BUSCA LINEAR

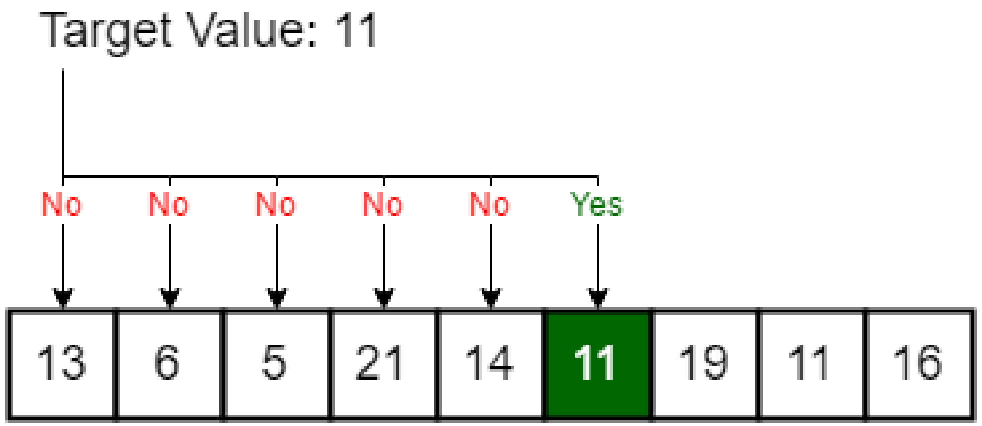
Aluno: Mércio

BUSCA LINEAR

- Entrada: um array A[1..n] e um valor x a ser procurado (valor alvo)
- Saída: um índice i tal que A[i] = x ou –1, caso x não esteja contido no array.

PSEUDOCÓDIGO

```
busca-linear(A, x)
   i <- 1
  enquanto i <= comprimento[A]
    se A[i] = x
      retorne i
    senão i <- i + 1
  se  i > comprimento[A]
    retorne -1
```



INVARIANTE DE LOOP

- Usamos invariantes de loop para nos ajudar a entender por que um algoritmo é correto.
- **Inicialização**: ele é verdadeiro antes da primeira iteração do loop.
- **Manutenção**: se for verdadeiro antes de uma iteração, ele permanecerá verdadeiro antes da próxima iteração.
- **Término**: Quando o loop termina, o invariante nos fornece uma propriedade útil que ajuda a mostrar que o algoritmo é correto.

INVARIANTE DE LOOP – BUSCA LINEAR

- Ao início de cada iteração do laço, temos que: **A[j] != x para todo inteiro j tal que 1 <= j <= i – 1.**
- i – 1 equivale ao índice da iteração anterior.
- Isso quer dizer que: até a última iteração, anterior a atual, o valor alvo não foi encontrado.

```
 i <- 1
Enquanto i <= comprimento[A]
    Se A[i] = x
       Retorne i
    Senão i <= i + 1  // i é incrementado a cada iteração
```

INICIALIZAÇÃO

- Ao início da primeira iteração, temos que i = 1 e o invariante será uma **afirmação vazia** (empty statement) já que não existe um inteiro j tal que 1 <= j <= 0(i-1), portanto ele será verdadeiro.

MANUTENÇÃO

- Suponhamos antes de uma certa iteração tenhamos A[j] != x para 1 <= j <= i – 1.
- Se A[i] = x ou i > comprimento[A] o loop finaliza e não há mais iterações.
- Senão, teremos durante a iteração, que A[j] != x para 1 <= j <= i.
- Ao decorrer da iteração, incrementamos i em 1, isso significa que teremos A[j] != x para 1 <= j <= i - 1 antes da próxima execução do loop.
- O invariante se mantém verdadeiro.

TÉRMINO

- Quando i <= comprimento[A] e A[i] = x o algoritmo termina.
- Quando i > comprimento[A], a condição de guarda do loop é quebrada e o algoritmo termina e retorna -1.
- Nesse caso, antes da quebra da condição de guarda, temos i = comprimento[A + 1].
- De acordo com o invariante, isso nos diz que A[j] != x para j = 1, ..., comprimento[A], (comprimento de [A + 1 - 1]) portanto, não existe nenhum j tal que 1 <= j <= comprimento[A] e A[j] = x, ou seja, o valor alvo não está presente no array.

INSERTION SORT - EXERCÍCIO

- Provar a corretude através de invariante de laço.

```
INSERTION-SORT(A)
  para j = 2 até comprimento[A]
     chave = A[j]
     // inseri A[j] na sequencia ordenada A[1..j-1]
     i = j - 1
     enquanto i > 0 e A[i] > chave
        A[i+1] = A[i]
        i = i - 1
     A[i+1] = chave
```

INVARIANTE DE LOOP

- A cada iteração, A[1..j-1] , contém os primeiros j-1 elementos ordenados

INICIALIZAÇÃO

- Antes do loop, j = 2 -> A[1..j-1] = A[1], que contém A[1..j-1] elementos.
- O array contém apenas um elemento, portanto ele está ordenado.

MANUTENÇÃO

- O loop for externo acessa o elemento A[j] e o insere de forma ordenada no array A[1..j-1], através do loop while.
- Considerando que o array A[1..j-1] começa ordenado, inserir o elemento A[j] de forma ordenada, produz o array ordenado A[1..j], contendo os primeiros j elementos.

TÉRMINO

- O loop termina quando j = n + 1 -> A[1..j-1] = A[1..(n + 1) - 1] = A[1..n]
- Como o array se mantém ordenado após cada iteração, temos que A[1..n] estará ordenado quando o loop termina
- A[1..n] contém todos os elementos dos array original ordenados

OBRIGADO!