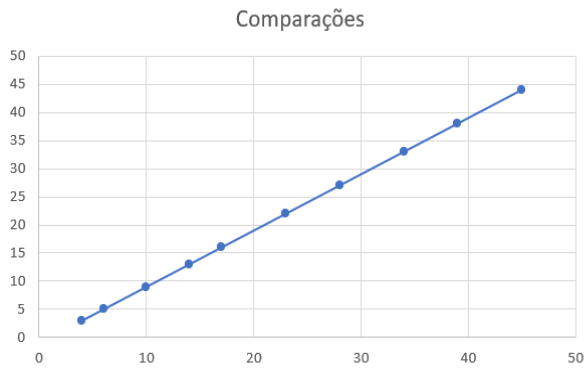
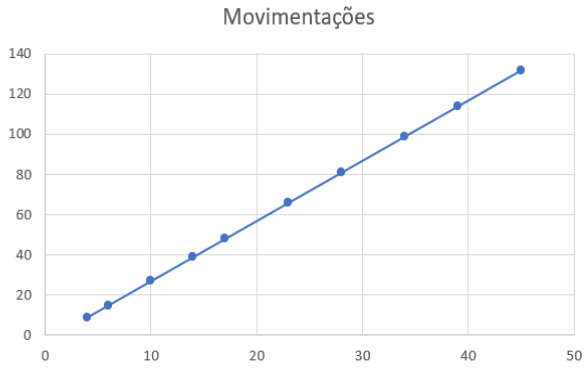
OBS: os valores usados para o tamanho do vetor foram: 4, 6, 10, 14, 17, 23, 28, 34, 39 e 45.

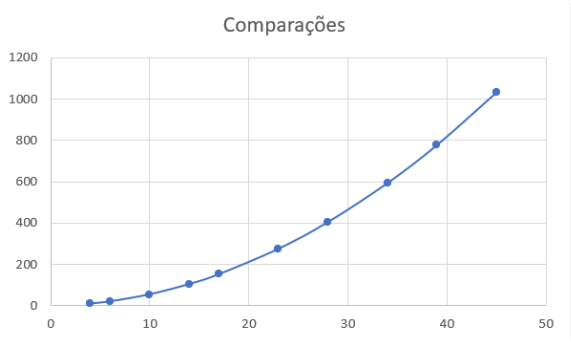
Inserção direta

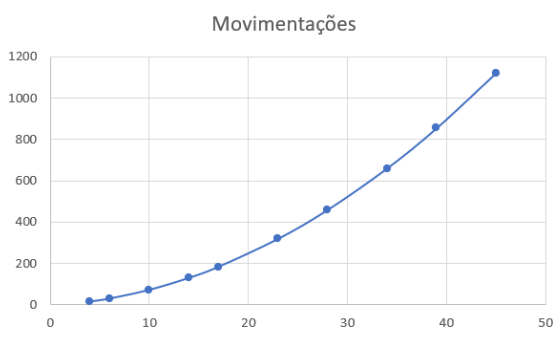
No melhor caso que ocorre quando o vetor está ordenado em *ordem crescente*, o algoritmo é O(N) em comparação e movimentação:



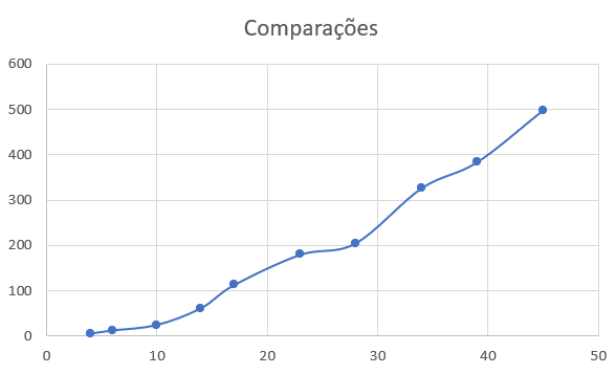


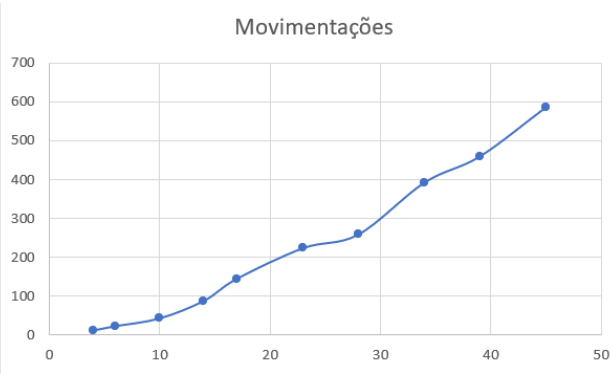
No pior caso, que ocorre quando o vetor está ordenado em *ordem decrescente*, o algoritmo é O(N^2) em movimentação e comparação:





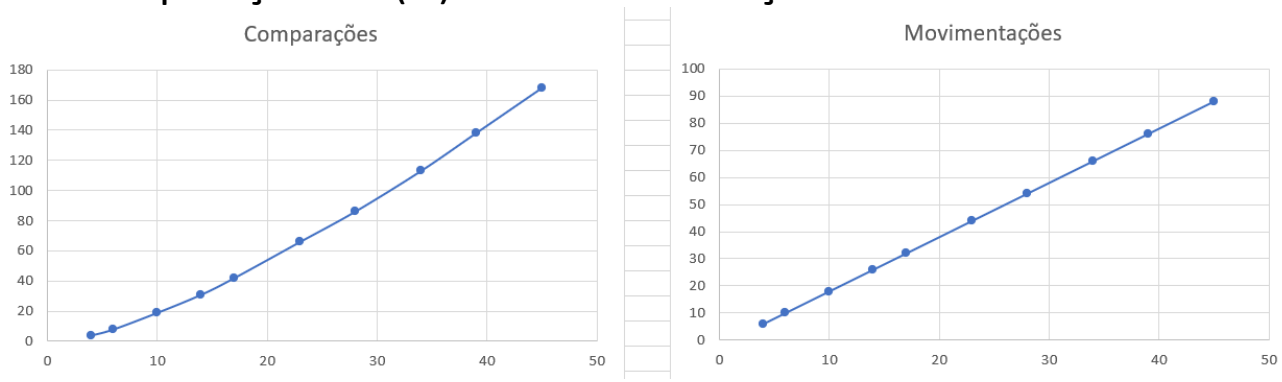
Para vetores de *valores aleatórios*, o algoritmo tende a ser O(N^2) em movimentação e comparação:



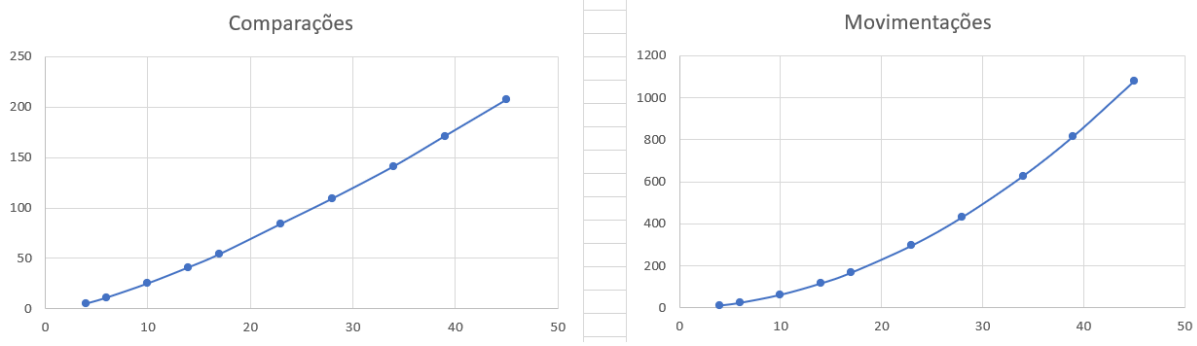


Inserção binária

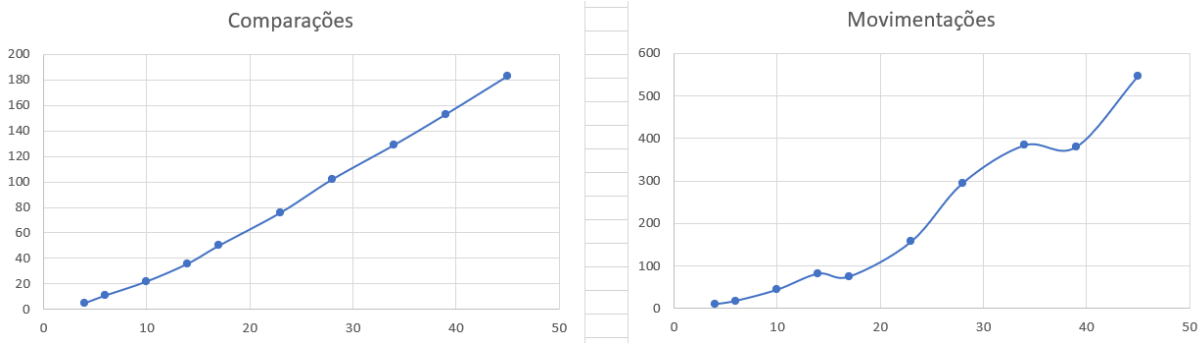
No melhor caso, que ocorre quando o vetor está ordenado em *ordem crescente*, o algoritmo é O(N * logN) em comparação e O(N) em movimentação:



No pior caso, que ocorre quando o vetor está ordenado em *ordem decrescente*, o algoritmo é O(N * logN) em comparação e O(N^2) em movimentação:



Para vetores de *valores aleatórios*, o algoritmo tende a ser O(N * logN) em comparação e O(N^2) em movimentação:



Seleção

Tomando como base o código do algoritmo:

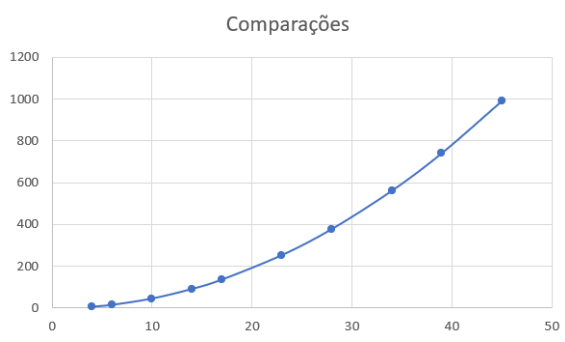
```
for(int i = 1; i <= (lenght - 1); i++)
{
    indice_menor = i;
    for(int j = i + 1; j <= lenght; j++)
    {
        selectionComparacoes++;
        if(array[j] < array[indice_menor])
            indice_menor = j;
    }
    aux = array[i];
    array[i] = array[indice_menor];
    array[indice_menor] = aux;
    selectionMovimentacoes += 3;
}
```

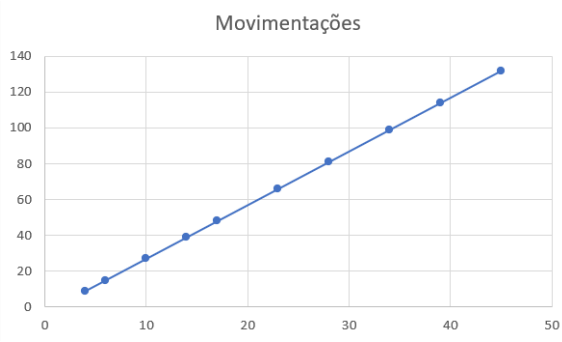
O algoritmo será O(N^2) para todos os casos de comparação, pois `if(array[j] < array[indice_menor])` é testado o máximo de vezes dentro do for aninhado.

Em questão de movimentação, o algoritmo será O(N) para todos os casos, pois

```
aux = array[i];
array[i] = array[indice_menor];
array[indice_menor] = aux;
selectionMovimentacoes += 3;
```

é executado ao máximo no for externo.



