){
            swap(i, j, array);
            i++;
            j--;
        }
    }

    if(esq < j){
        QuickSortFirstPivot(esq, j, array);
    }
    if(i < dir){
        QuickSortFirstPivot(i, dir, array);
    }
}
```

- Último Elemento:

o pivô sendo o último elemento do array é uma escolha válida, mas não necessariamente a mais eficiente para todos os tipos de dados. Ela pode levar a problemas de performance no caso de arrays ordenados.

```java
public static void QuickSortLastPivot(int esq, int dir, int[] array){
    int i = esq, j = dir;
    int pivo = array[dir];

    while (i <= j){
        while(array[i] < pivo){
            i++;
        }
        while(array[j] > pivo){
            j--;
        }

        if(i <= j){
            swap(i, j, array);
            i++;
            j--;
        }
    }

    if(esq < j){
        QuickSortLastPivot(esq, j, array);
    }
    if(i < dir){
        QuickSortLastPivot(i, dir, array);
    }
}
```

- Pivô Aleatório:

Escolher um pivô aleatoriamente é uma estratégia poderosa que minimiza o risco de pior caso e torna o algoritmo mais robusto, especialmente contra inputs adversários.

```java
public static void QuickSortRandomPivot(int esq, int dir, int[] array){
    int i = esq, j = dir;
    int pivo = array[(dir + esq) / 2];

    while (i <= j){
        while(array[i] < pivo){
            i++;
        }
        while(array[j] > pivo){
            j--;
        }

        if(i <= j){
            swap(i, j, array);
            i++;
            j--;
        }
    }

    if(esq < j){
        QuickSortRandomPivot(esq, j, array);
    }
    if(i < dir){
        QuickSortRandomPivot(i, dir, array);
    }
}
```

- Mediana de Três Elementos:

A estratégia de Mediana de Três é uma forma eficaz de escolher um pivô que tende a gerar partições mais equilibradas, melhorando a eficiência do algoritmo.

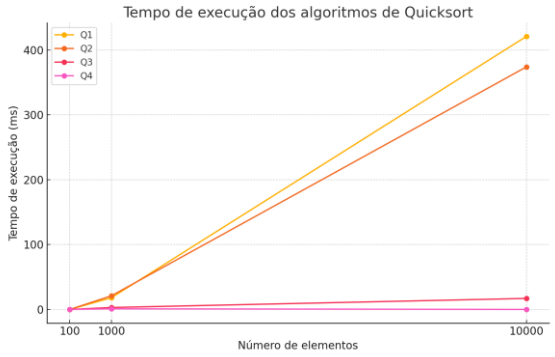
```java
public static void QuickSortMedianOfThree(int esq, int dir, int[] array){
    int i = esq, j = dir;
    int tmp = (esq + dir) / 2;

    if(array[esq] > array[tmp]){
        swap(esq, tmp, array);
    }
    if(array[esq] > array[dir]){
        swap(esq, dir, array);
    }
    if(array[tmp] > array[dir]){
        swap(tmp, dir, array);
    }

    int pivo = array[tmp];
    while (i <= j){
        while(array[i] < pivo){
            i++;
        }
        while(array[j] > pivo){
            j--;
        }
```

```java

            if(i <= j){
                swap(i, j, array);
                i++;
                j--;
            }
        }

