Análises Sobre

Os

Algoritmos De Ordenação

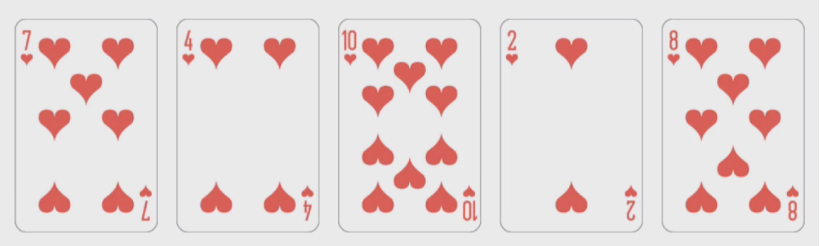
**Todos os testes realizados tiveram n (tamanho) definidos como: 1.000, 5.000, 10.000, 50.000, 100.000, 500.000 e 1.000.000, 5.000.000). Para testar com um tamanho diferente, basta ir na linha 5 do arquivo .c e alterar a macro parametrizada para atribuir o n desejado a variável MAX. Todos os algoritmos 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):

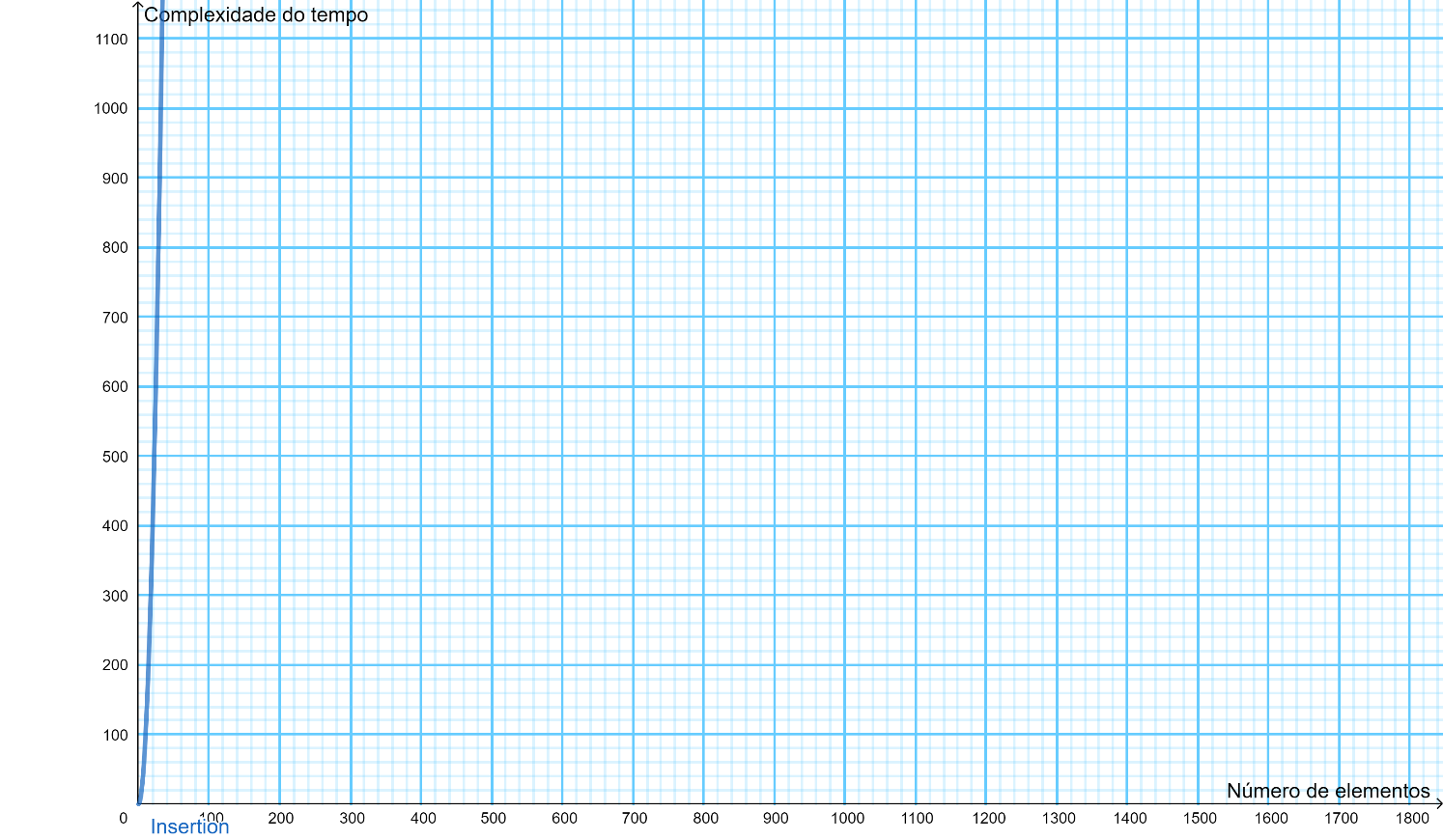
 Supondo que o vetor esteja ordenado de forma decrescente e quer ordenar em forma crescente, nesse momento é configurado o seu pior caso, pois cada elemento tem que ser comparado com os outros elementos, ou seja, para elemento na posição n do vetor, serão feitas n-1 comparações. Logo o numero de comparações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

