Let’s Go with Algo

By Melakesleam Moges

18-11-2023

BLAST FROM THE PAST

Quick Quiz

1. **What is an Algorithm?**
2. **Name two common Strategies used in Algorithms**
3. **What are the basic parameters used to evaluate a possible solution(Algorithm) to a problem?**

BLAST FROM THE PAST

Quick Quiz

1. What is an Algorithm?
   a. **a step-by-step procedure for solving a problem or accomplishing some end**
2. Name two common Strategies used in Algorithms
   a. **Brute Force and Divide and Conquer**

BLAST FROM THE PAST

Quick Quiz

3. What are the basic parameters used to evaluate a possible solution(Algorithm) to a problem?
   a. **practicality - doable with available resources,**
   b. **time efficiency - can be done with the time available using the tools and technologies available**
   c. **result producing - it will have a measurable and finite output**

LECTURE 4

Fundamental Algorithms

PROPERTIES OF USEFUL ALGORITHM

1. **Accepts 0 or More Inputs**
2. **Generates 1 or More Outputs**
3. **It is Feasible**
4. **It Terminates**
5. **It is Correct**
6. **It has Precise and Definite Steps**
    a. Meaning of step should not be confusing
7. **It is Computable**
8.
9. **Its Steps have logical order**
10. **Each step is necessary**
11. **It produces result**

LIMITED RESOURCES NEEDED BY AN ALGORITHM

1. **Memory**
2. **Processing Power (Time)**
3. **Network Speed and Bandwidth**

etc…

Common Tasks in Computing

Performance depends on the underlying data structure

Primitive - pure

1. Search

   Finding an element in a list
   - *Reading Operation*

2. Sort

   rearranging elements of a list
   - *Reading Operation*
   - *Swapping Positions Operation*

Common Tasks in Computing

Performance depends on the underlying data structure

Derivative - combination

1. Insert
   - *Finding location in a list - Searching Operation*
   - *Make space - Shifting Operation*
   - *Add new element - Writing Operation*
2. Update
   - *Finding element in a list - Searching Operation*
   - *Change to new element - Writing Operation*

Common Tasks in Computing

Performance depends on the underlying data structure

3. Delete
- *Finding element in a list - Searching Operation*
- *Remove element - Writing Operation*
- *Truncate list - Shifting Operation*

Fundamental Algorithms of Searching

Used to locate an element in a List

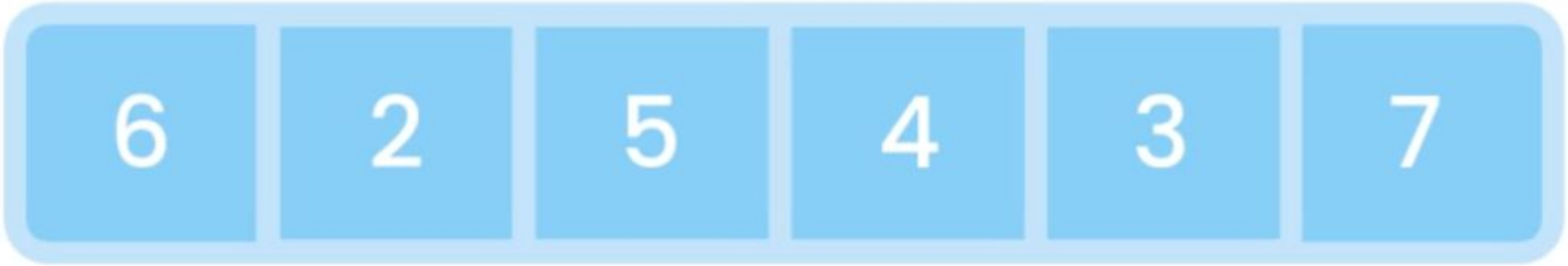
1. Linear Search
2. Binary Search (N-ary Search)
3. Jump Search
4. Exponential Search

Searching - Linear

Basic searching algorithm that can be applied on any list regardless of the order of the elements

1. Iterate over the list starting from one end
2. Compare each element with the value of the target
3. If found, return the index of the element
4. If NOT found, return a negative number (typically -1)

