# Busca Binária

## Introdução

A **Busca Binária** é um algoritmo que facilita a busca por um elemento numa lista de elementos ordenados. Vamos supor um problema base:

> Dado um vetor de $n$ números inteiros ordenados ($n < 10^7$),
> e um valor inteiro $x$, queremos um programa que nos diga:

- a posição (de 0 a $n-1$) de $x$ no vetor (supomos que os elementos são todos distintos), caso $x$ esteja na sequência;
- $-1$, caso contrário.

A primeira abordagem que vem à mente é busca linear.

## Busca Linear

A busca linear consiste em verificar, de um em um, na sequência dos números dados, se o número analisado é igual ao valor buscado.
Apesar de parecer uma boa estratégia, esse algoritmo pode ficar lento demais para valores maiores de $n$. Logo, surge a necessidade de algo mais otimizado. Então, temos a Busca Binária.

## Busca Binária

Esse algoritmo se aproveita de como os elementos da sequência já começam ordenados. Assim, podemos começar por um ponto arbitrário, como o ponto central, e, então, nos orientarmos de acordo com a difereça entre o valor buscado e o valor central.
Uma implementação básica seguiria os seguintes passos:

1. Defina $lo$ (menor valor) como 0, e $hi$ (maior valor) como $(n-1)$;
2. Defina $mid$ como a parte inteira de $\frac{hi+lo}{2}$. Esse será o valor analisado;
3. Se $vetor[mid]$ é igual a $x$, $mid$ é o valor que queremos retornar e podemos acabar o algoritmo por aqui. Caso contrário, se $vetor[mid] < x$, podemos

fazer $lo = (mid + 1)$, já que já achamos um valor mínimo para nossa resposta. Se $vetor[mid] > x$, fazemos $hi = (mid - 1)$, analogamente ao caso contrário. Se $lo \leq hi$, podemos voltar ao **Passo 2**, já que os valores mínimo e máximo da resposta;

4. Só podemos chegar nesse passo se $lo > hi$. Ou seja, o intervalo em que buscamos $x$ ficou vazio. Logo, $x$ não está na sequência e retornamos $-1$.

## Complexidade

Por fim, podemos analisar a complexidade do algoritmo. Podemos notar que, para cada intervalo de busca, fazemos uma operação de comparação, e, a cada iteração, o intervalo de busca é dividido ao meio. Ou seja, o número de operações é igual à quantidade de vezes que podemos dividir $n$ por 2 até atingirmos 1. Essa é, incidetalmente, a definição de $log_2(n)$. Assim, podemos definir a complexidade desse algoritmo como $logn$.