Análises Sobre

Os

Algoritmos De Ordenação

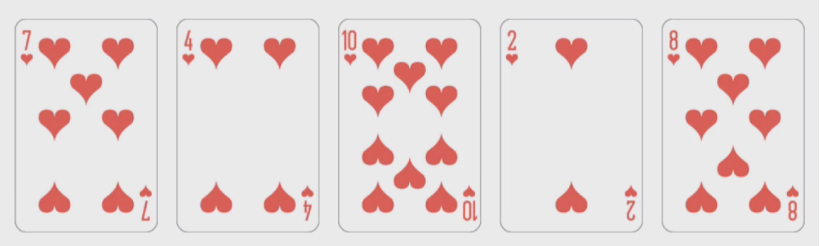
**Todos os testes realizados tiveram n (tamanho) definidos como: 1.000, 5.000, 10.000, 50.000, 100.000, 500.000 e 1.000.000, 5.000.000). Para testar com um tamanho diferente, basta ir na linha 5 do arquivo .c e alterar a macro parametrizada para atribuir o n desejado a variável MAX. Todos os algoritmos encontram-se no arquivo .c, então não incluímos eles nessa parte da análise para não ficar redundante.**

Insertion Sort  
Ideia Geral:

 Fazendo uma analogia em relação a esse algoritmo é como se tivéssemos uma certa quantidade de cartas de um baralho na mão e ordenamos ela começando de um lado até chegar no outro.

Nós assumimos que a primeira carta já está ordenada, então seleciona-se uma outra carta e se essa outra carta é maior, coloca-se à direita, caso contrário é inserida a esquerda.

Considerando a sequência de cartas abaixo, ordenando 1 de cada vez começando pelo lado esquerdo teríamos o seguinte:



1o) (7 > 4) 🡪 4-7-10-2-8  
2o) (7 < 10) 🡪 4-7-10-2-8  
3o) (2 < 4) 🡪 2-4-7-10-8  
4o) (8 < 10 && 8 > 7) 🡪 2-4-7-8-10

Na ideia computacional segue o mesmo princípio da analogia, usando vetores. Considere o vetor: |9|5|1|4|3|.

1. O primeiro elemento do vetor (9) é considerado já ordenado. Então seleciona-se o próximo elemento e armazena-se em uma variável separada, seja essa variável chamada de ***Next***. Nesse momento ***Next*** armazena o número 5. Se ***Next*** é maior que o primeiro elemento (9) então ele é colocado a sua direita, mas nesse caso 5 < 9, logo ele é inserido à esquerda de 9. Gerando o vetor: |5|9|1|4|3|.
2. Agora os dois primeiros elementos estão ordenados, então pega o próximo elemento, atualiza ***Next*** agora armazena o número 1. Usando o ***Next*** percorre pela parte já ordenada e verifica-se que 1 é menor que toda a parte ordenada então organiza-se o vetor para inserir o ***Next*** em sua devida posição gerando o vetor: |5|9|9|4|3| e depois |5|5|9|4|3|. Por fim insere-se o elemento no início do vetor gerando: |1|5|9|4|3|.
3. Agora ***Next*** armazena o número 4 e percorre a parte já ordenada e verifica que 4 < 5. Então inicia a organização do vetor para inserir o ***Next***. Gerando nos passos as seguintes situações: |1|5|9|9|3|, |1|5|5|9|3| e por fim gera o resultado final: |1|4|5|9|3|.
4. Por fim ***Next*** armazena o número 3 e percorre novamente toda a parte já ordenada e descobre que 3 < 4. Seguindo a organização temos: |1|4|5|9|9|, |1|4|5|5|9|, |1|4|4|5|9| e por fim inserindo no devido local obtemos: |1|3|4|5|9|.

## Complexidade:

Pior caso θ(n2):

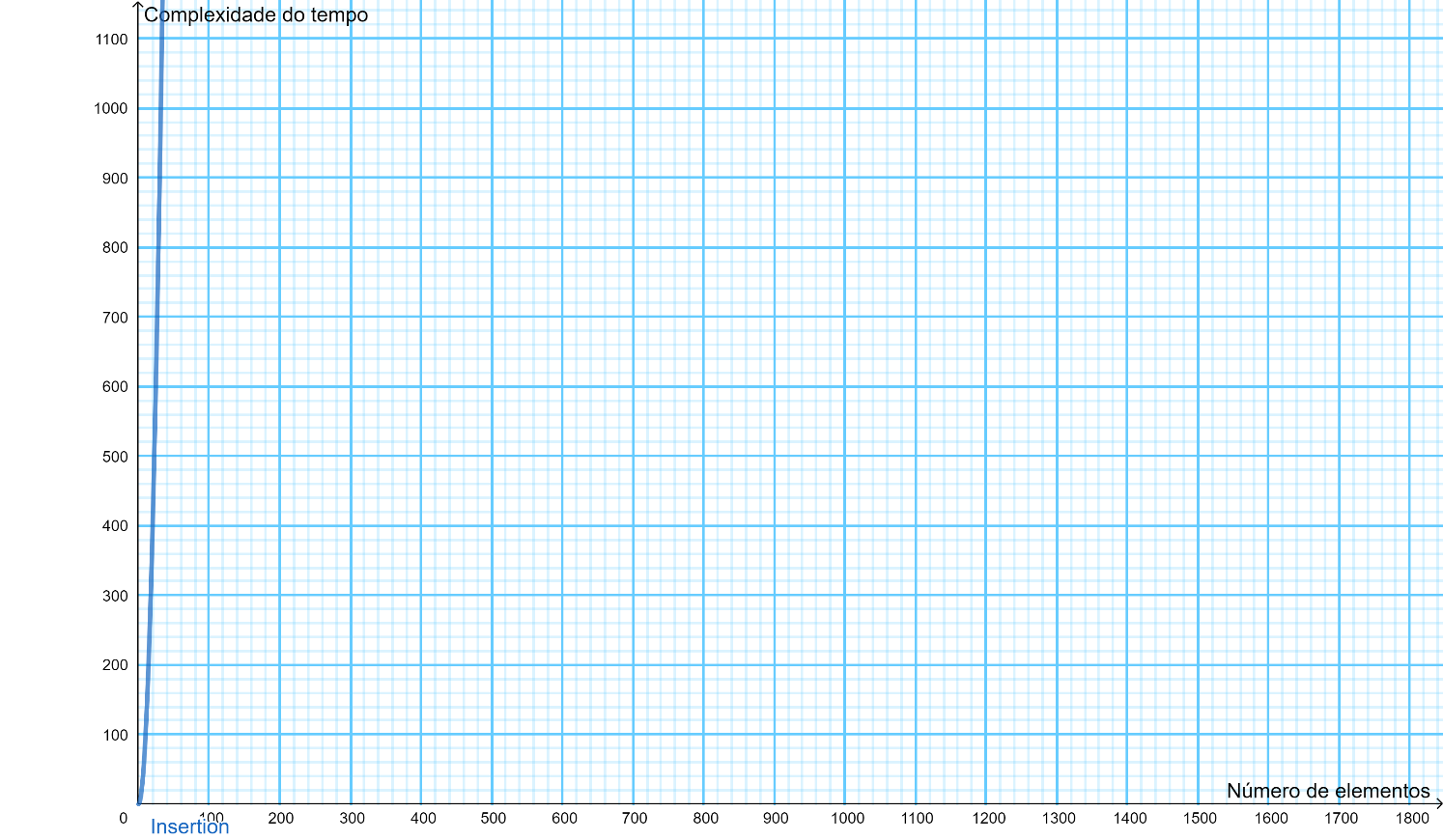
 Supondo que o vetor esteja ordenado de forma decrescente e quer ordenar em forma crescente, nesse momento é configurado o seu pior caso, pois cada elemento tem que ser comparado com os outros elementos, ou seja, para elemento na posição n do vetor, serão feitas n-1 comparações. Logo o numero de comparações é dada por n\*(n-1) ≅ n2.