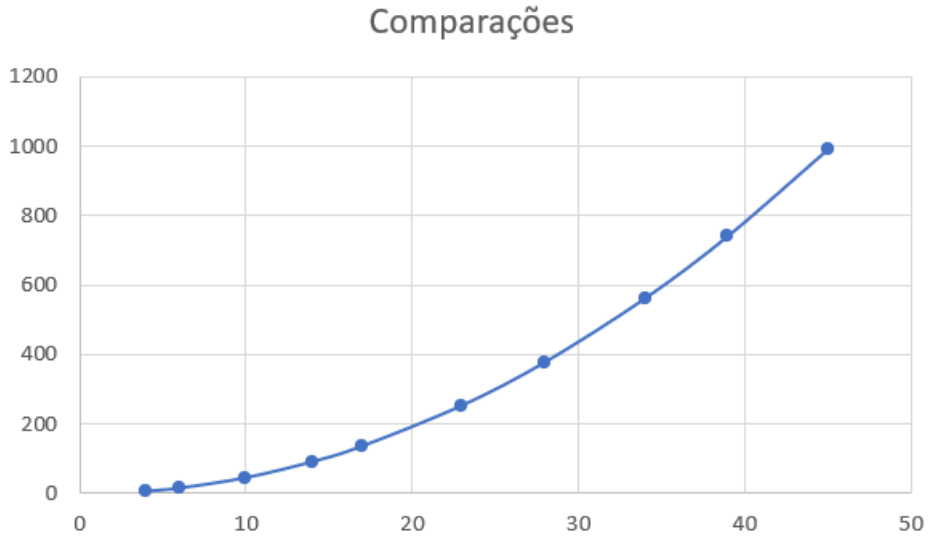


Bubblesort

Tomando como base parte do código do algoritmo:

```
for(int i = 2; i <= lenght; i++)
{
    for(int j = lenght; j >= i; j--)
    {
        bubbleComparacoes++;
        if(array[j - 1] > array[j])
        {
            aux = array[j - 1];
            array[j - 1] = array[j];
            array[j] = aux;
            bubbleMovimentacoes += 3;
        }
    }
}
```

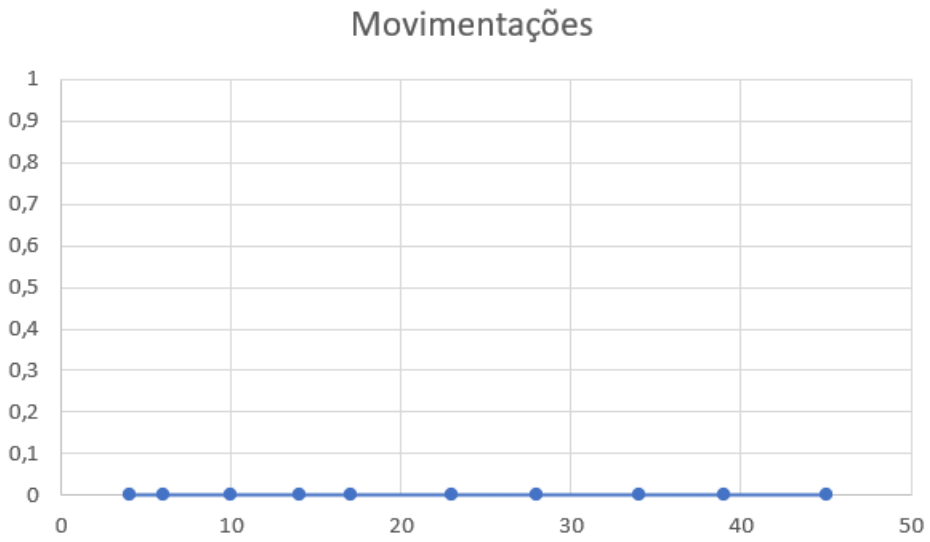
O algoritmo será O(N^2) para todos os casos de comparação, pois `if(array[j - 1] > array[j])` é testado o máximo de vezes dentro do for aninhado:



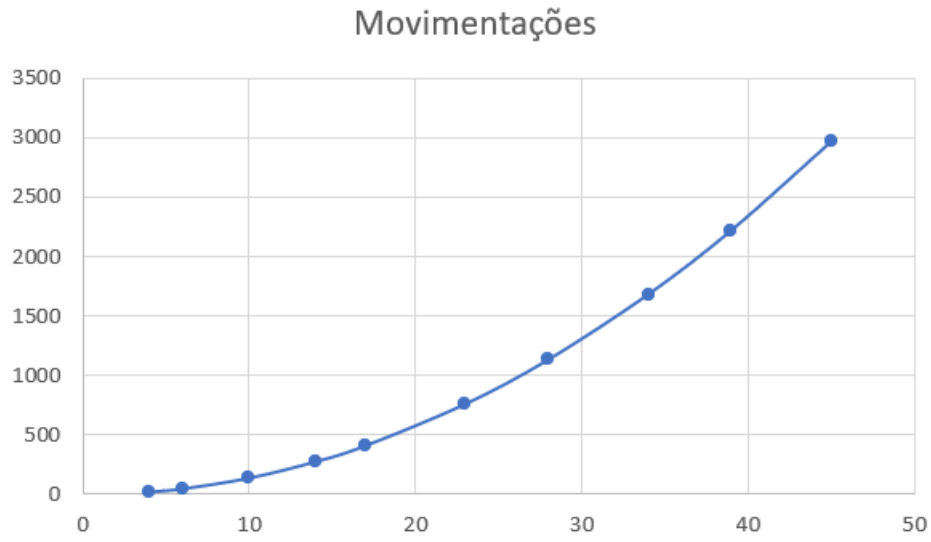
Em questão de movimentação, o algoritmo será O(1) no melhor caso, que ocorre quando o vetor está ordenado em *ordem crescente*, pois

```
aux = array[j - 1];
array[j - 1] = array[j];
array[j] = aux;
bubbleMovimentacoes += 3;
```

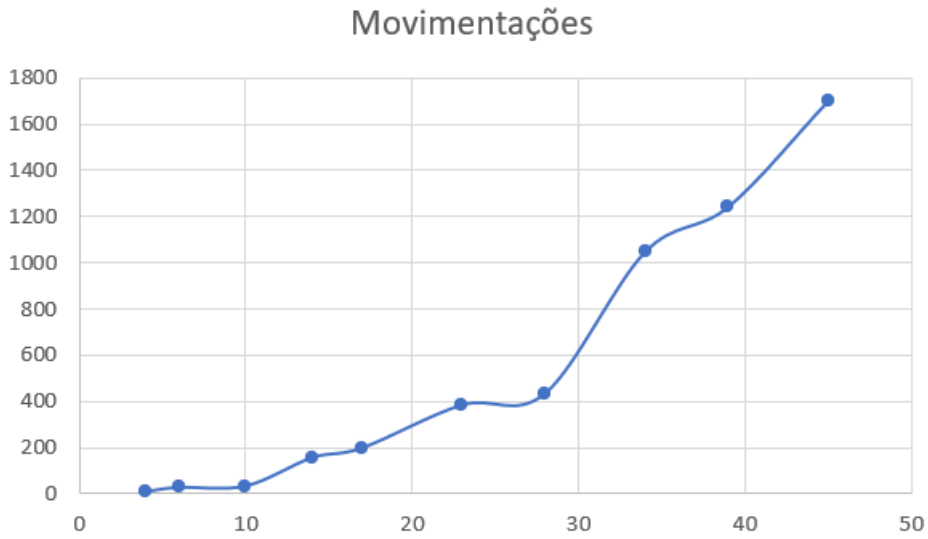
não é executado:



No pior caso, que ocorre quando o vetor está ordenado em *ordem decrescente*, o algoritmo será O(N^2) em movimentação:

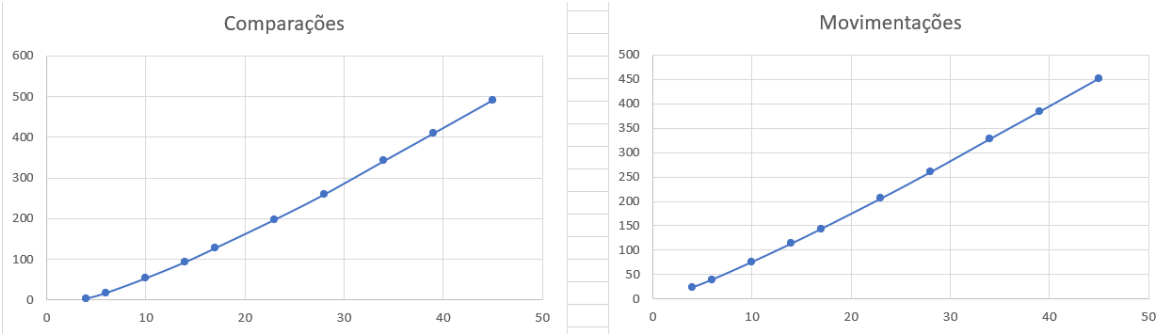


Para vetores de *valores aleatórios*, o algoritmo tende a ser O(N^2) em movimentação:



Heapsort

O algoritmo é O(N * logN) para todos os casos de movimentação e de comparação:

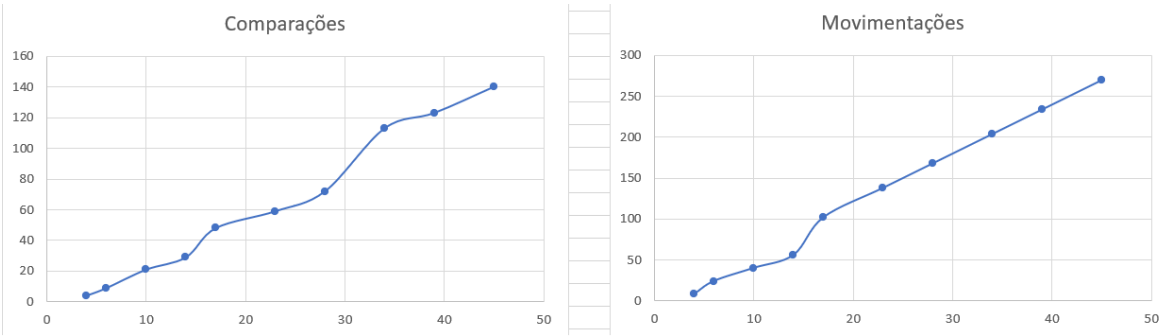


Fusão

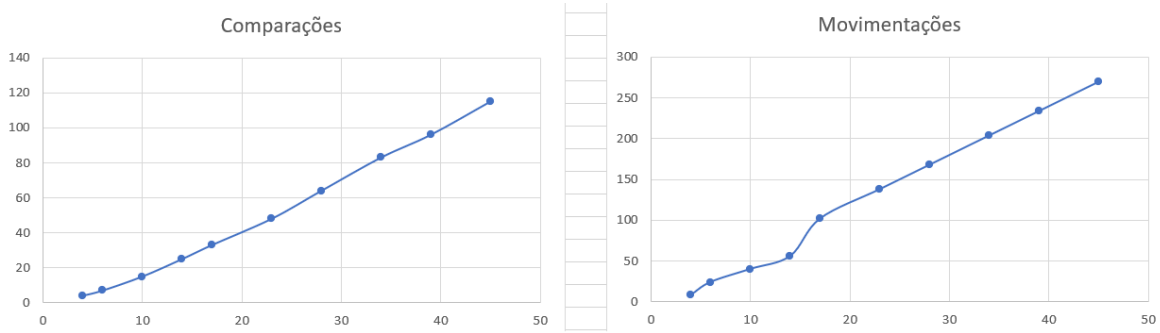
- Para vetores em que os valores não são todos potência de 2:

Pelos gráficos, todos os casos de comparação e de movimentação tendem a O(N * logN):