Figura 1

Melhor caso θ(n):

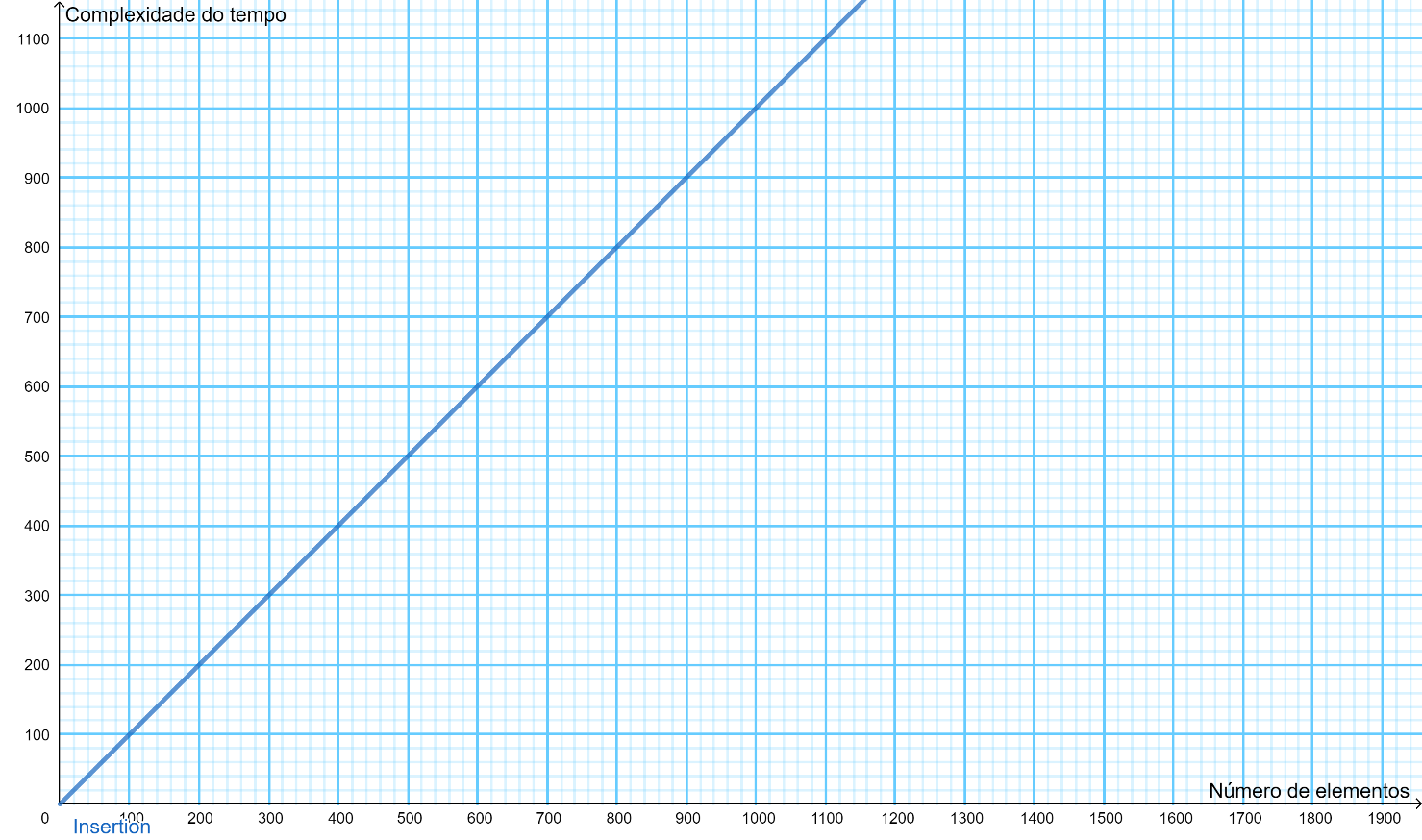
 Quando o vetor já se encontra ordenado então não é necessário fazer alterações logo somente o loop mais externo que vai até n será executado fazendo n comparações.