        if(esq < j){
            QuickSortMedianOfThree(esq, j, array);
        }
        if(i < dir){
            QuickSortMedianOfThree(i, dir, array);
        }
    }
```

Desempenho representado com gráficos e tabelas:

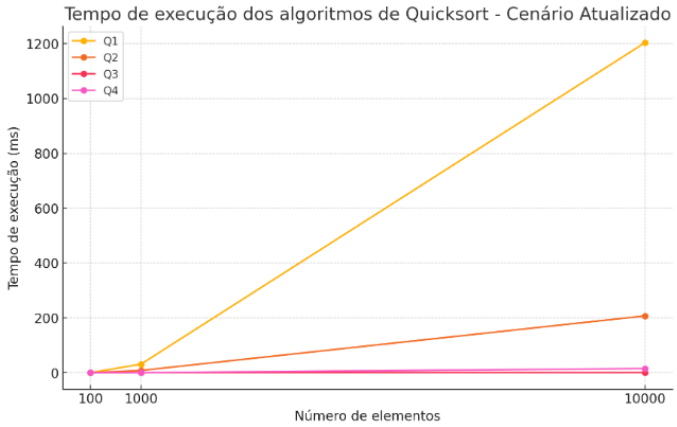
- Cenário 1 – Array Ordenado Crescente:

-	100	1000	10.000
Q1	0.0	18.0	421.0
Q2	0.0	21.0	374.0
Q3	0.0	3.0	17.0
Q4	0.0	1.0	0.0



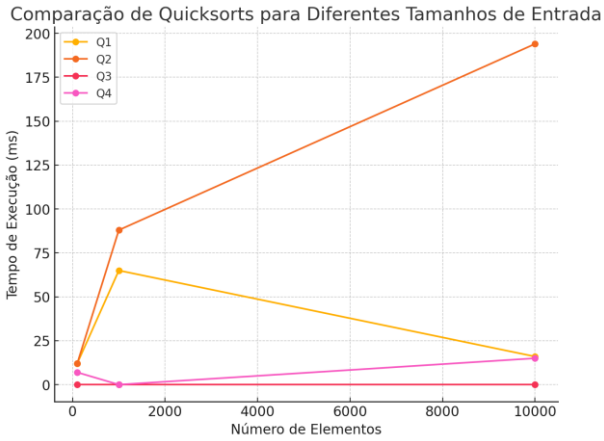
- Cenário 2 – Array Ordenado Decrescente:

-	100	1000	10.000
Q1	0.0	31.0	1203.0
Q2	0.0	8.0	207.0
Q3	0.0	0.0	0.0
Q4	0.0	0.0	15.0



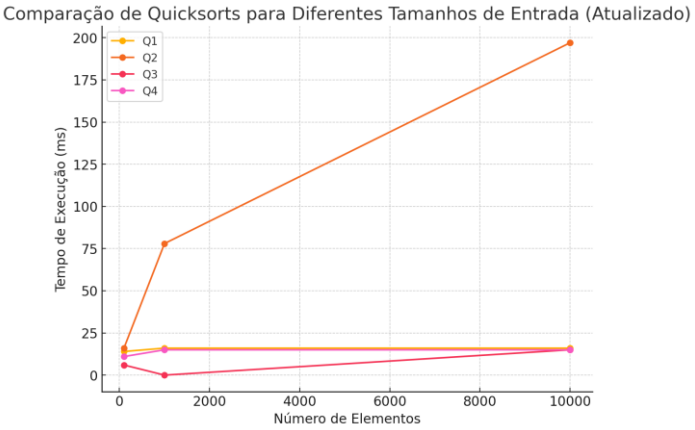
- Cenário 3 – Array Parcial Ordenado Crescente:

-	100	1000	10.000
Q1	12.0	65.0	16.0
Q2	12.0	88.0	194.0
Q3	0.0	0.0	0.0
Q4	7.0	0.0	15.0



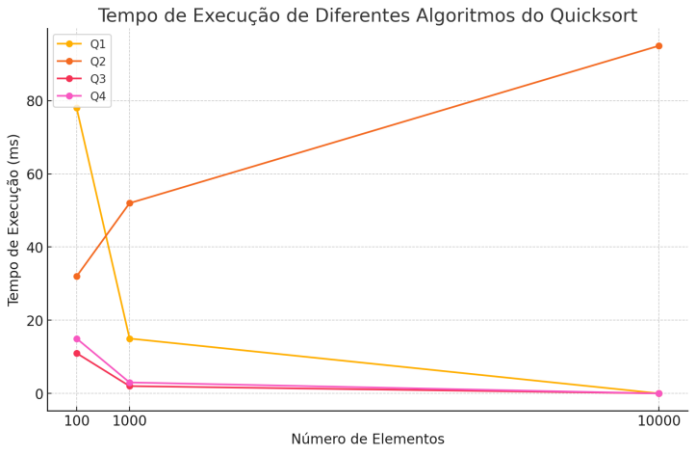
- Cenário 4 – Array Parcial Ordenado Decrescente:

-	100	1000	10.000
Q1	14.0	16.0	16.0
Q2	16.0	78.0	197.0
Q3	6.0	0.0	15.0
Q4	11.0	15.0	15.0



- Cenário 5 – Array Aleatório:

-	100	1000	10.000
Q1	78.0	15.0	0.0
Q2	32.0	52.0	95.0
Q3	11.0	2.0	0.0
Q4	15.0	3.0	0.0



Discussão sobre qual estratégia é mais eficiente e porquê:

A mediana de três é a mais eficiente na maioria dos casos porque garante que o pivô seja representativo dos dados. Isso leva a partições mais balanceadas e reduz a chance de o algoritmo precisar fazer muitas chamadas recursivas.