Searching - Linear



Searching - Linear

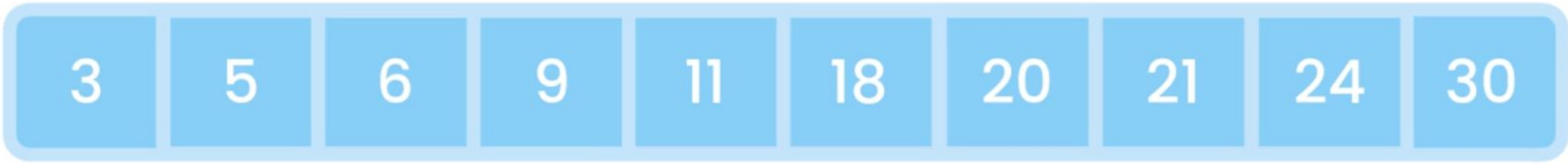
```
Algorithm linearSearch(L, n, target)
    INPUT list L of n elements
                target element to be found
    OUTPUT index of element in L OR -1

    FOR i ← 1 TO n-1 DO
        IF L[i] == target THEN
            RETURN i
    RETURN -1
```

Searching - Binary (N-ary Operations)

This searching algorithm required the list to be searched in to be arranged in ascending or descending order. It is able to eliminate half of the list each time a comparison is made.

1. Set the Left and Right Boundary of the current List as Index 0 and length of List -1.
2. If Left Boundary is Greater Than Right Boundary, return a negative number (typically -1), the Target is NOT found in the List
3. Find the Middle element
4. If the Target element is equal to the Middle element, return the index of the element
5. If the Target element is Less Than the Middle element, set the Right Boundary to be the Middle Index - 1 and GoTo Step 2
6. If the Target element is Greater Than the Middle element, set the Left Boundary to the Middle Index + 1 and GoTo Step 2

Searching - Binary (N-ary Operations)

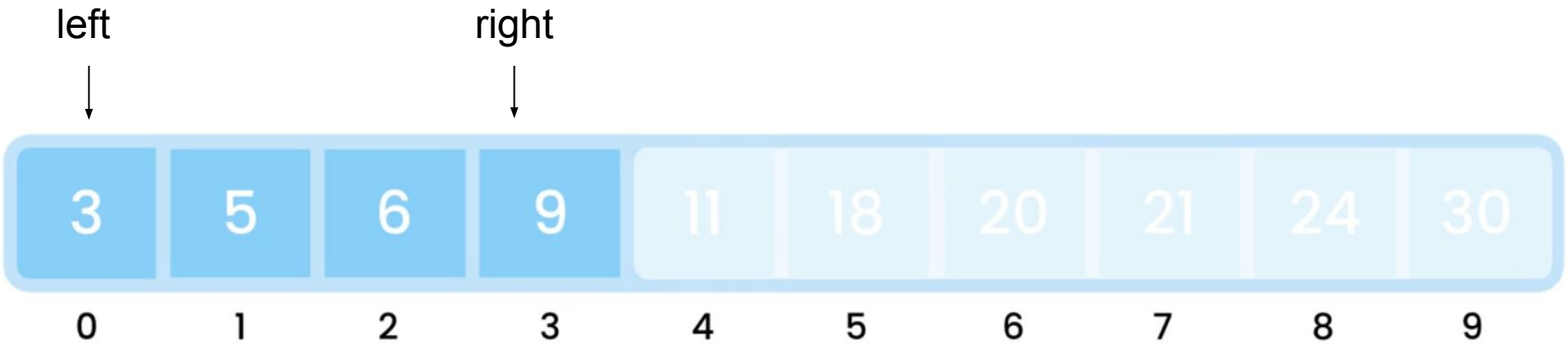


middle = (left + right) / 2

= (0 + 9) / 2

= 4

Searching - Binary (N-ary Operations)



middle = (left + right) / 2

Searching - Binary

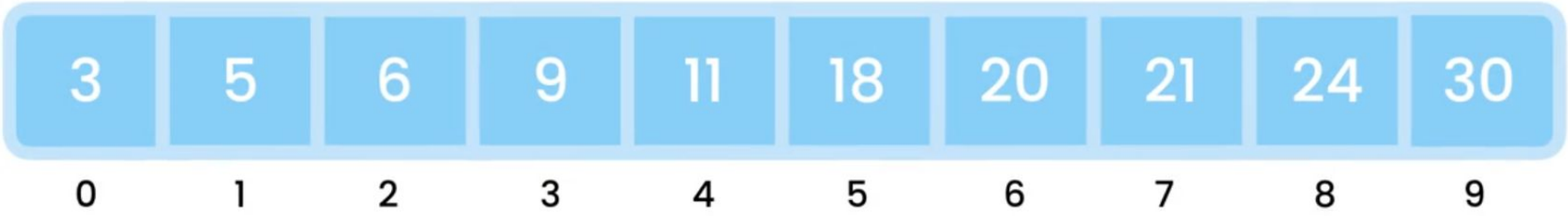
```
Algorithm BinarySearch(L, n, target)
    INPUT list L of n elements
                target element to be found
    OUTPUT index of element in L OR -1
    l ← 0, r ← len(L) -1
    WHILE l <= r DO
        mid ← (r - l) / 2
        IF target == L[mid] THEN
            RETURN mid
        ELSE IF target < L[mid] THEN
            r ← mid - 1
        ELSE
            l ← mid + 1
    RETURN -1
```

Searching - Ternary (N-ary Operations)

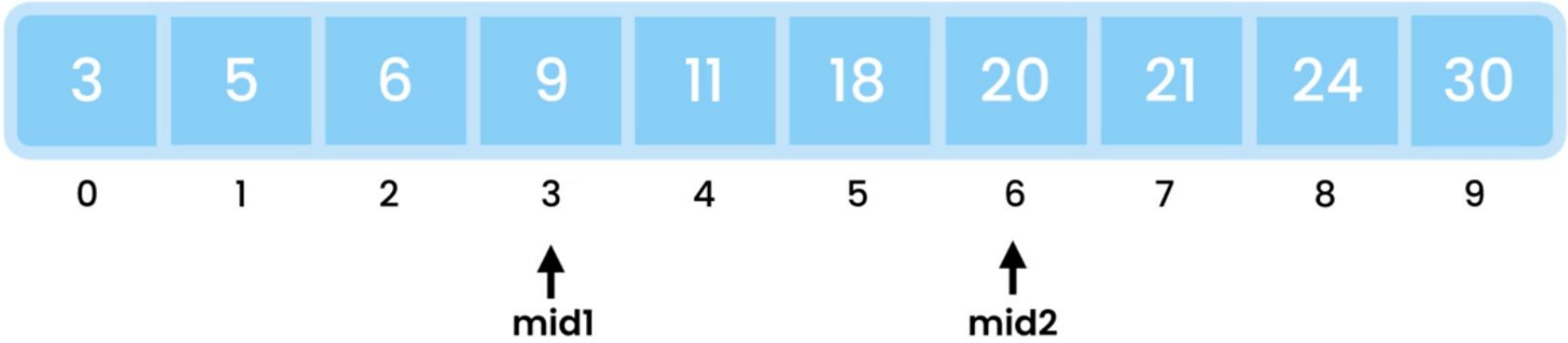
This searching algorithm is like Binary Search but instead of dividing into TWO parts, it divides into THREE parts. It is able to eliminate Two Thirds of the list each time a comparison is made.

1. Set the Left and Right Boundary of the current List as Index 0 and length of List -1.
2. If Left Boundary is Greater Than Right Boundary, return a negative number (typically -1), the Target is NOT found in the List
3. Find the Two Middle element.
4. If the Target element is equal any to the Two Middle elements, return the index of the element
5. Adjust the Left and Right Boundaries depending on where the Target is expected to exist and GoTo Step 2

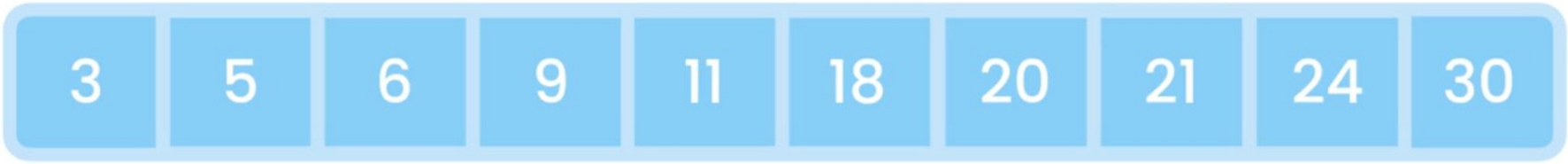
Searching - Ternary (N-ary Operations)



Searching - Ternary (N-ary Operations)



Searching - Ternary (N-ary Operations)



partitionSize = (right - left) / 3

mid1 = left + partitionSize

mid2 = right - partitionSize

Searching - Linear

```
Algorithm TernarySearch(L, n, target)
    INPUT list L of n elements
                target element to be found
    OUTPUT index of element in L OR -1
    l ← 0, r ← len(L) -1
    WHILE l <= r DO
        partition ← (r - l) / 3
        mid1 ← l + partition, m2 ← r - partition
        IF target == L[mid1] THEN
            RETURN mid1
        ELSE IF target == L[mid2] THEN
            RETURN mid2
        ELSE IF target < L[mid1] THEN
                r ← mid1 - 1
        ELSE IF target > L[mid1] AND target < L[mid2] THEN
            l ← mid1 + 1, r ← mid2 - 1
        ELSE
                l ← mid2 + 1
    RETURN -1
```

Which is the Best Performing N-ary Searching Algorithm?

Class Discussions

Which is the Best Performing N-ary Searching Algorithm?

BINARY

$\log_2 n$

target == mid
target > mid
target < mid

TERNARY

$\log_3 n$

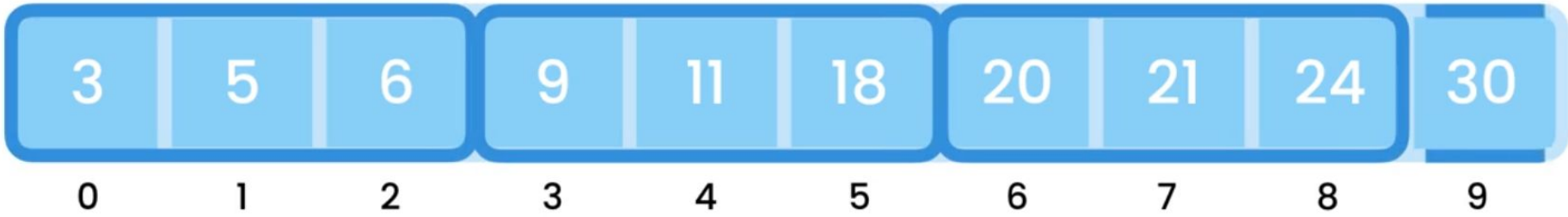
target > mid2
target == mid2
target < mid2 && target > mid1
target == mid1
target < mid1

Searching - Jump

Like Binary Search, this searching algorithm required the list to be searched in to be arranged in ascending or descending order. It is an improvement on Linear Search but not as Fast as Binary Search.

1. Divide the List into several blocks (ideally blocksize is $\sqrt{(n)}$).
2. Jump to the block that could contain the target element
    a. Define current block boundry
    b. If Target element is Less Than the element at the Upper Boundry, Perform Linear Search on this Block
    c. Else check the next block until the end of the list
    d. Return negative number

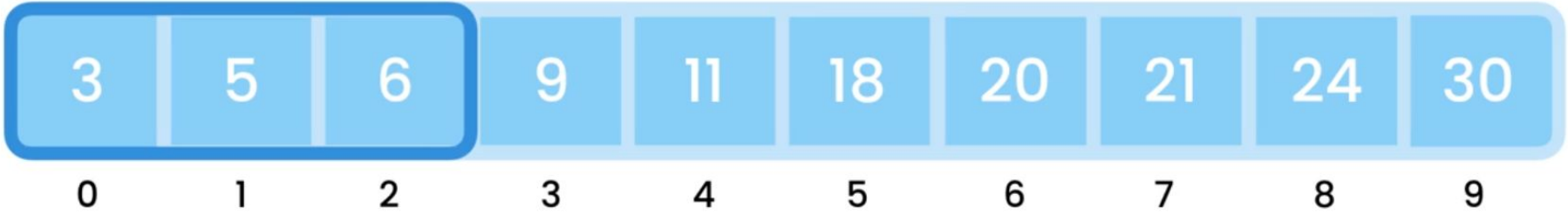
Searching - Jump



$$\text{blockSize} = \sqrt{n}$$

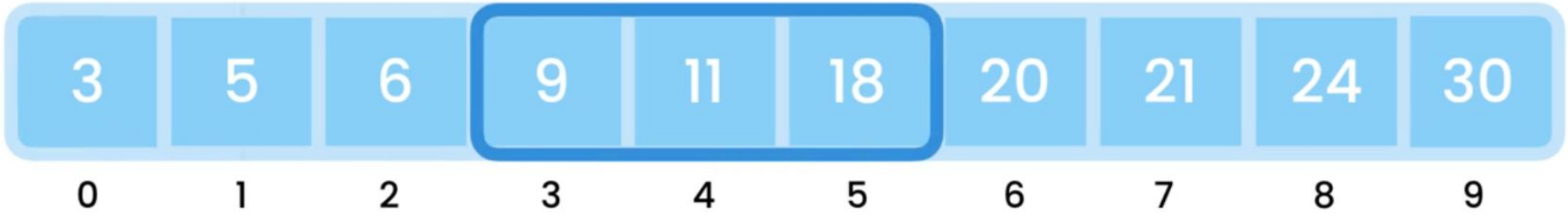
$$= 3$$

Searching - Jump



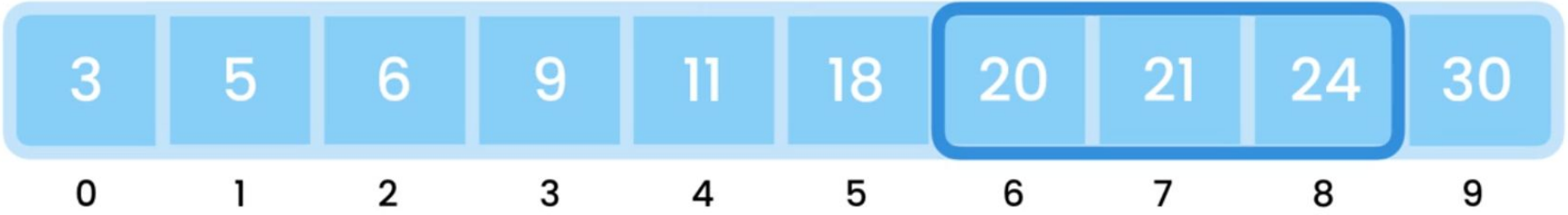
target = 23

Searching - Jump



target = 23

Searching - Jump



target = 23

Searching - Jump

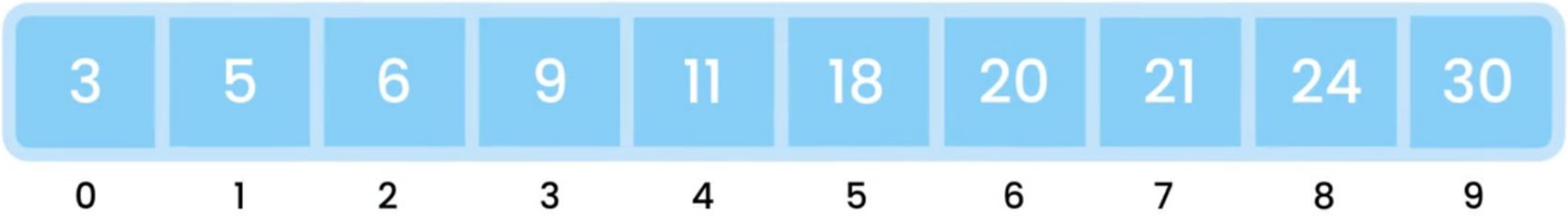
```
Algorithm BinarySearch(L, n, target)
    INPUT list L of n elements
                target element to be found
    OUTPUT index of element in L OR -1
    blocksize ← floor(sqrt(n))
    rangeDiff ← blocksize -1
    index ← -1
    FOR i ← rangeDiff TO n-1 DO
        IF target <= L[i] THEN
            index ← LinearSearchRange(L, target, i - rangeDiff, i)
        IF index >= 0 RETURN index
        i ← min(i + blocksize, n-1)
    RETURN -1
```

Searching - Exponential

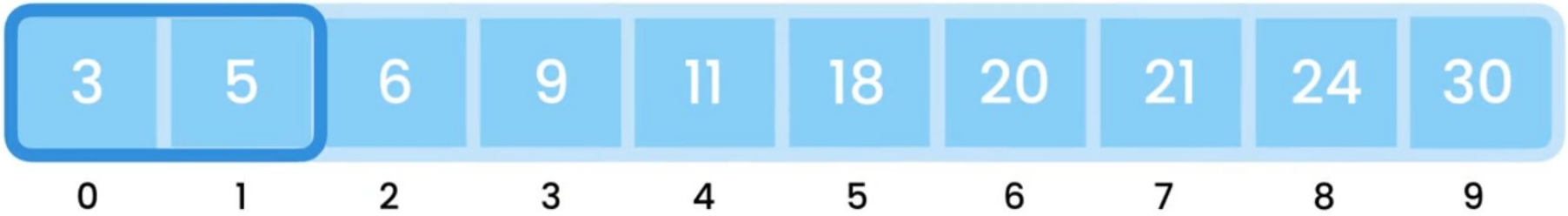
Like Binary Search, this searching algorithm similar to Jump Search to identify the range in which the target is located.

1. Start with a small range
2. If the target is in that range, use Linear Search to find it in the range
3. If Not, double the range and try again

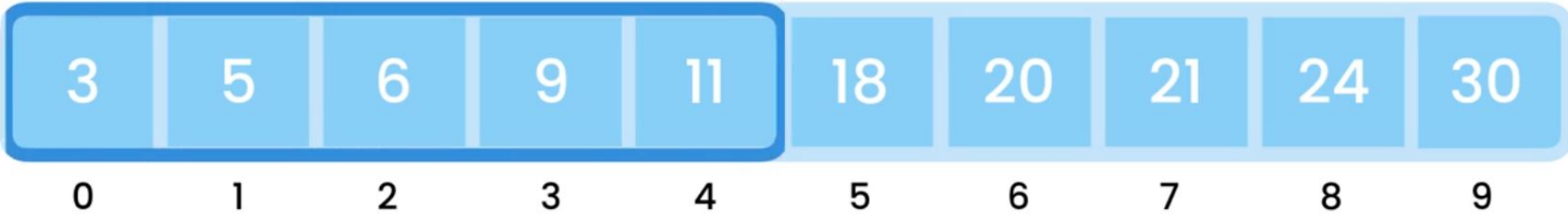
Searching - Exponential



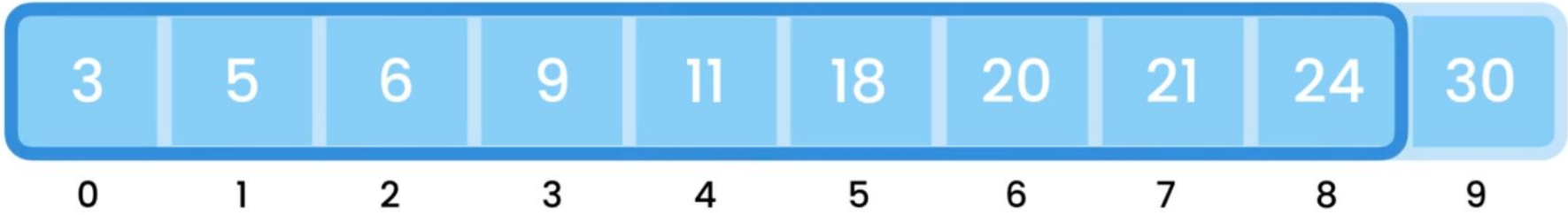
Searching - Exponential



Searching - Exponential



Searching - Exponential



Searching - Exponential

Exercise

Write the Pseudocode of this Algorithm.