Figura 1

Melhor caso θ(n):

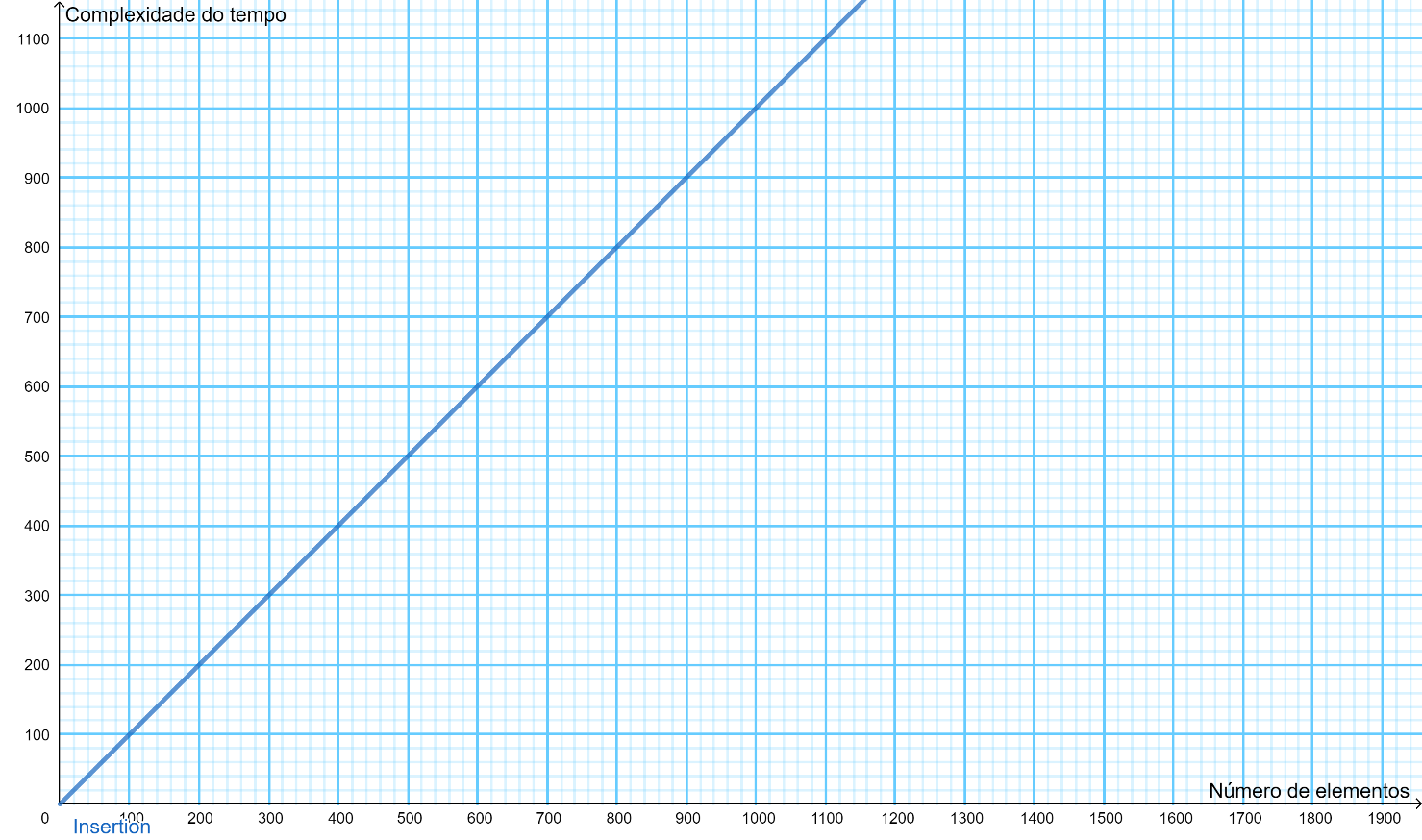
 Quando o vetor já se encontra ordenado então não é necessário fazer alterações logo somente o loop mais externo que vai até n será executado fazendo n comparações.

Figura 2

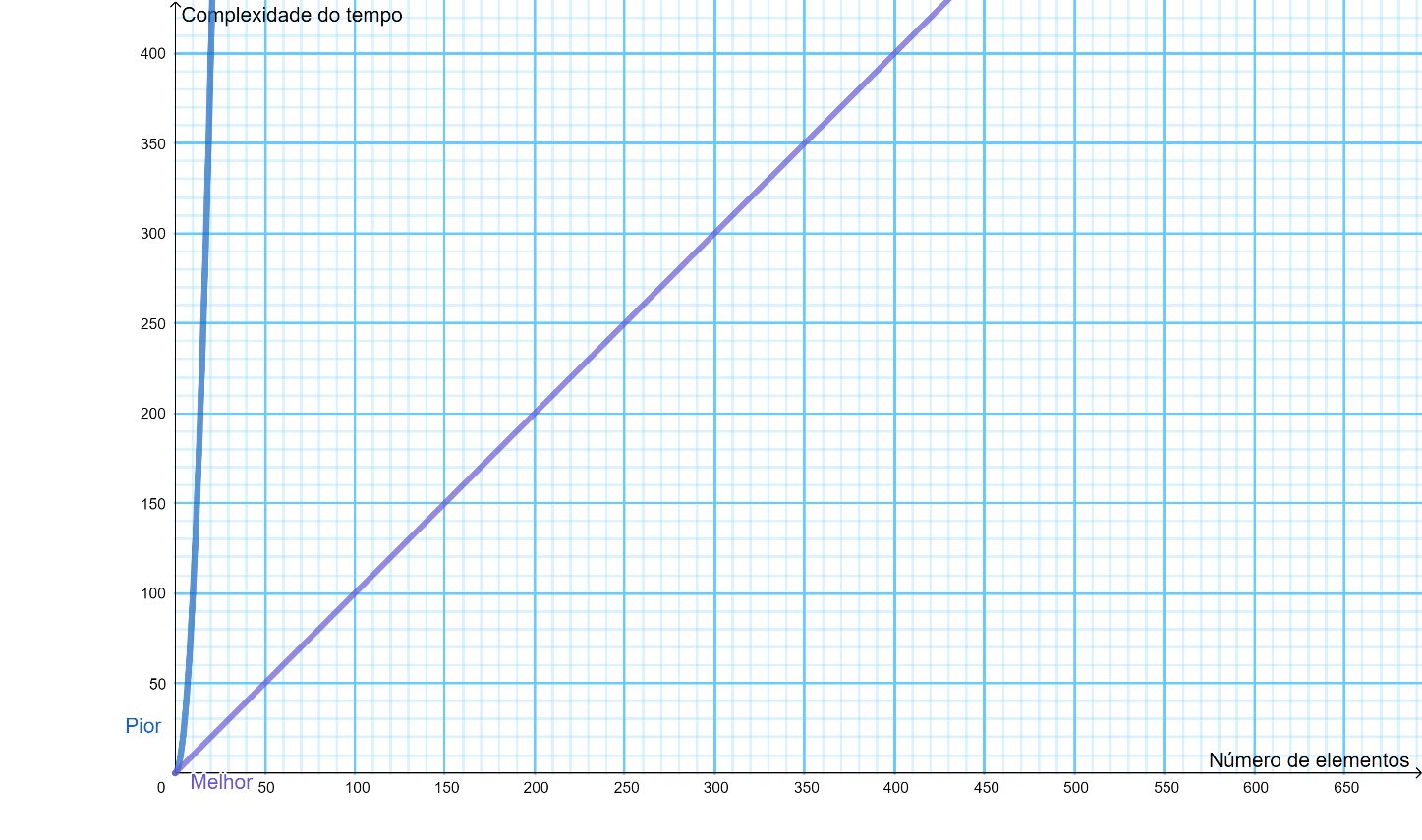
Nota-se pelo gráfico das funções logo abaixo nas figuras 3 e 4: Conforme a quantidade de elementos aumenta o tempo aumenta e no pior caso mesmo com uma quantidade baixa de elemento no vetor o tempo é extremamente maior observa-se melhor a discrepância na Figura 3 e observa-se também na Figura 4 o momento em que ambos possuem o mesmo tempo de execução no ponto A. 

