Linear Search

Linear Search

- The Lineary Search algorithm allows the user to search for a specific value in a list of values.
- The searching starts at the beginning.
- Stops when the item is found OR when the search reaches the end of the list.
- The worst-cast performance: O(N) linear

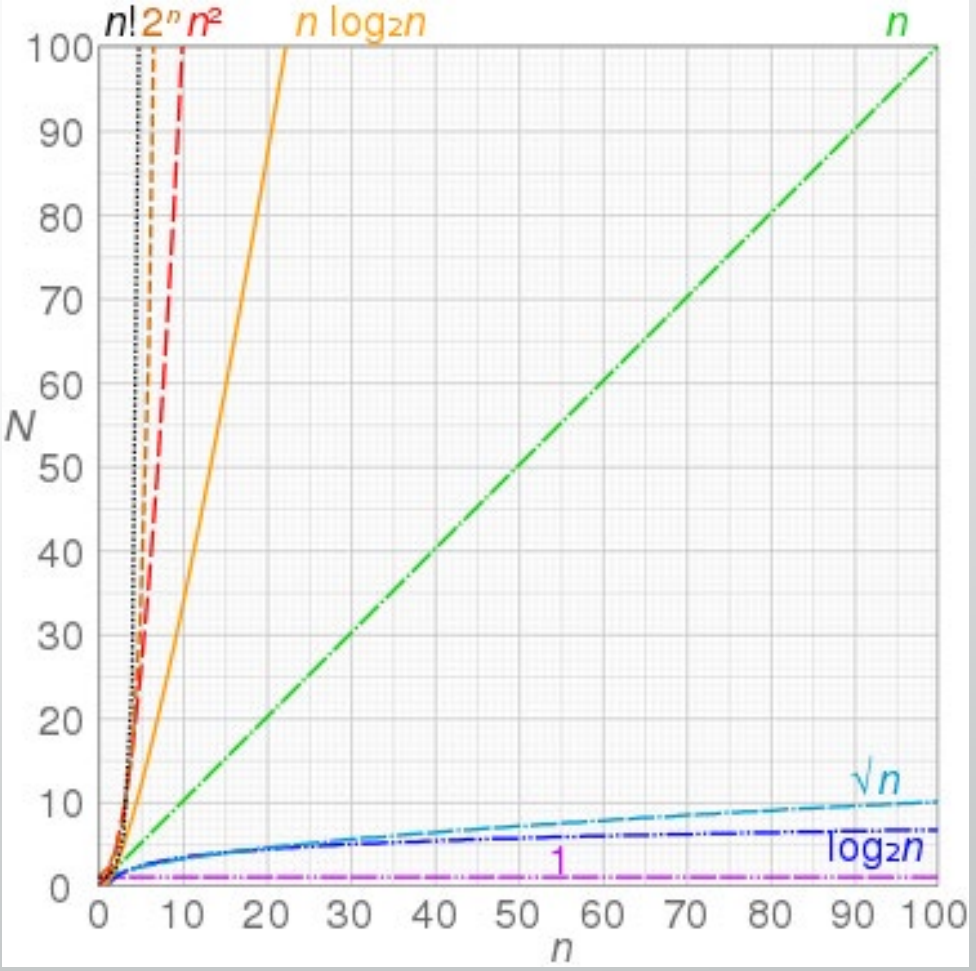
Binary Search

Binary Search Algorithm

- The algorithm is ***very efficient***! **O(log n)**
- **ONLY** works on sorted data
- Divides-and-conquers!

Binary Search Algorithm

O(1)	constant
O(log n)	logarithmic
O(n)	linear
O(n log n)	loglinear
O(n^2)	quadratic
O(2^n)	exponential
O(n!)	factorial



Binary Search Algorithm

- The Binary Search algorithm:
  - If the min index > max index, return `-1` (this is an exit condition)
  - If the middle item is your search term, quit! You found it!
    - Return the index of the middle item
  - Else if the search term is LESS than the middle item
    - repeat the search with the left half
  - Else if the search term is GREATER than the middle item
    - repeat the search with the right half

```
// initially called with low = 0, high = N-1
BinarySearch(A[0..N-1], searchTerm, low, high)
{
    if (high < low)
        return -1 // -1 means not found

    mid = (low + high) / 2

    if (searchTerm < A[mid])
      return BinarySearch(A, searchTerm, low, mid-1)

    else if (searchTerm > A[mid])
      return BinarySearch(A, searchTerm, mid+1, high)

    else
      return mid //the searchTerm was found so return its index

 }
```

```
// initially called with low = 0, high = N-1
BinarySearch(A[0..N-1], searchTerm, low, high)
{
    if (high < low)
        return -1 // -1 means not found

    mid = (low + high) / 2

    if (searchTerm < A[mid])
      return BinarySearch(A, searchTerm, low, mid-1)

    else if (searchTerm > A[mid])
      return BinarySearch(A, searchTerm, mid+1, high)

    else
      return mid //the searchTerm was found so return its index

 }
```

std::map<Tkey,Tvalue>

std::map<Tkey,Tvalue>

- std::map - cppreference.com
- std::map is a sorted, associative collection that stores key-value pairs where the keys are unique.
- The key is associated with the value.

std::map<Tkey,Tvalue>

- Example: We want to look up a specific student's information at Full Sail.
- If we use an array to store student info, how would we find the info for a specific student? We would have to loop over the entire array.
- With a map, we can jump to the student's record using the student's Id as the key.
- The student's Id is *associated* with the student's record.

std::map<Tkey,Tvalue>
Creating and Adding

std::map<Tkey,Tvalue>

- Need to `#include <map>`
- Replace Tkey with the type of the keys and Tvalue with the type of the values.
- Example: Let's say we want to store menu items (strings) as keys and the prices (double) as values…
- `std::map<std::string, double> menuPrices;`

std::map<Tkey,Tvalue>

- There are 2 ways to add key-value pairs to the map…
  1. Use the insert method
  2. Use [key] = value

std::map<Tkey,Tvalue>

The insert method

If the key is already in the map, insert will do nothing. It will return false in the returned pair (see below).

The insert method needs a **std::pair** argument sent to it.

```
std::map<std::string, double> menuPrices;
auto isInserted = menuPrices.insert(std::make_pair("Cheeseburger", 8.99));
```

isInserted has 2 parts:

1. First – which is an iterator to the item in the map (if it was inserted)
2. Second – a bool indicating if the key-value pair was added

std::map<Tkey,Tvalue>

Use [key] = value

- map[key] = value will insert/update the value for the key.
- If the key is already in the map, it will simply overwrite the existing value (which is different than the insert method).
- If the key is not in the map, it will add the key and the value.

```
std::map<std::string, double> menuPrices;
menuPrices["Curly Fries"] = 3.99;
```

std::map<Tkey,Tvalue>
Looping

std::map<Tkey,Tvalue>

You can iterate (loop) over a map in a couple of ways:

- Using iterators
- Using range-based for loop

std::map<Tkey,Tvalue>

Looping with iterators…

```
std::map<std::string, double>::iterator iter = menuPrices.begin();
while (iter != menuPrices.end())
{
    std::string menuName = iter->first;
    double price = iter->second;

    std::cout << menuName << ": " << price << "\n";
    ++iter;
}
```

std::map<Tkey,Tvalue>

Looping with range-based for loop (C++ v17 or higher)…

```
for (const auto& [name, price] : menuPrices)
{
}
```

std::map<Tkey,Tvalue>
Finding Keys

std::map<Tkey,Tvalue>

To check if a key is in the map, use the **find** method.

- **find** returns an iterator.
- If the iterator equals end(), then the key was not found.
- If the iterator does NOT equal end(), you have the item. Use second on the iterator to access the value for the key.

std::map<Tkey,Tvalue>

```
std::map<std::string, double>::iterator isFound = menuPrices.find("Chicken
Nuggets");

if(isFound != menuPrices.end()) //if true, we found the key
{ }
else //the key was not found
{ }
```

std::map<Tkey,Tvalue>
Updating Values

std::map<Tkey,Tvalue>

Use [key] = value

- If the key is already in the map, it will simply **overwrite the existing value**
- If the key is not in the map, it will add the key and the value.

```
std::map<std::string, double> menuPrices;
menuPrices["Curly Fries"] = 3.99;//adds key and value
menuPrices["Curly Fries"] = 5.99;//updates value
```