Figura 2

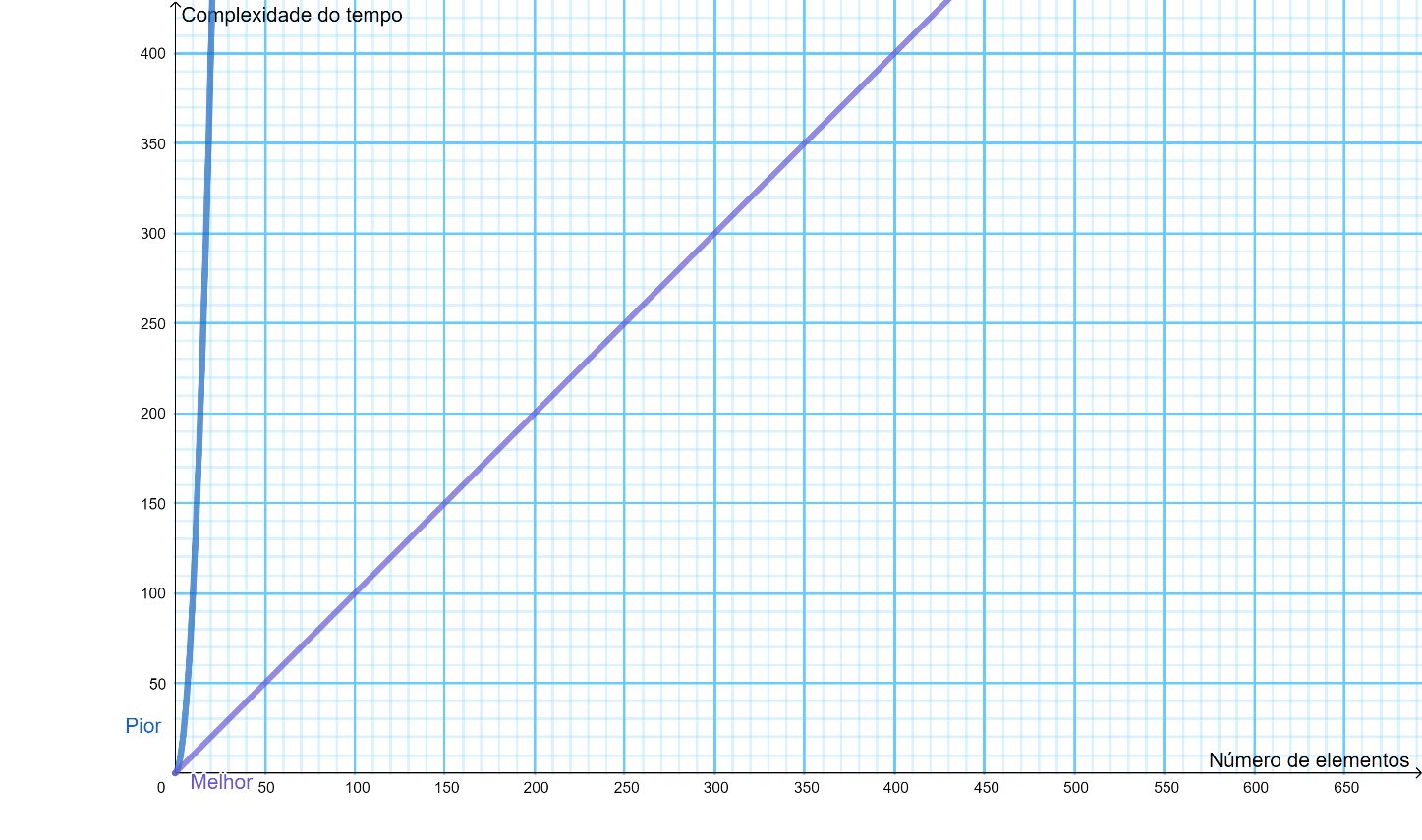
Nota-se pelo gráfico das funções logo abaixo nas figuras 3 e 4: Conforme a quantidade de elementos aumenta o tempo aumenta e no pior caso mesmo com uma quantidade baixa de elemento no vetor o tempo é extremamente maior observa-se melhor a discrepância na Figura 3 e observa-se também na Figura 4 o momento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

Figura 3

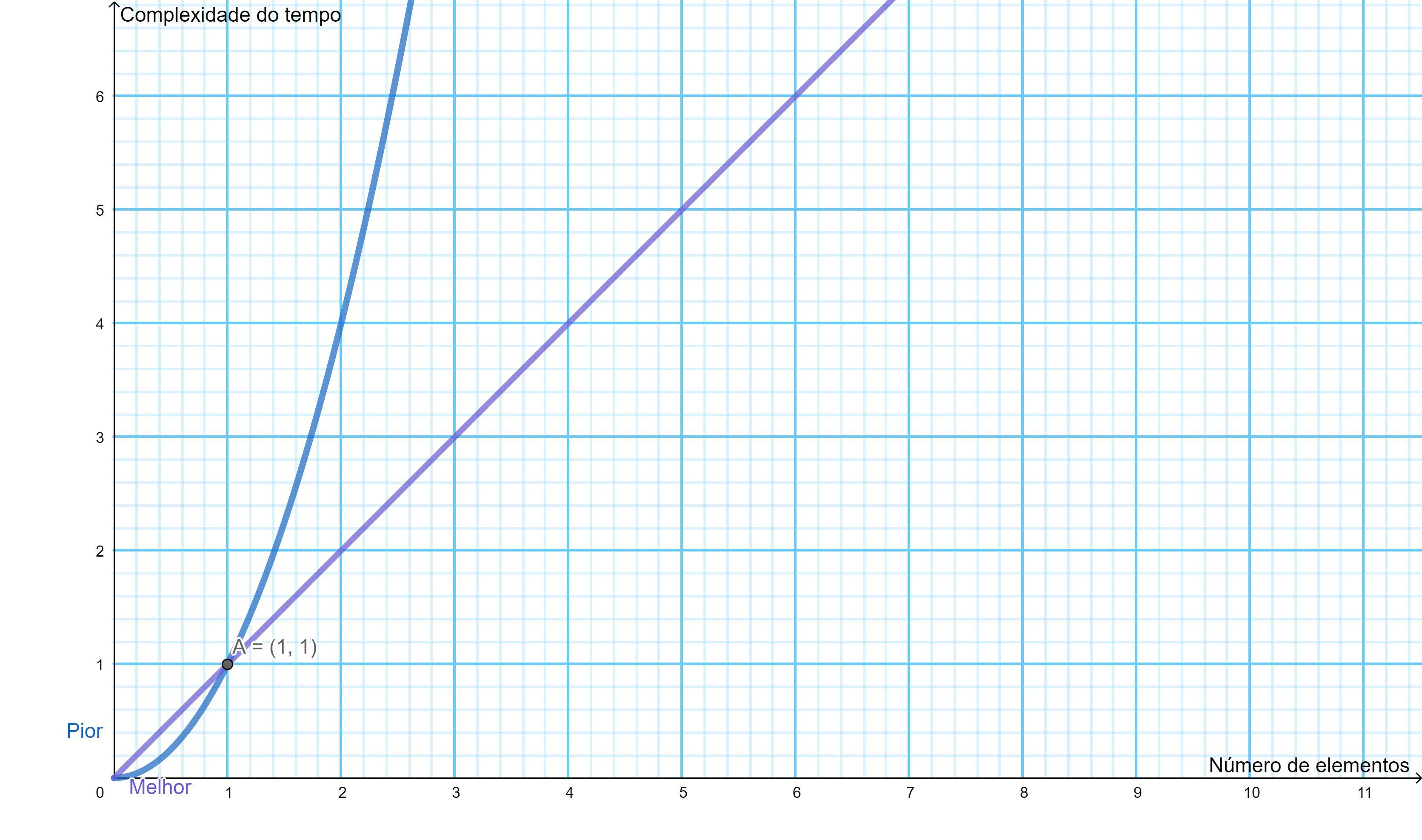


Figura 4