Figura 3

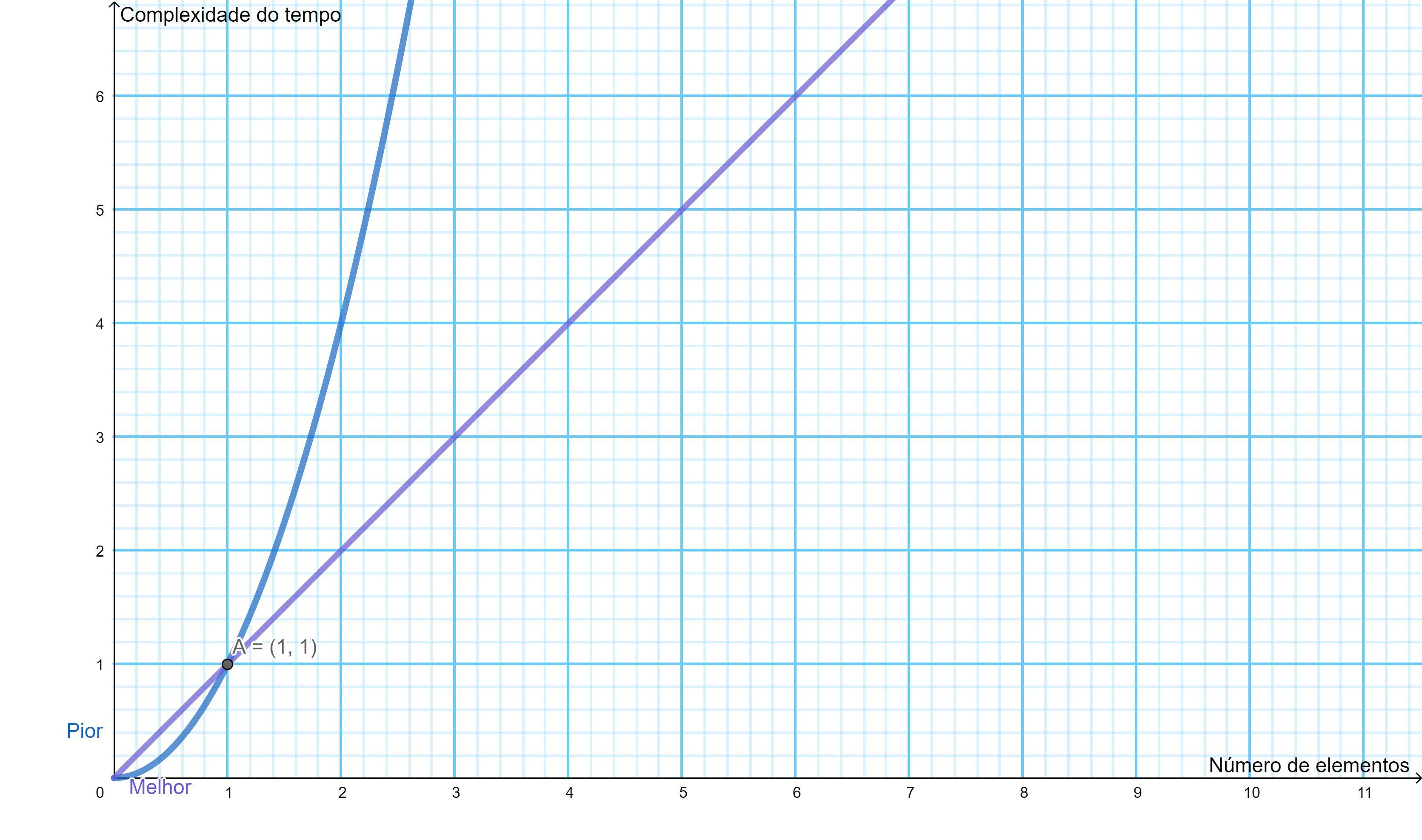


Figura 4

# Shell Sort

## Ideia Geral:

O algoritmo de ordenação Shell Sort tem como princípio classificar os elementos distantes uns dos outros e reduz sucessivamente o intervalo entre esses elementos a serem ordenados. É uma versão generalizada do Insertion Sort como já estudado anteriormente.