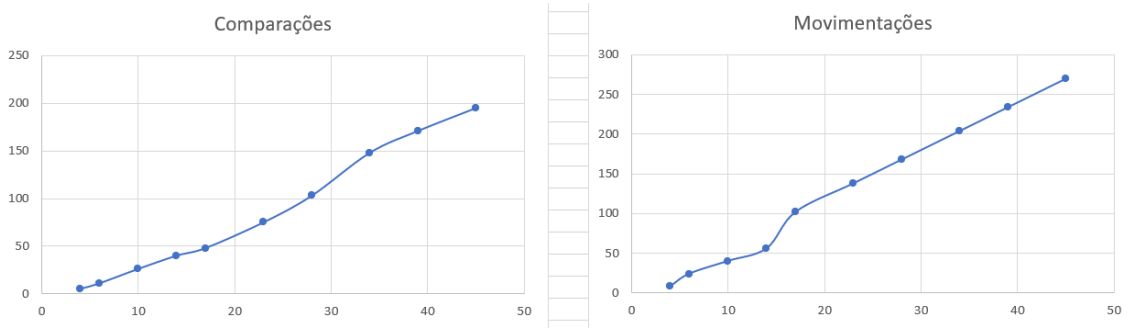
Para valores *crescentes*:



Para valores *decrescentes*:

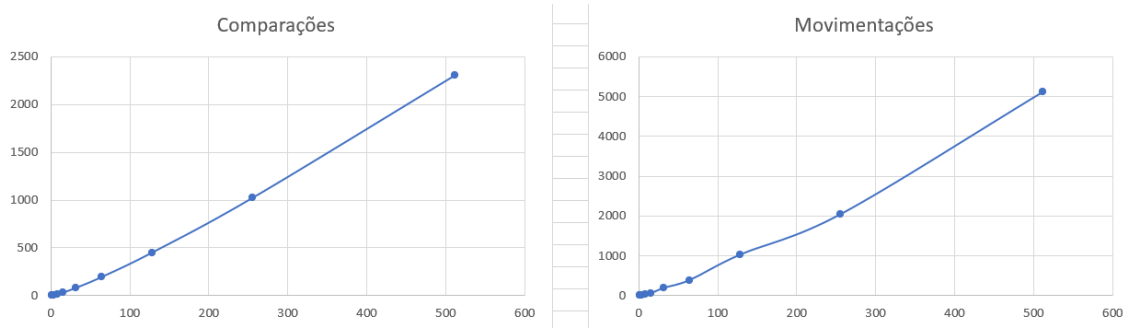


Para valores *aleatórios*:

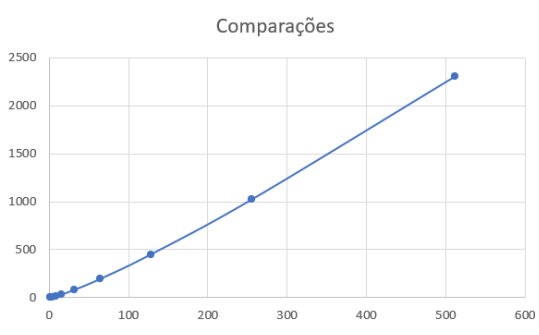


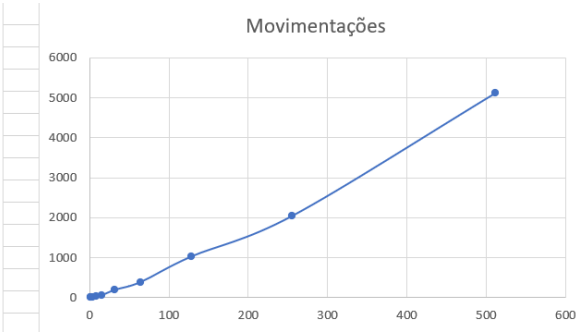
- Supondo que os valores do vetor são todos potência de 2 (1, 2, 4, 8, 16, 32, 64, 128, 256, 512), o vetor será particionado potências de 2 cada vez menores. Dessa forma, será, para todos os casos de comparação e de movimentação, O(N * logN):

Para valores *crescentes*:

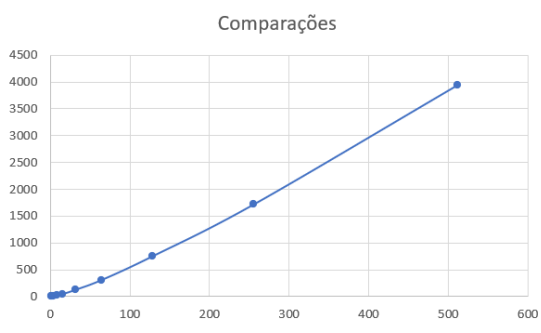


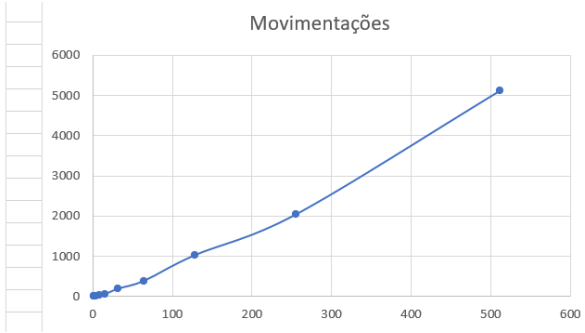
Para valores *decrescentes*:





Para valores *aleatórios*:





QuickSort

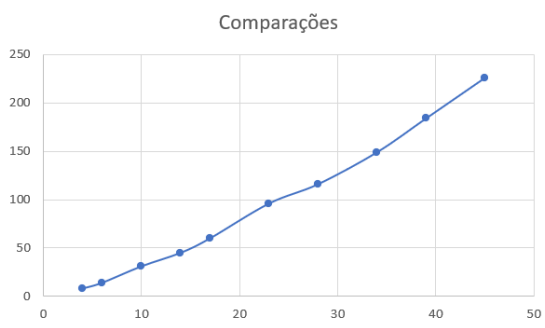
Tomando como base parte do código do algoritmo:

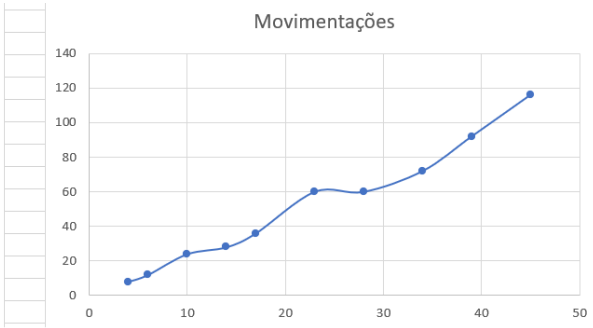
```
int meio = floor((L + R) / 2);
int i = L, j = R, pivot = array[meio];
quickMovimentacoes++;
```

Como o pivot escolhido é sempre o elemento do meio, essa implementação não resultará no pior caso, mesmo que o vetor já esteja ordenado em ordem *crescente* ou *decrescente*. Também cabe ressaltar que, nessa implementação, nem sempre o vetor será dividido em porções iguais, haja vista as diferentes entradas de tamanho, o que dificulta a ocorrência do melhor caso.

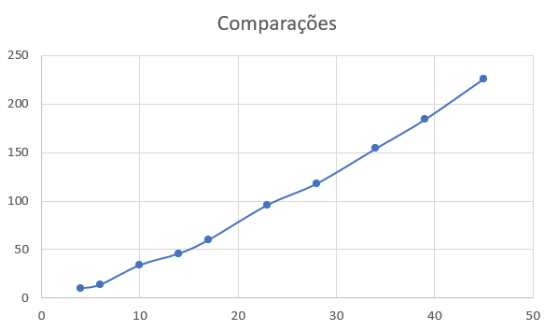
Assim, como é possível ver pelos gráficos, o algoritmo tende ao caso médio: O(N * logN) em comparação e movimentação:

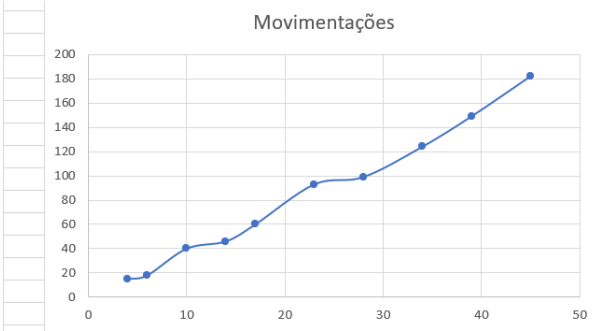
Para valores *crescentes*:





Para valores *decrescentes*:





Para valores *aleatórios*: