Johns Hopkins
Whiting School of Engineering

EN.540.635
Software Carpentry

Lecture 10
Searching Algorithms and Gradient-Based Minimization in Optimization Problems

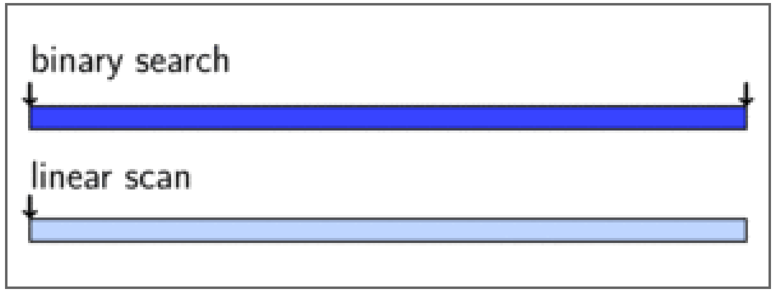
Searching

Finding the index of a value/object in a list:

- In a list, where is the maximum?
- In a list, where is the minimum?
- In a list, where is a specific value located?

Two main ideas:

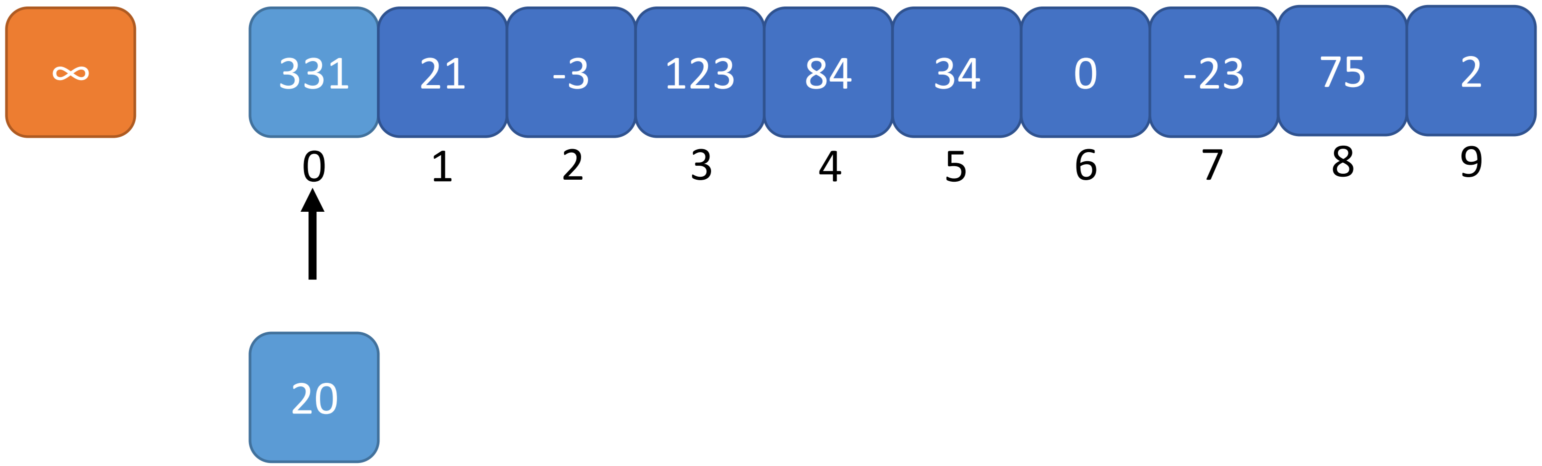
- Linear searching
- Binary searching



Linear Searching

Let's find the smallest number larger than 20 in this list

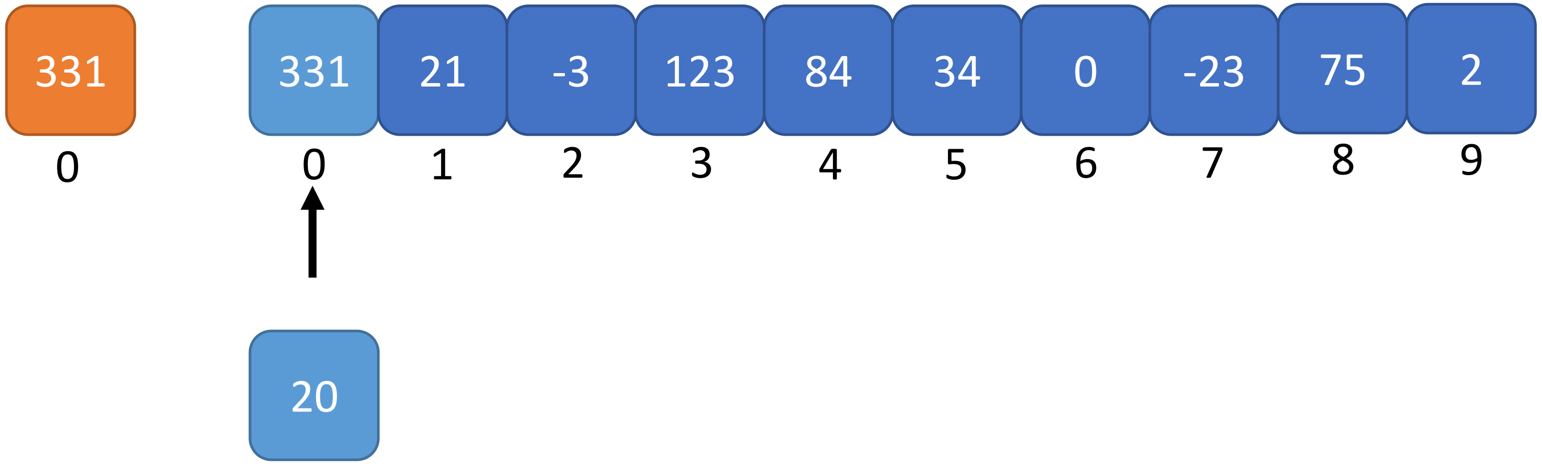
- Check if $N > 20$
- Check if $N < N_{prev}$



Linear Searching

Let's find the smallest number larger than 20 in this list

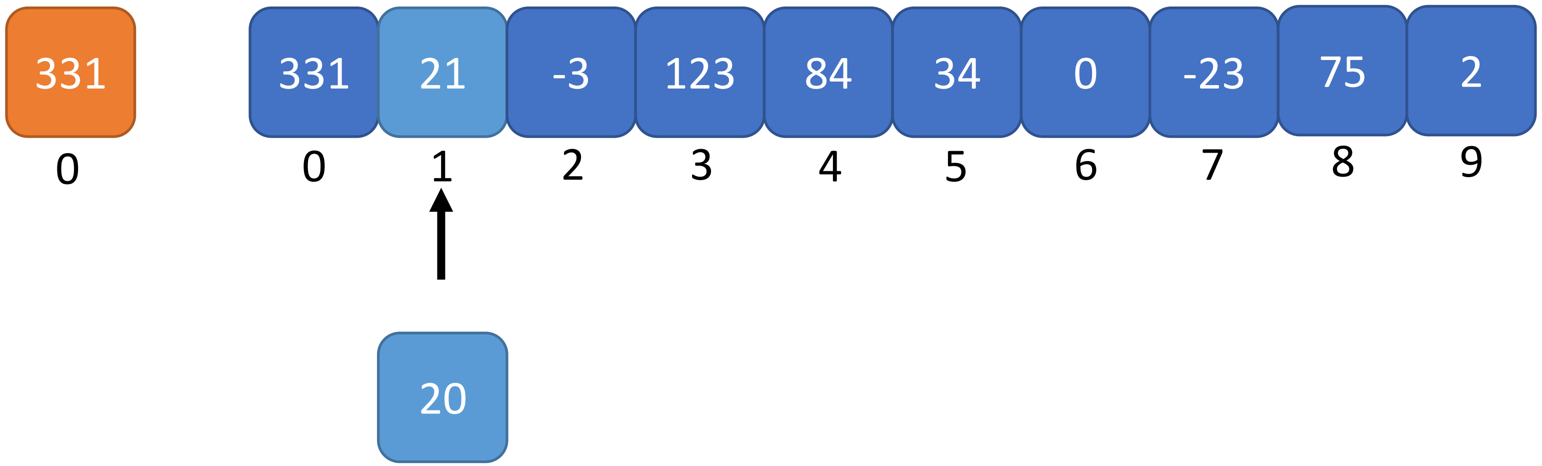
- Check if N > 20 **yes**
- Check if $N < N_{prev}$ **yes**



Linear Searching

Let's find the smallest number larger than 20 in this list

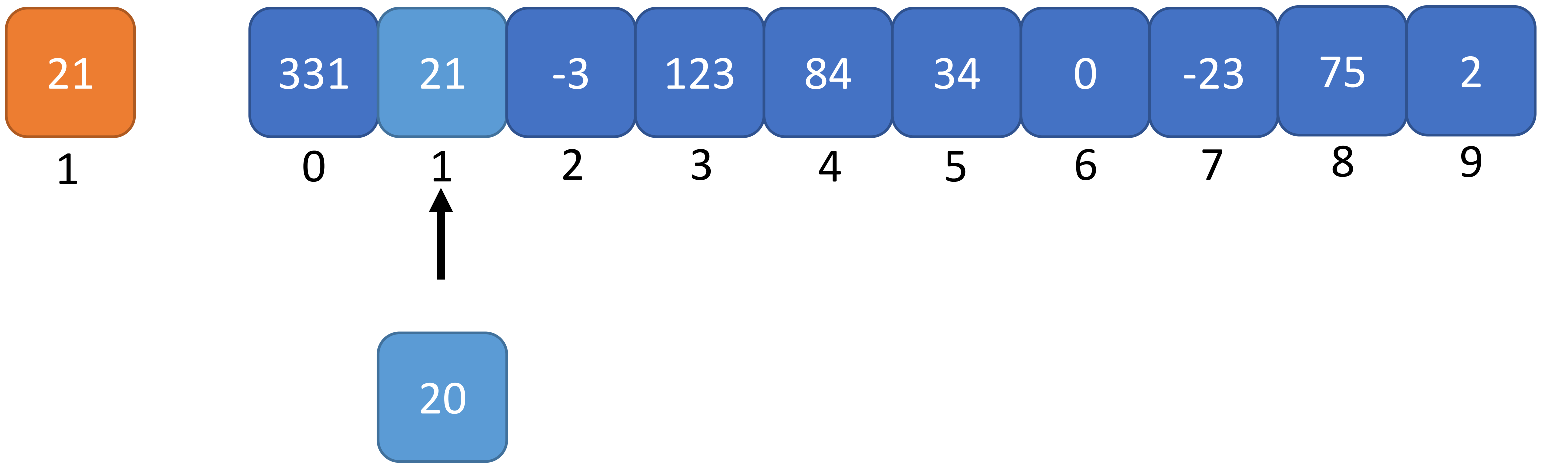
- Check if N > 20 **yes**
- Check if $N < N_{prev}$ **yes**



Linear Searching

Let’s find the smallest number larger than 20 in this list

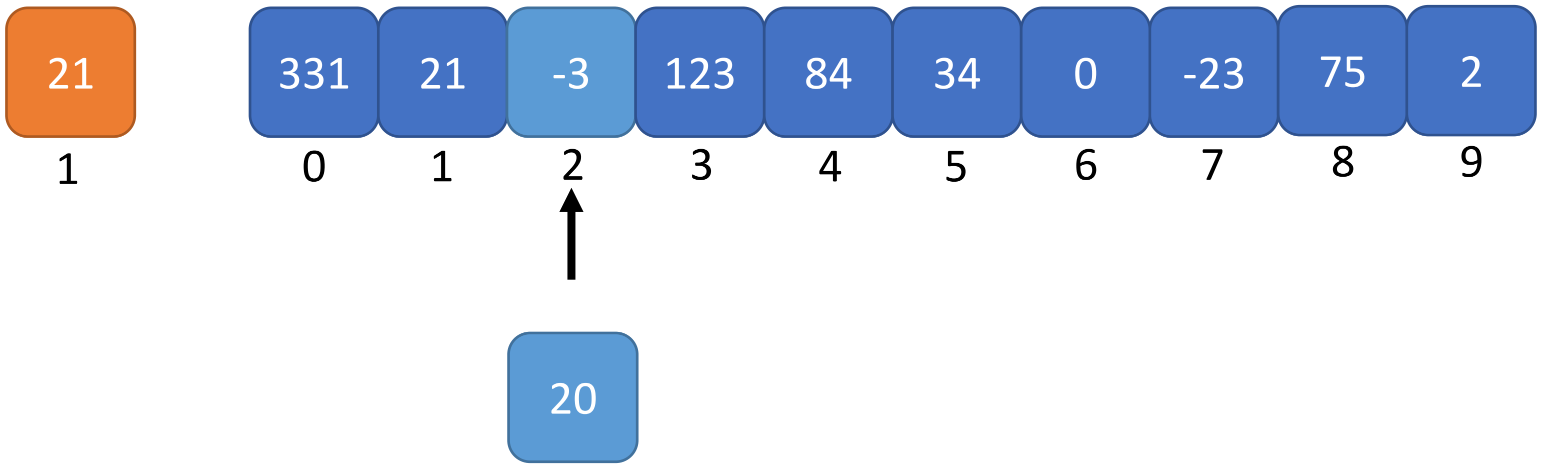
- Check if $N > 20$ **yes**
- Check if $N < N_{prev}$ **yes**



Linear Searching

Let's find the smallest number larger than 20 in this list

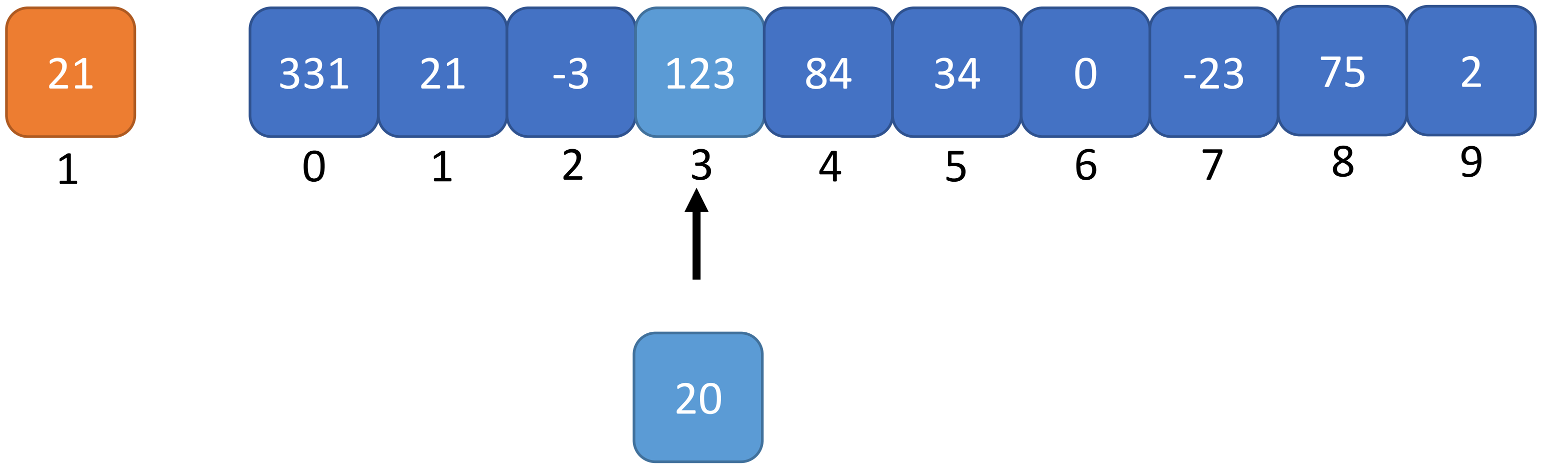
- Check if $N > 20$ **no**
- Check if $N < N_{prev}$ **yes**



Linear Searching

Let’s find the smallest number larger than 20 in this list

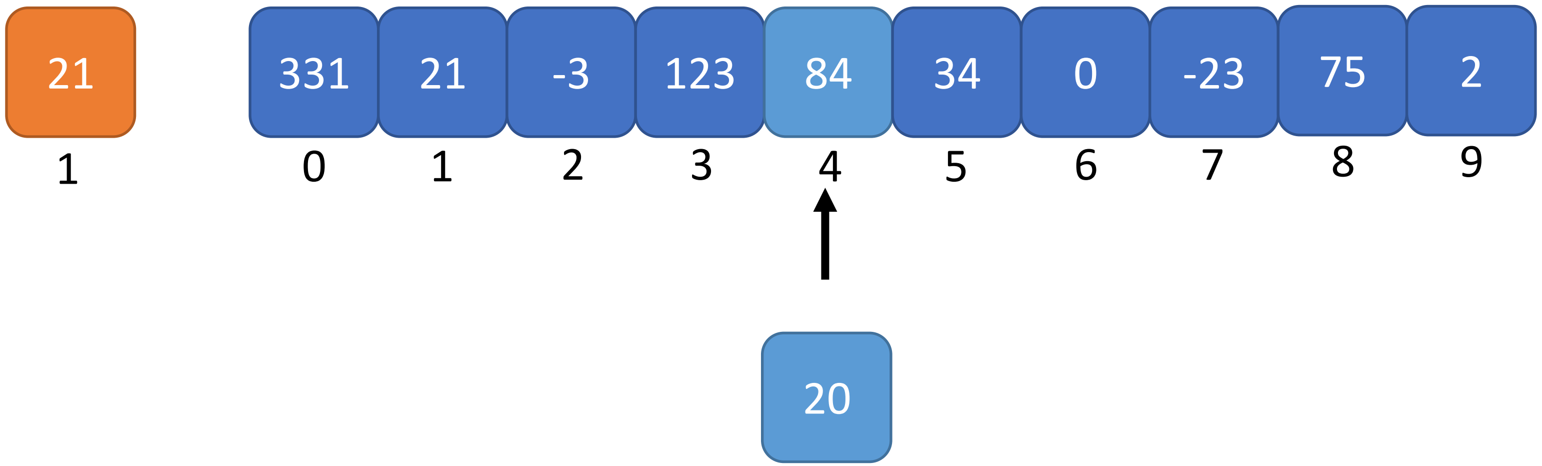
- Check if N > 20 **yes**
- Check if $N < N_{prev}$ **no**



Linear Searching

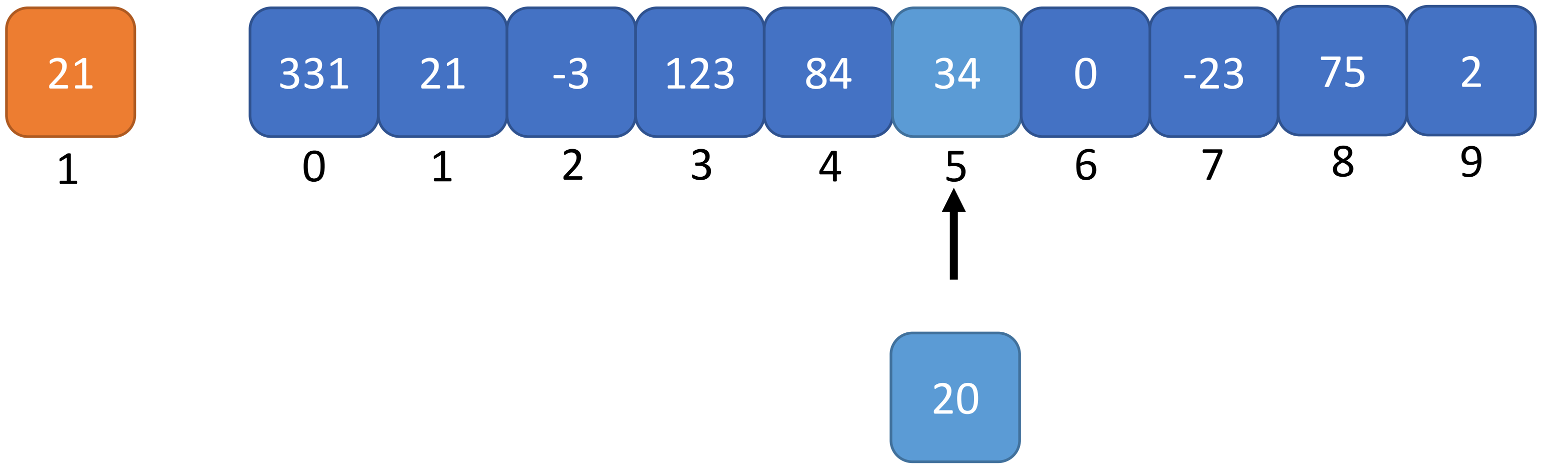
Let’s find the smallest number larger than 20 in this list

- Check if N > 20 **yes**
- Check if $N < N_{prev}$ **no**



Let's find the smallest number larger than 20 in this list

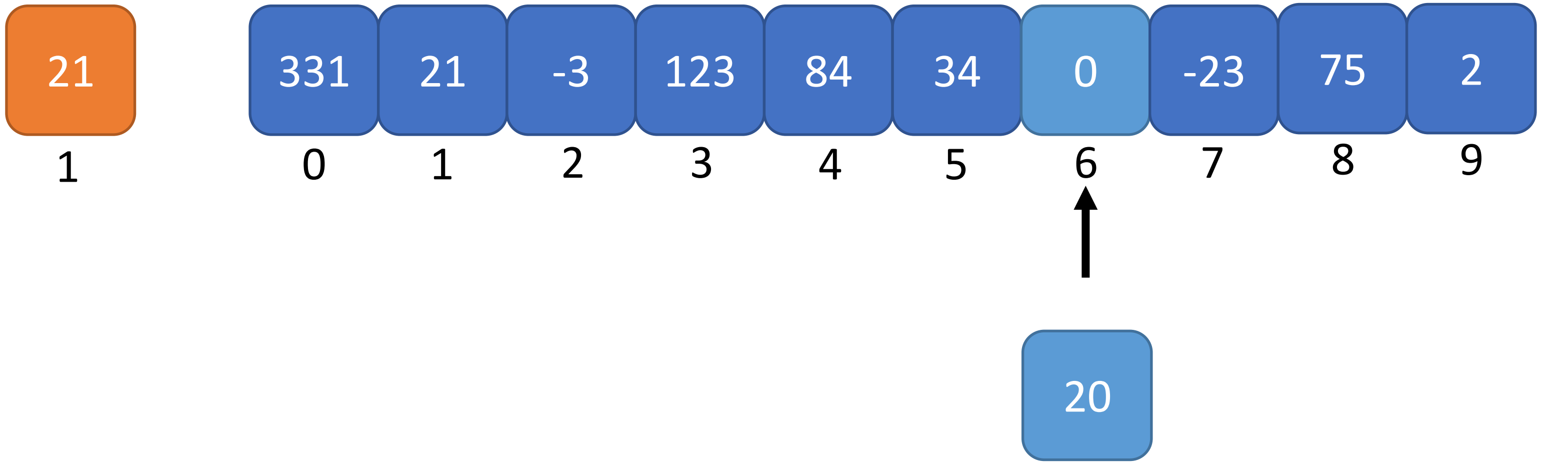
- Check if N > 20 **yes**
- Check if $N < N_{prev}$ **no**



Linear Searching

Let's find the smallest number larger than 20 in this list

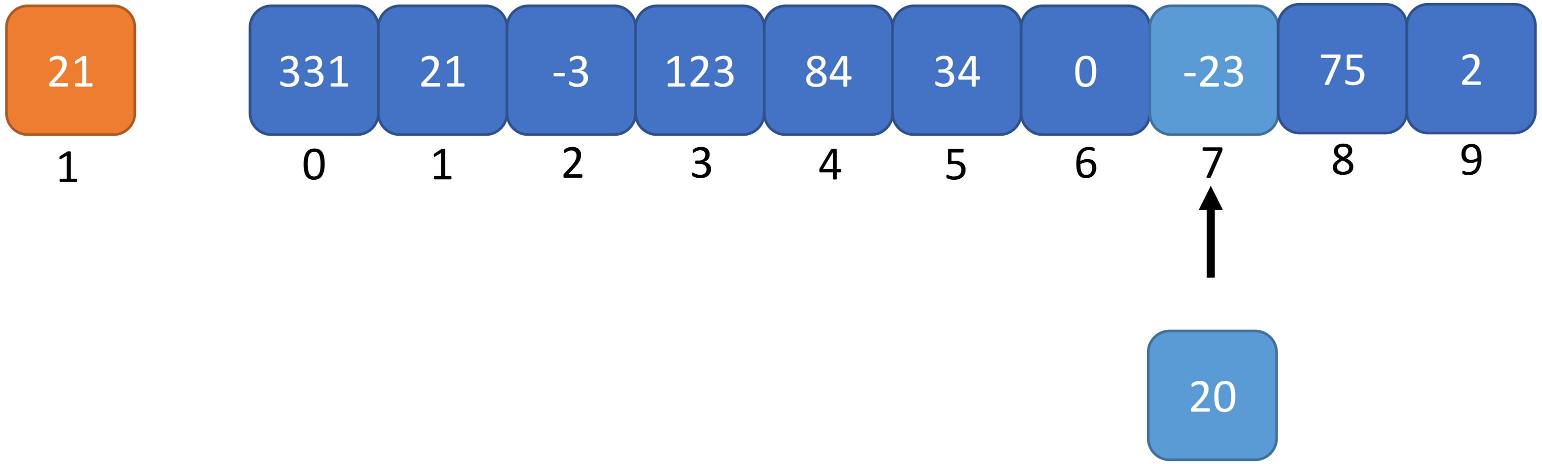
- Check if N > 20 **no**
- Check if $N < N_{prev}$ **yes**



Linear Searching

Let's find the smallest number larger than 20 in this list

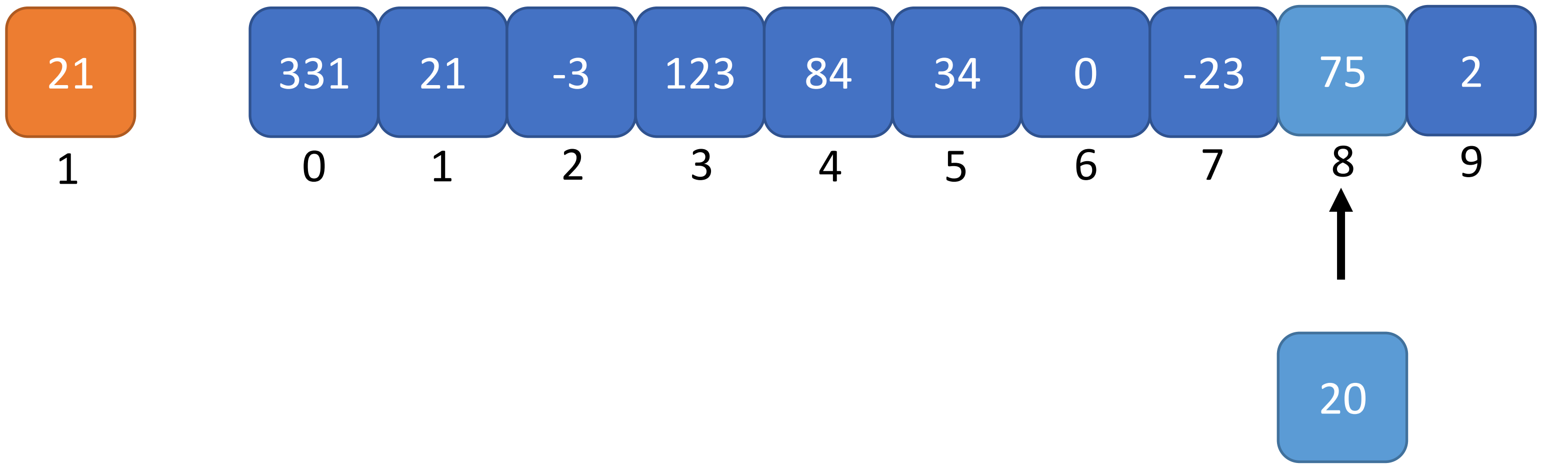
- Check if N > 20 **no**
- Check if $N < N_{prev}$ **yes**



Linear Searching

Let's find the smallest number larger than 20 in this list

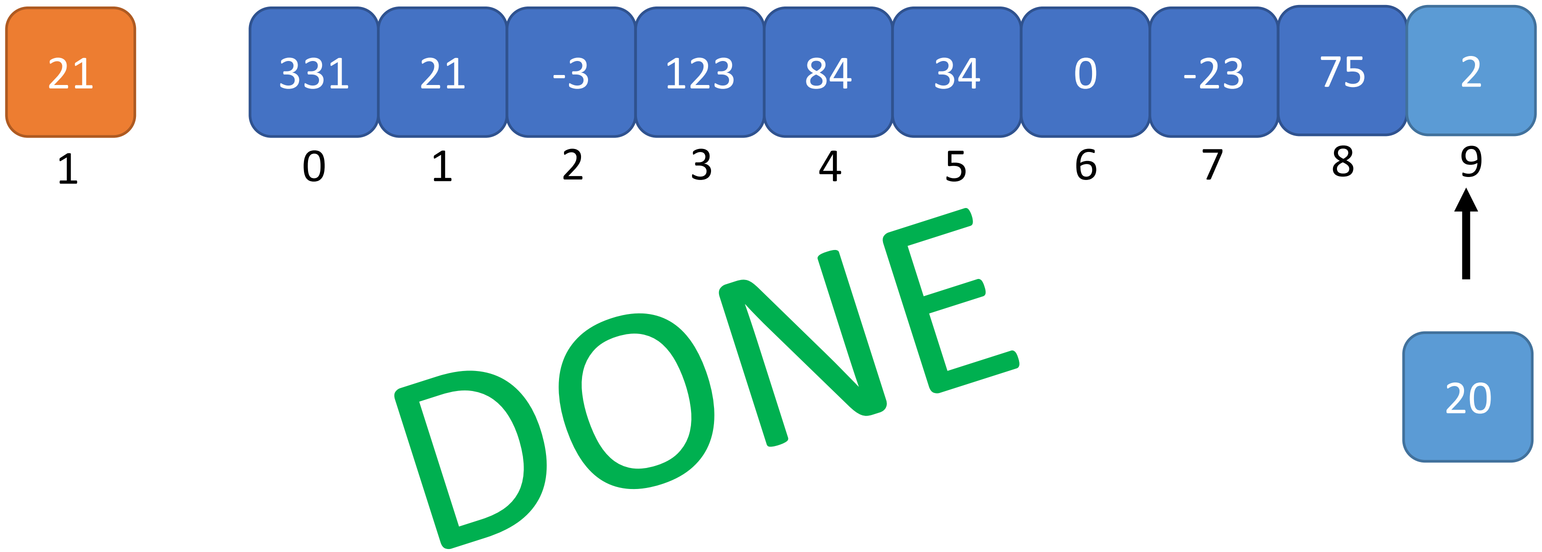
- Check if N > 20 yes
- Check if N < N_{prev} no



Linear Searching

Let's find the smallest number larger than 20 in this list

- Check if $N > 20$ **no**
- Check if $N < N_{prev}$ **yes**



Binary Searching

Requires a sorted binary tree

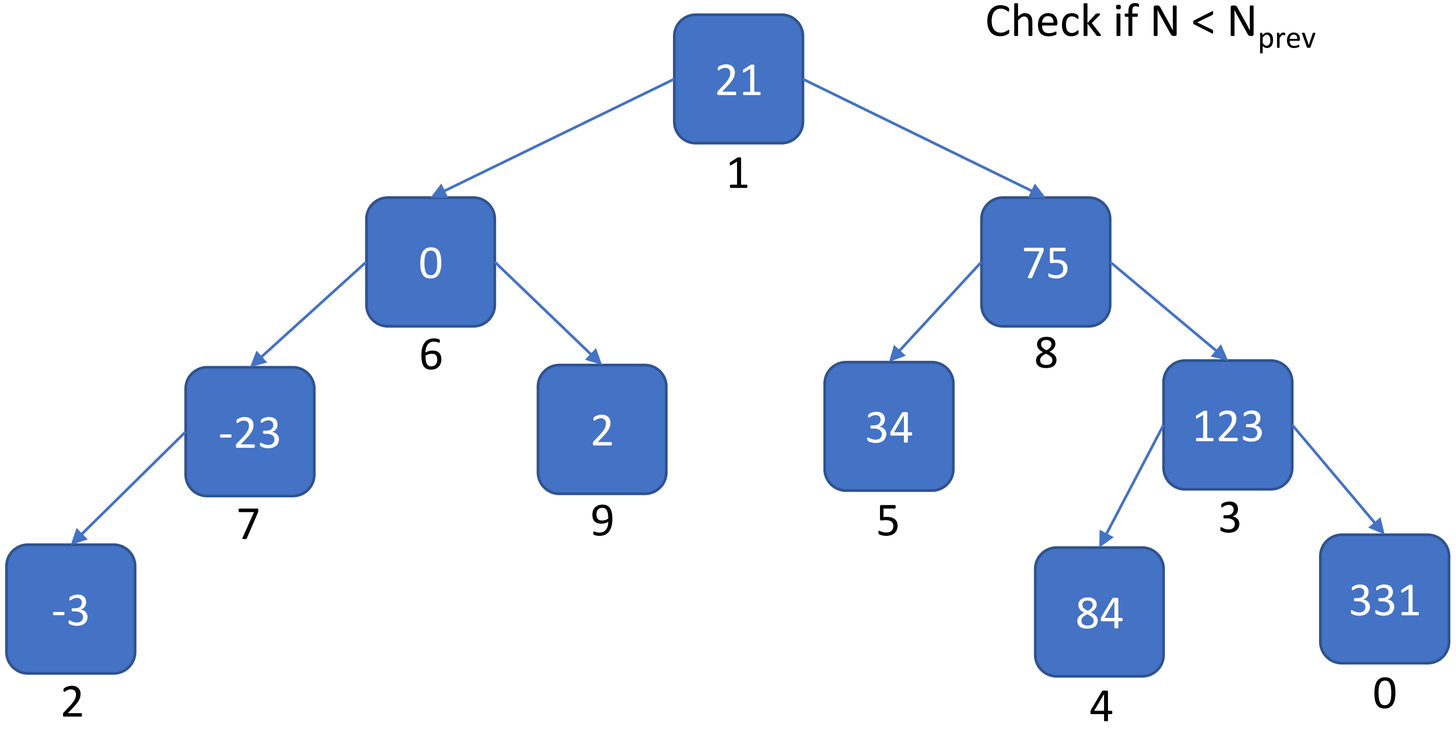
331	21	-3	123	84	34	0	-23	75	2
0	1	2	3	4	5	6	7	8	9