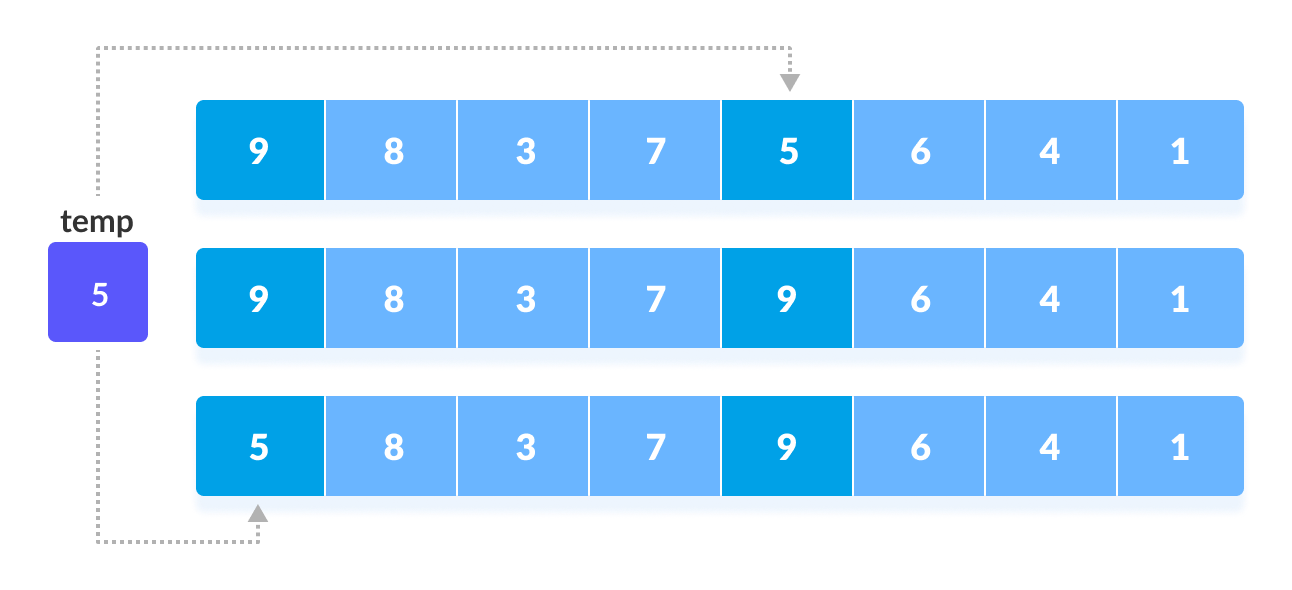
Nesse método de ordenação os elementos em um intervalo específico são ordenados. Esse intervalo entre os elementos vai gradualmente diminuindo baseado na sequencia usada. A sua performance depende do tipo de sequência inserida como input. Esse método de ordenação é instável pelo fato de que não examina os elementos presentes entre os intervalos.

Há algumas diferentes maneiras de criar a sequencia do algoritmo, a versão original é dada por: N/2, N/4, ..., 1. Onde N é o no de elementos do vetor(input).

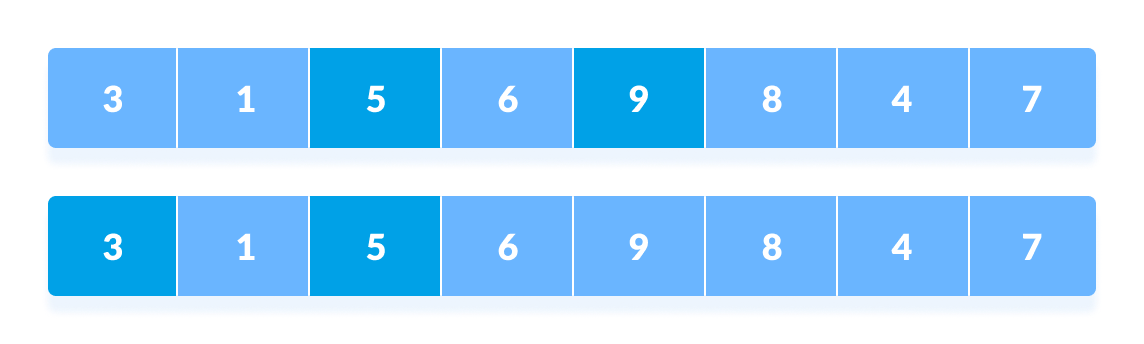
Computacionalmente falando podemos descrever o Shell Sort da seguinte maneira:

Seja o vetor a ser ordenado: |9|8|3|7|5|6|4|1|.

1. Utilizando a versão original de geração de intervalos do shell sort (N/2, N/4, ..., 1). Temos o tamanho do vetor N = 8, no primeiro loop os elementos que ocupam o invertalo até N/2 = 4, são comparados e invertidos caso não estejam em ordem. O elemento da posição 0 (9) é comparado com o elemento da posição 4 (5), se o elemento da posição 0 é maior do que o elemento da posição 4, então o elemento da posição 4 é armazenado em uma variável temporária e o elemento da posição 0 agora é transferido para a posição 4 e o elemento que se encontra na variável temporária vai para a posição 0.  
    Assim obtemos a seguinte situação: (9 > 5) |9|8|3|7|9|6|4|1| temporário = 5; por fim posição 0 recebe o temporário|5|8|3|7|9|6|4|1|. Esse ainda no loop agora o intervalo está suscetivelmente entre 1-5, 2-6, 3-7, e então ele executa os mesmos processos que anteriormente gerando: (1-5) (8 > 6) |5|8|3|7|9|8|4|1|temporário = 6; |5|6|3|7|9|8|4|1|. (2-6) (3 < 4), não precisa de nenhuma ação. (3-7) (7 > 1) |5|6|3|7|9|8|4|7| temporário = 1; |5|6|3|1|9|8|4|7|. Ilustração desse loop nas figuras abaixo.



1. No segundo loop o novo tamanho do intervalo é N/4 = 2, e assim vai executando com as mesmas regras vistas anteriormente pelos intervalos (0-2), (1-3), (2-4), (3-5),(4-6) e (5-7) gerando: (0-2) (5 > 3) |3|6|5|1|9|8|4|7|; (1-3) (6 > 1) |3|1|5|6|9|8|4|7|; (2-4) (5 < 9) mas ao mesmo tempo as posições 0 e 2 também são comparadas (intervalos anteriores ao atual); E assim segue conforme as ilustrações abaixo.





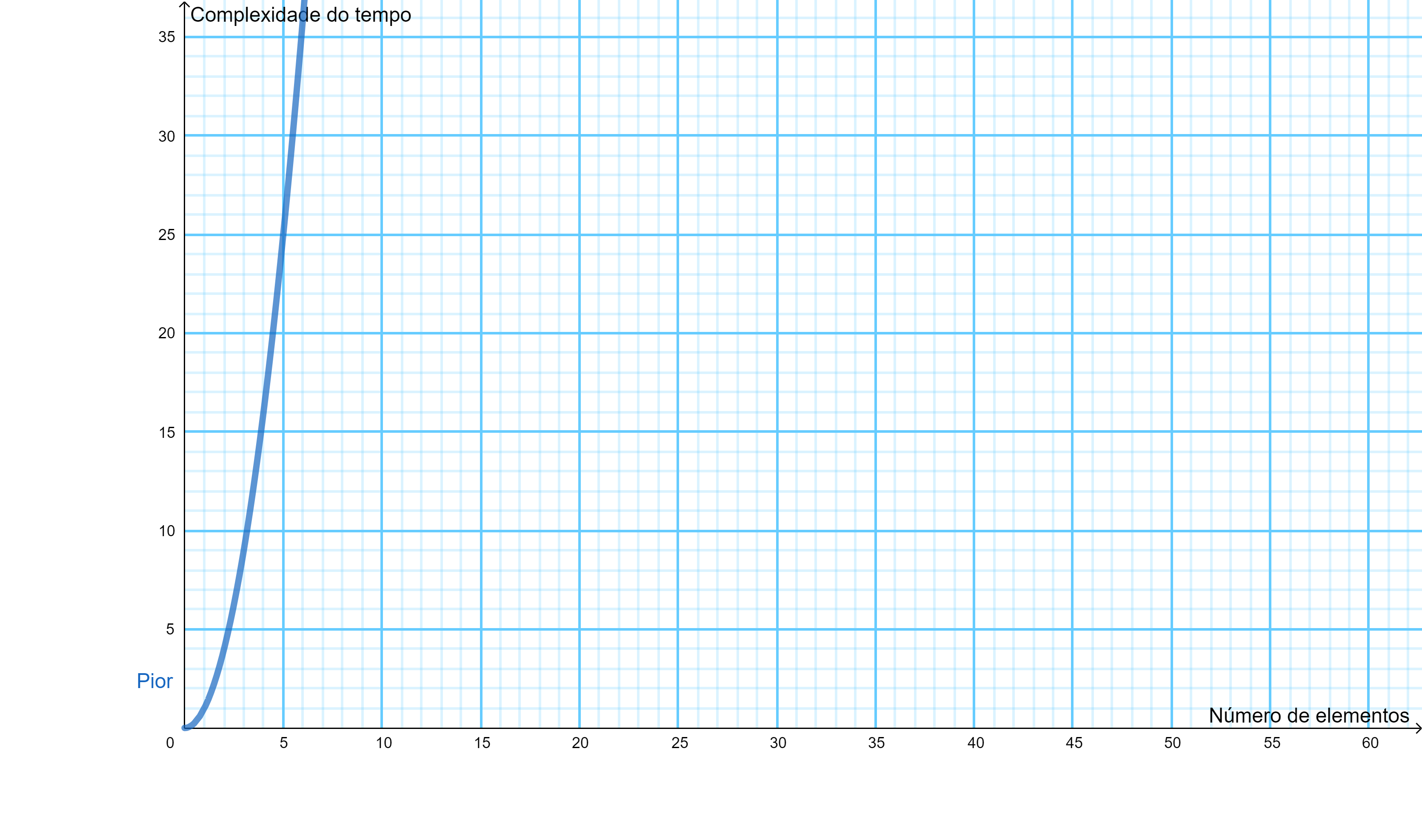
1. Agora temos o intervalo N/8 = 1. Assim terminando a ordenação, nota-se que poucos estão fora de posição.



## Complexidade:

Pior Caso O(n2):

O pior caso do Shell Sort é sempre menor ou igual a O(n2). Pelo teorema de Poonen a complexidade do pior caso é ou ou 2 ) ou algo entre esses valores.



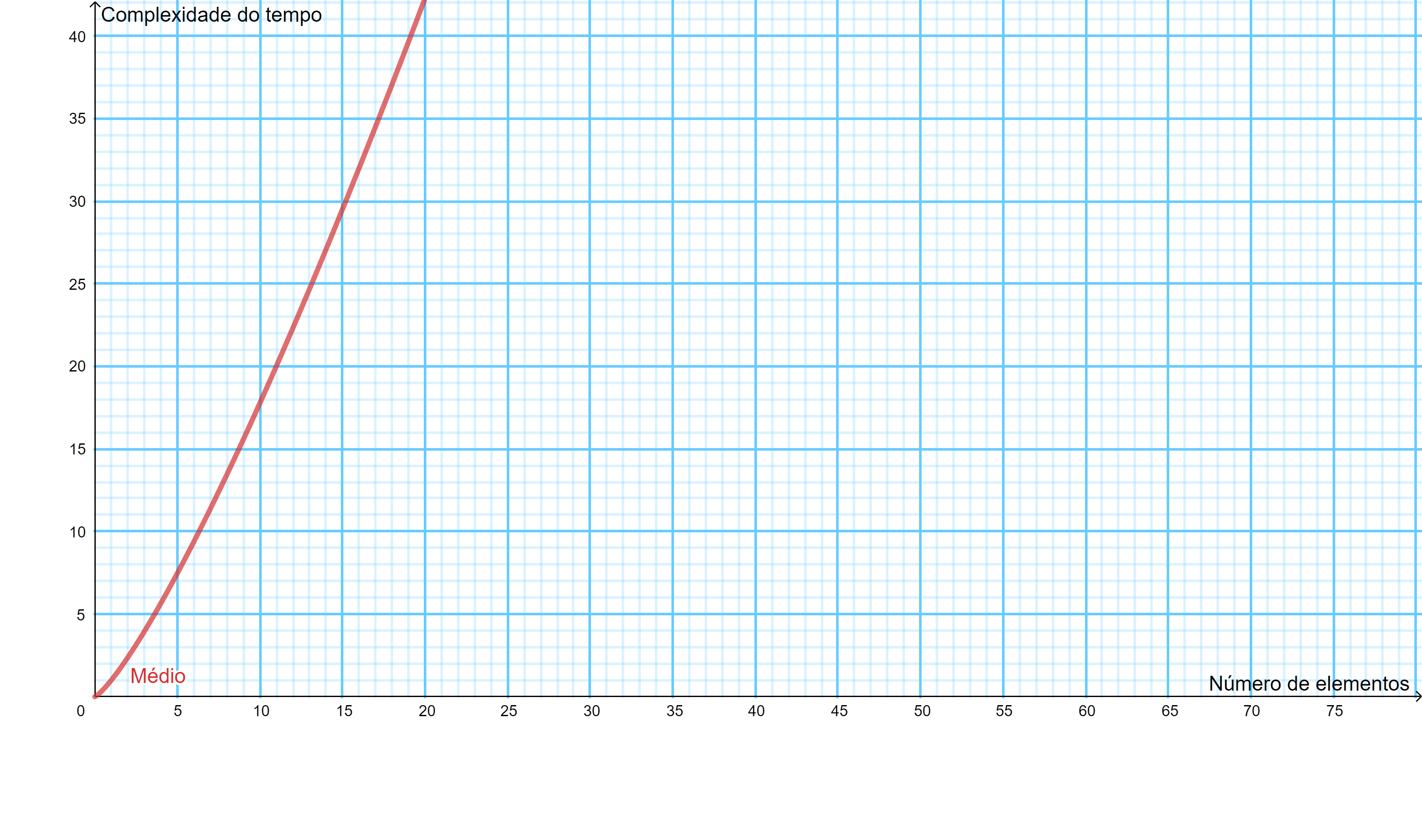
Melhor Caso O(n.log(n)):

Quando o vetor já está ordenado, o número total de comparações para cada intervalo (ou incremento) é igual ao tamanho do vetor.

