Logarithmic Time Complexity: Takeaways

Syntax

- Sorting a list:

```
sorted_values = sorted(values)
```

- Binary search:

```
def binary_search(sorted_values, target_value):
    range_start = 0
    range_end = len(sorted_values) - 1
    while range_start < range_end:
        range_middle = (range_end + range_start) // 2
        value = sorted_values[range_middle]
        if value == target_value:
            return range_middle
        elif value < target_value:
            range_start = range_middle + 1
        else:
            range_end = range_middle - 1
    if sorted_values[range_start] != target_value:
        return -1
    return range_start
```

- Compute the base-b logarithm of N:

```
import math
base_b_log_N = math.log(N, b)
```

Concepts

- The number of times we need to divide a value N by 2 in order to reach 1 is called the base-2 logarithm of N.
- When data is unsorted, we cannot do better than linear time complexity for looking up an element in a list.
- Algorithms with logarithmic complexity are very efficient. Their runtime grows very slowly as the data grows. Using binary search to search into 1 TB sorted list of integers would require only 34 list lookups.

Resources

- Introduction to logarithms
- Logarithms
- Binary search algorithm