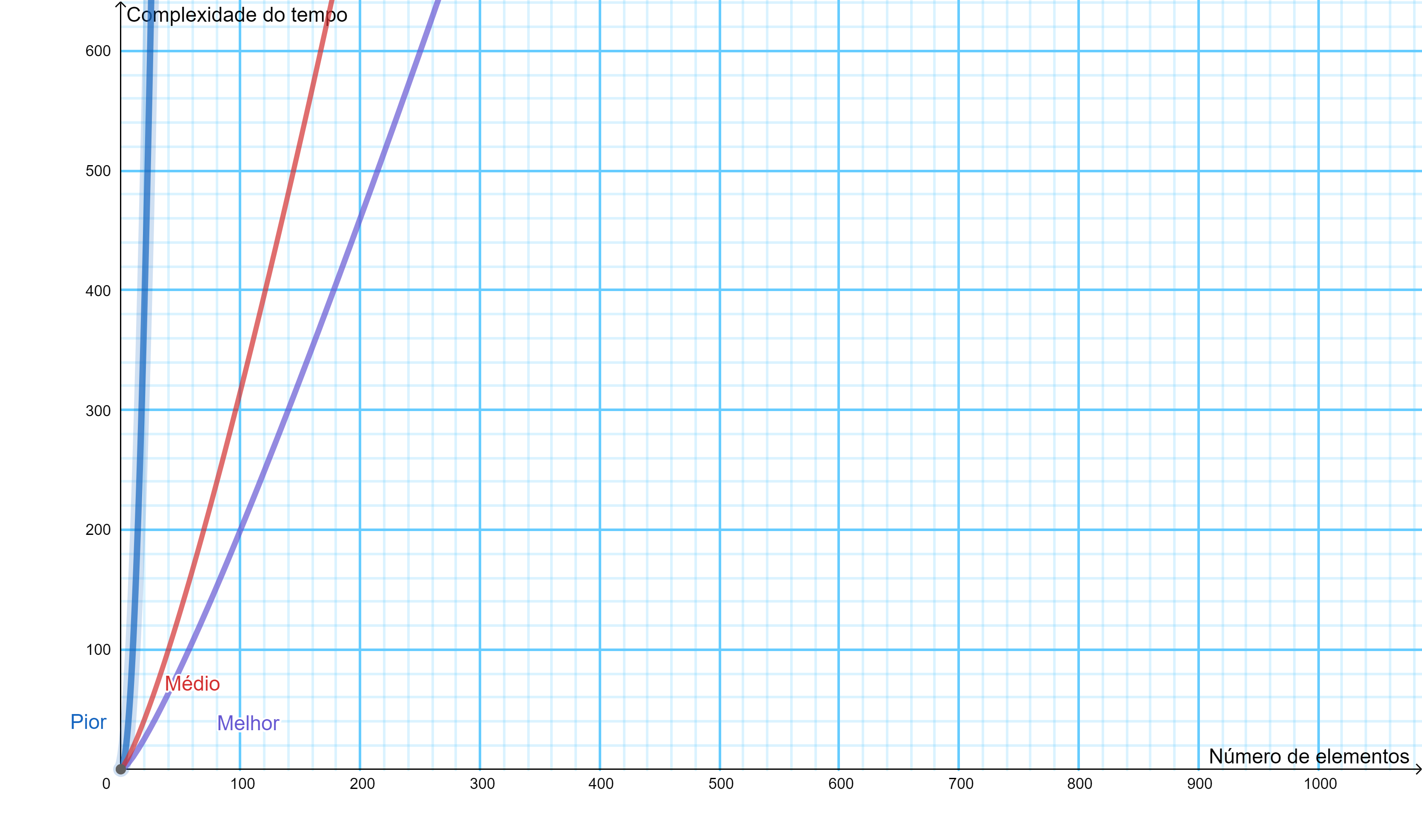


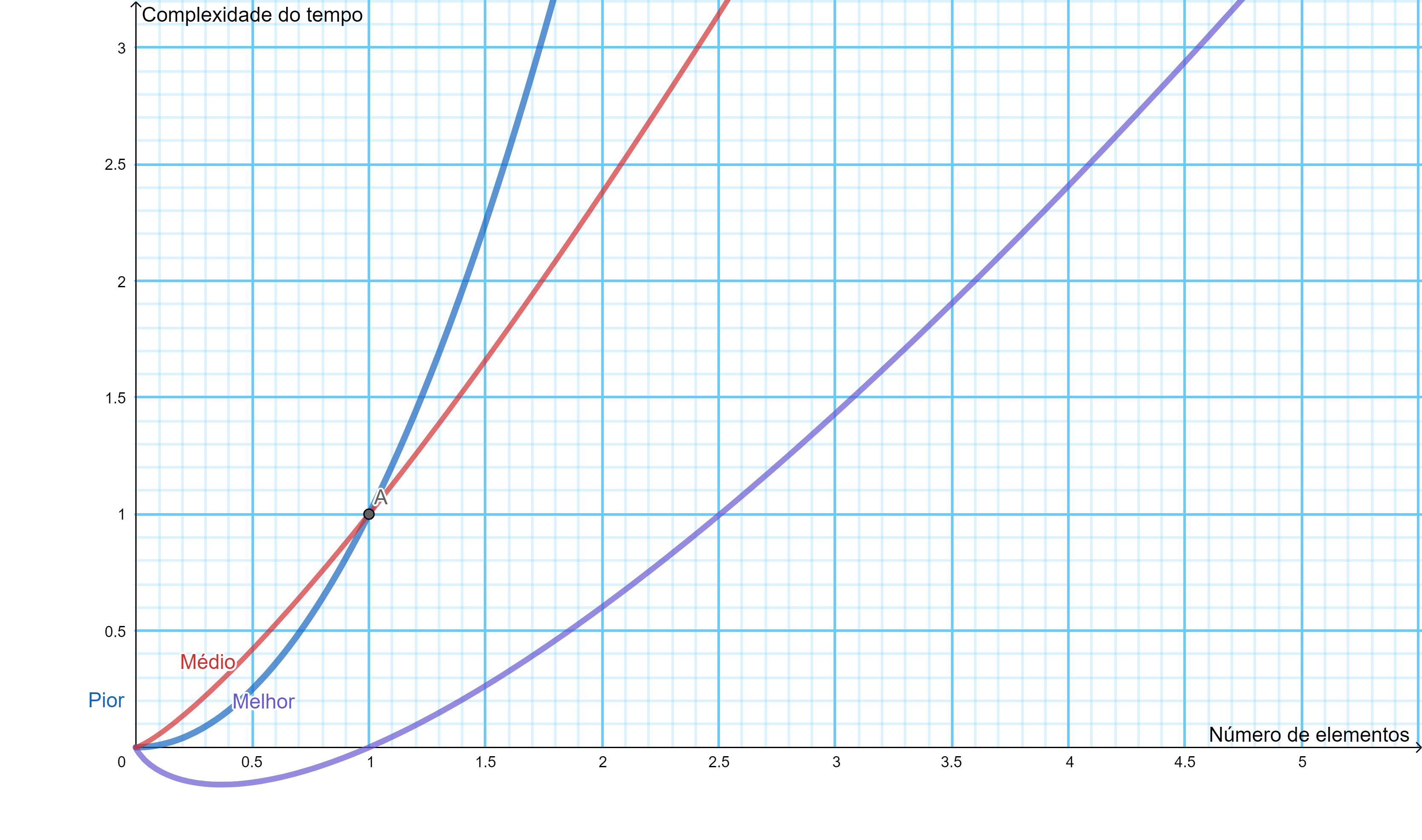
Médio Caso O(n.log(n)):

Que é por volta de O(n1.25 ).



Obervando o comportamento das funções no gráfico fica claro a diferença entre os casos de complexidade. Vale notar que o melhor caso será sempre melhor literalmente, e só há intersecção entre o pior e o médio caso no ponto (1,1).



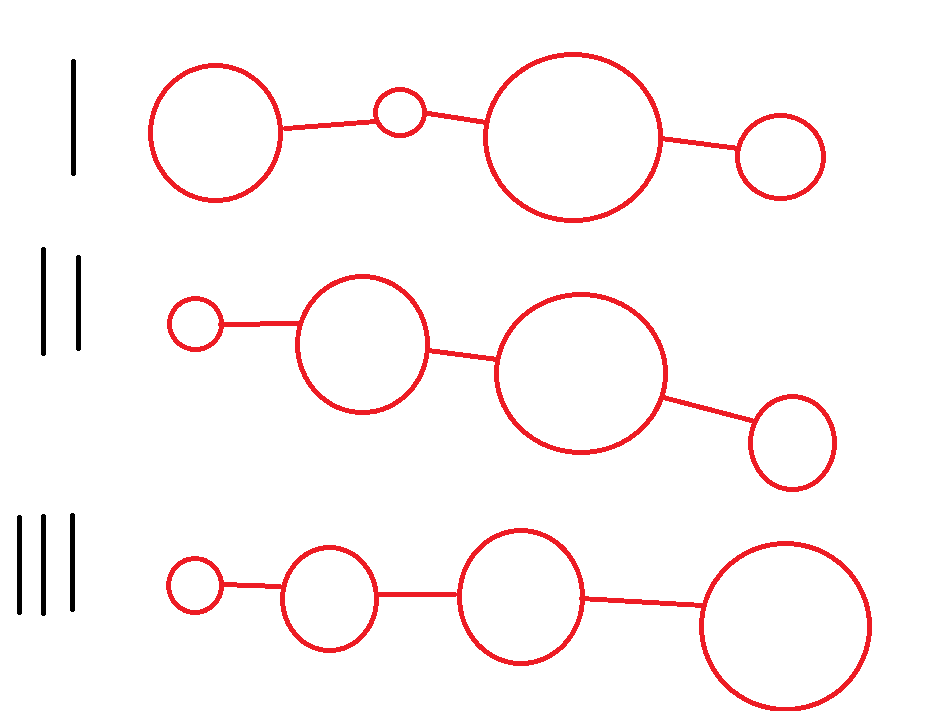


A complexidade depende do intervalo escolhido. As complexidades acima diferem para diferentes sequências de incremento escolhidas. A melhor sequência de incremento é desconhecida.

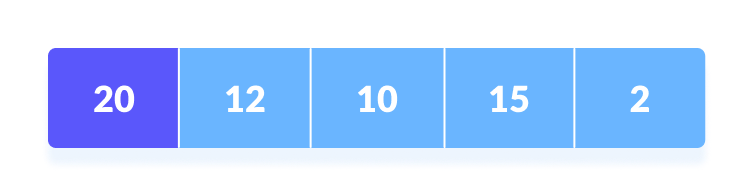
# Selection Sort

## Ideia Geral:

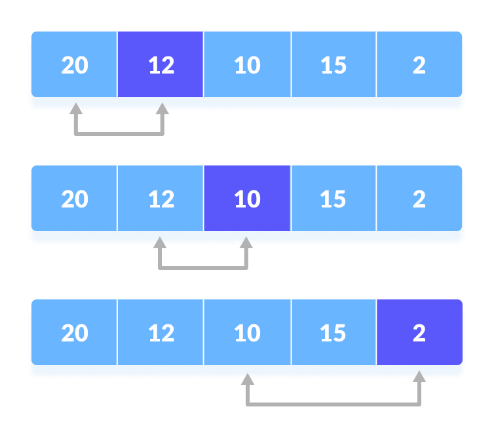
Em suma esse algoritmo seleciona o menor elemento de uma lista desordenada em cada iteração e o coloca esse elemento no início da lista desordenada.



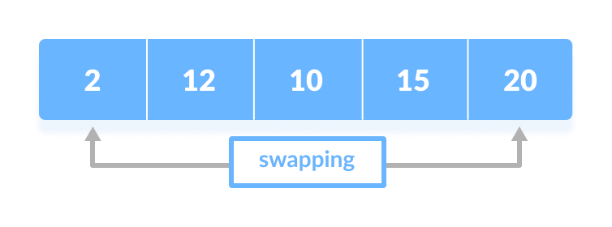
Detalhando seu comportamento, seja o vetor:

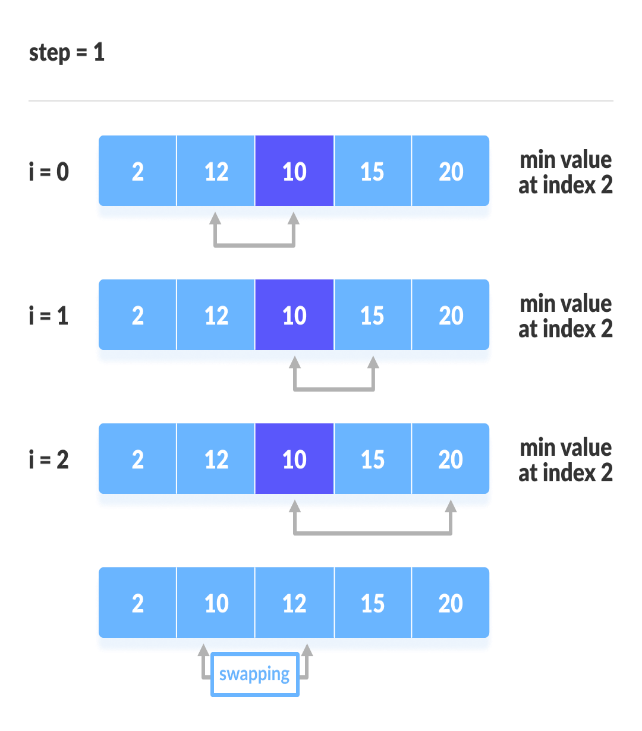
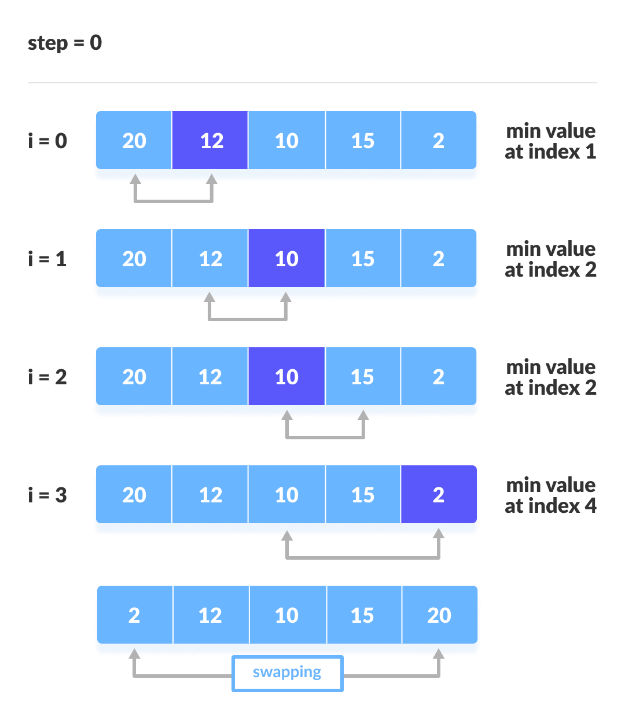


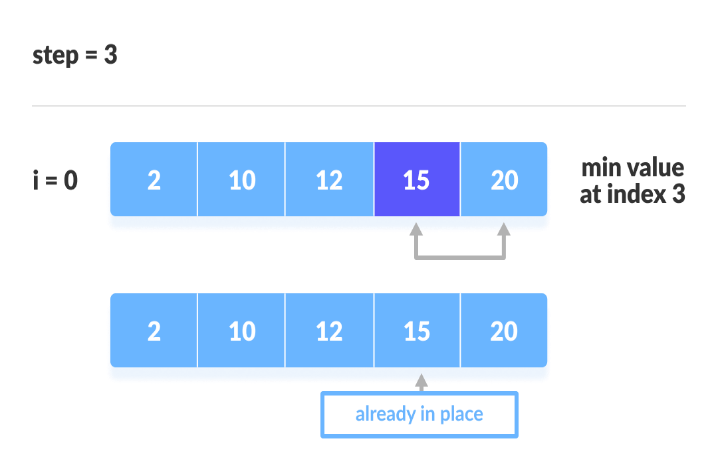
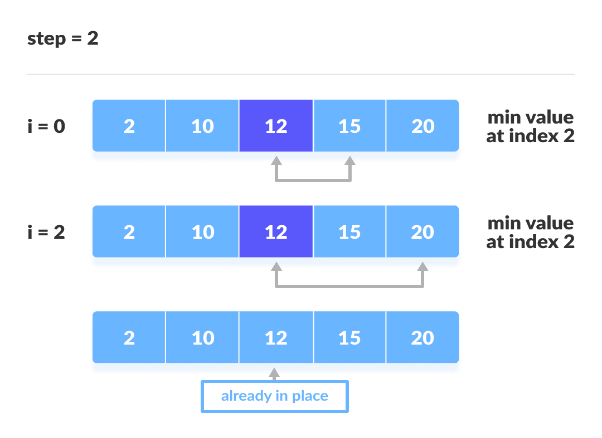
Inicialmente o algoritmo define o primeiro elemento como o mínimo do conjunto (20) e então compara com o próximo elemento e verifica se ele é menor ou maior, caso seja menor o novo número se torna o novo mínimo e continua sucessivamente como vemos abaixo:



Após cada iteração faz a troca do elemento mínimo com o elemento de onde a posição começou no conjunto de iterações:





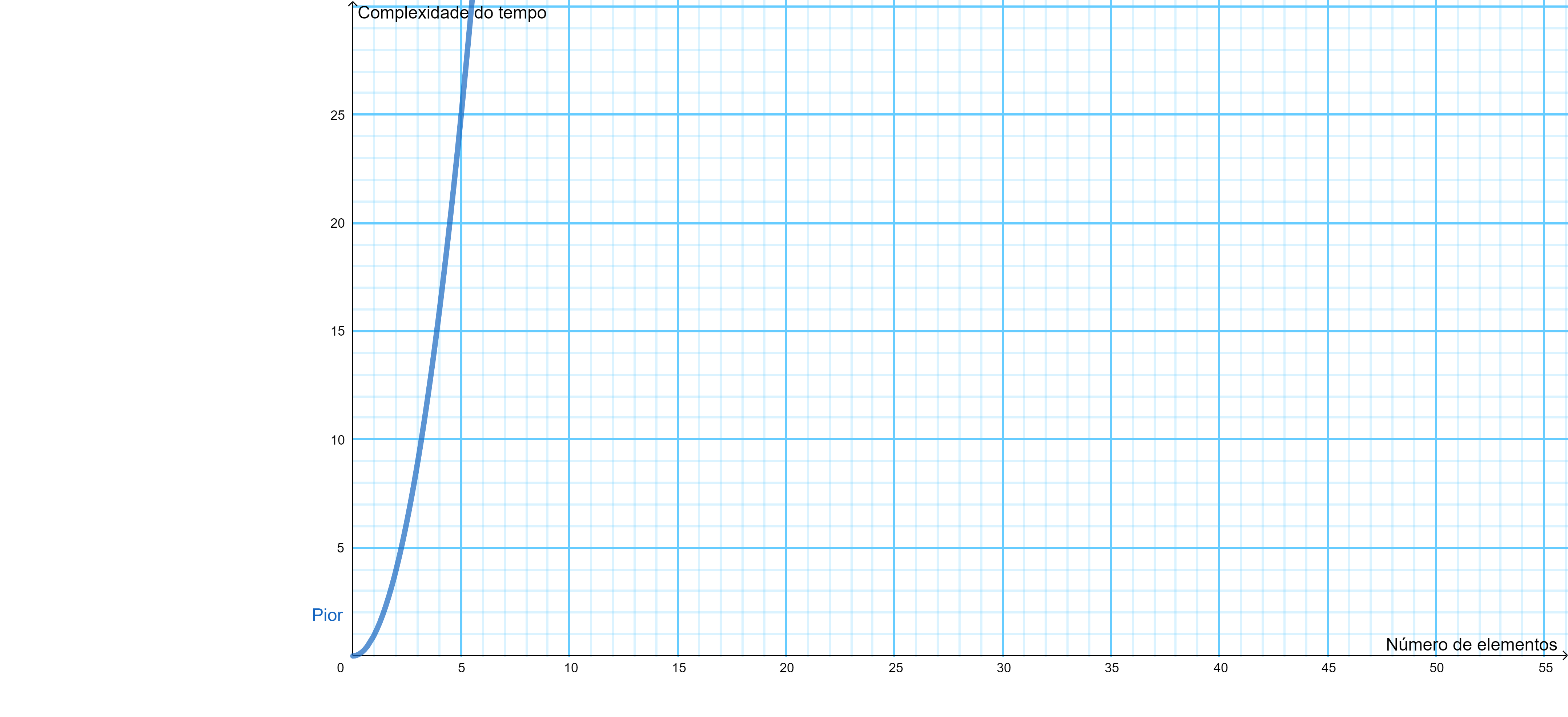


## Complexidade:

Pior Caso O(n2):

Isso é definido com base nos ciclos de execução, por exemplo:

Ciclo 1: n-1 comparações + ciclo 2: n-2 + ... + 1 = .



Melhor Caso O(n2):

Ocorre quando a lista já está ordenada.

Caso Médio O(n2):

Ocorre quando os elementos do vetor estão randomizados (nem ascendente nem descendente).

A complexidade do Insertion sort é a mesma em todos os casos, pois a cada passo, deve-se encontrar o elemento mínimo e coloca-lo em seu devido lugar e o elemento mínimo não é conhecido até que se percorra todo o vetor.