Binary Searching

Requires a sorted binary tree

Check if N > 20

Check if $N < N_{prev}$

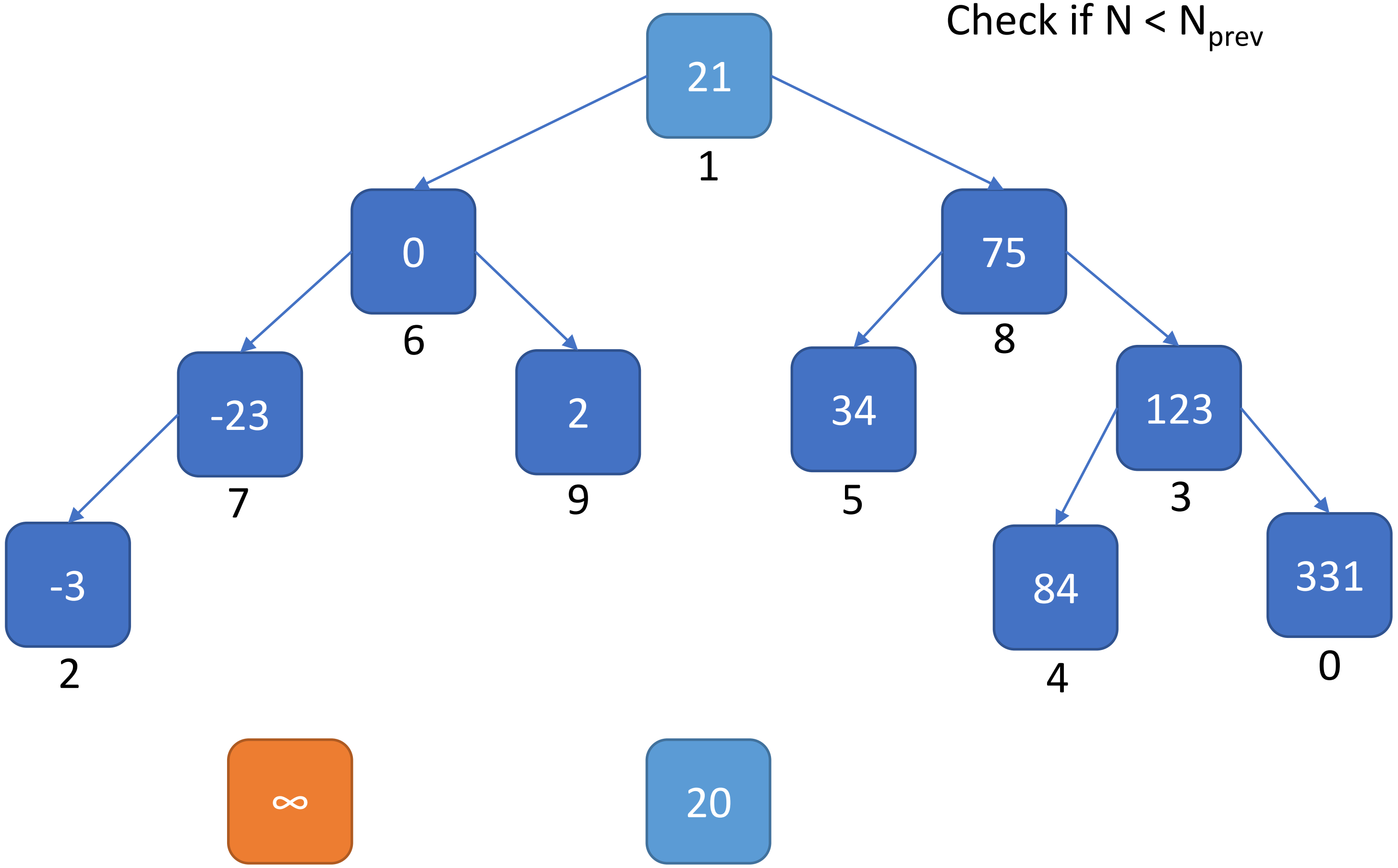


Binary Searching

Requires a sorted binary tree

Check if N > 20 yes

Check if N < N_{prev} yes

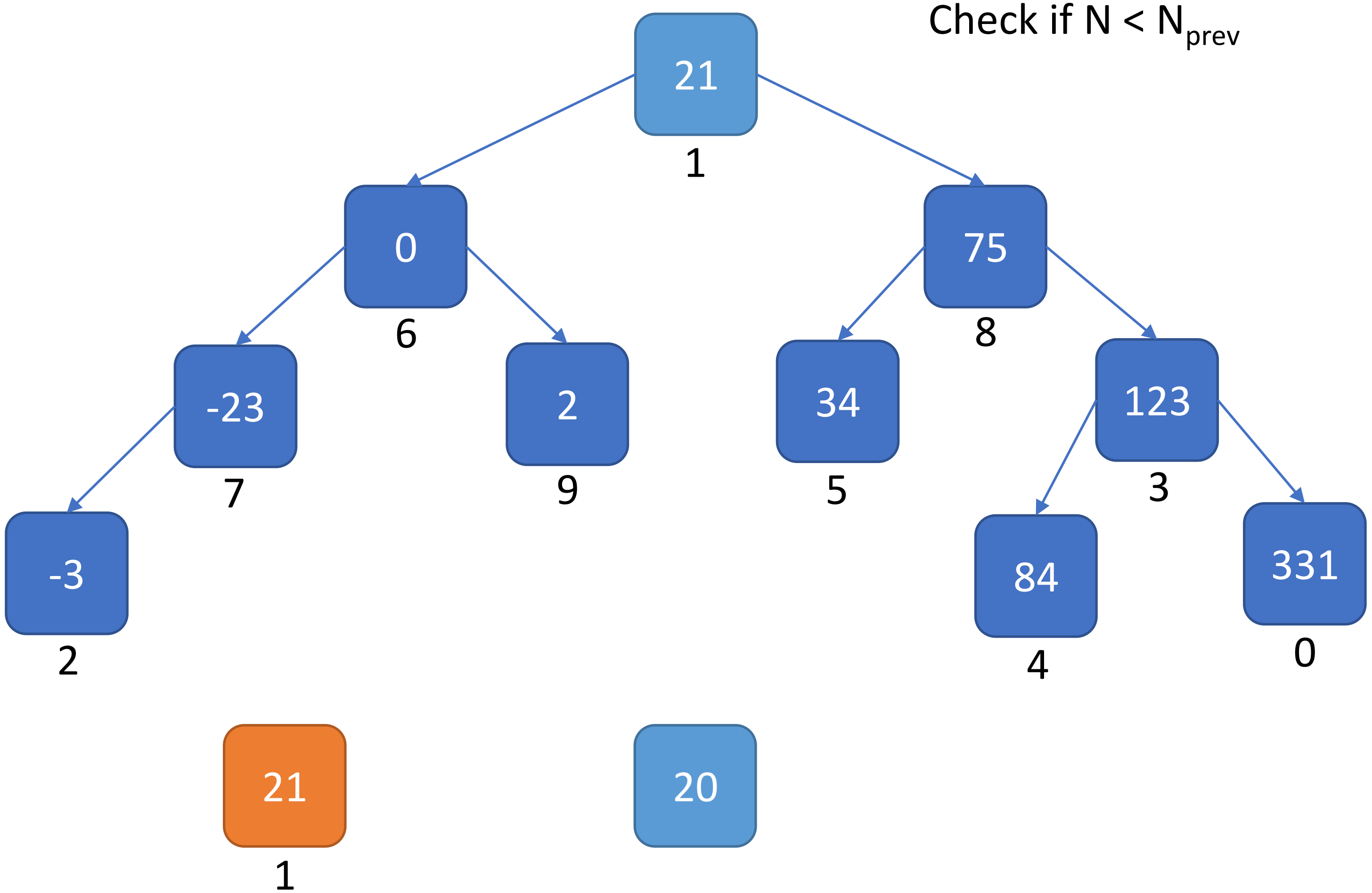


Binary Searching

Requires a sorted binary tree

Check if N > 20 yes

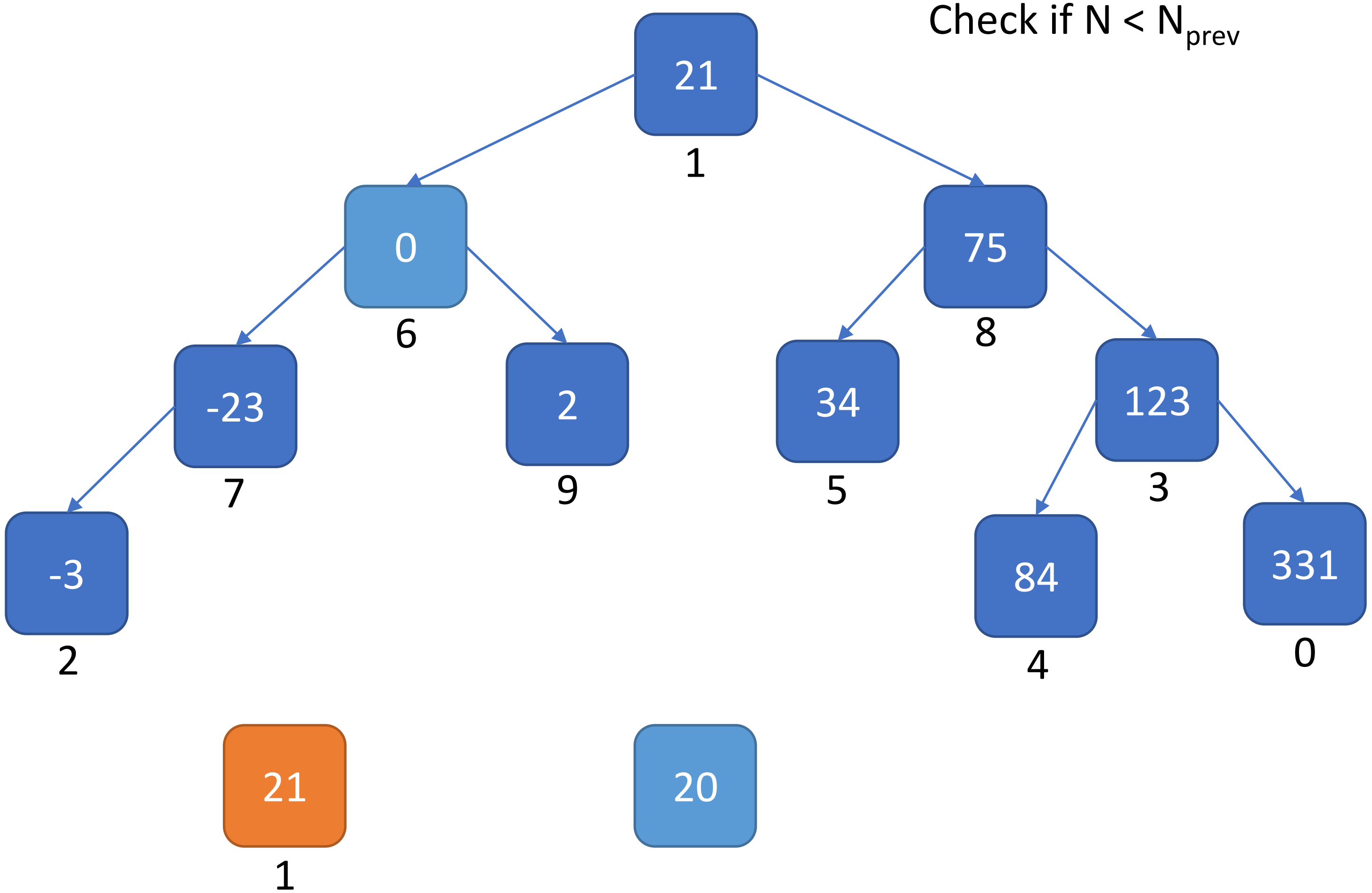
Check if N < N_{prev} yes



Requires a sorted binary tree

Check if N > 20 **no**

Check if N < N_{prev} **yes**

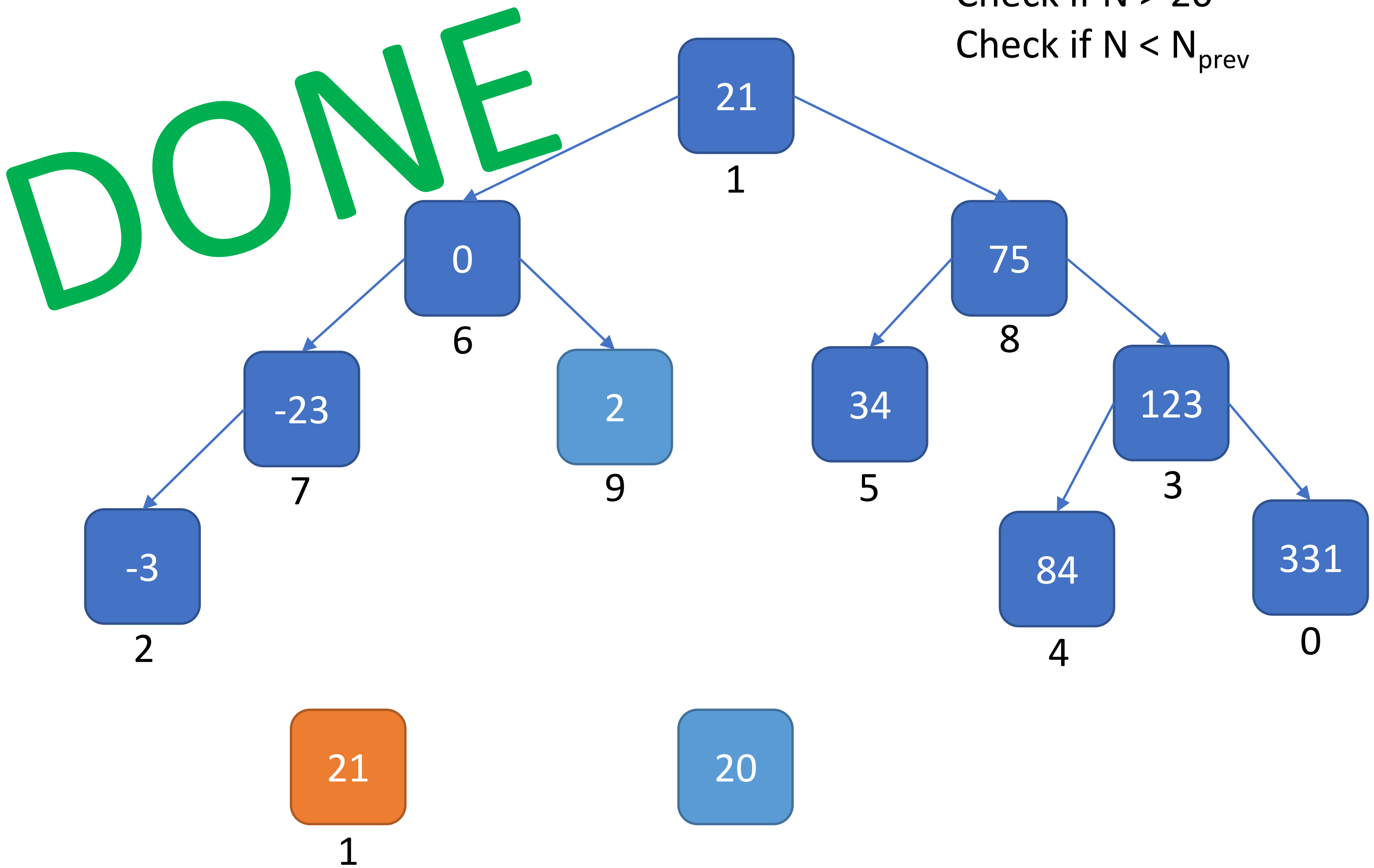


Binary Searching

Requires a sorted binary tree

Check if N > 20 **no**

Check if N < N_{prev} **yes**



Comparing Linear and Binary Search

- Linear search took 20 comparisons to find the smallest number larger than 20, while the binary search only took 6!

- Should we only ever use binary?

- What situations is linear better in?

Practice

- We can write our own linear and binary searching algorithms in Python.

- There are several other searching algorithms besides linear and binary search:
    - Jump search
    - Interpolation search
    - Exponential search
    - Recursive searching methods

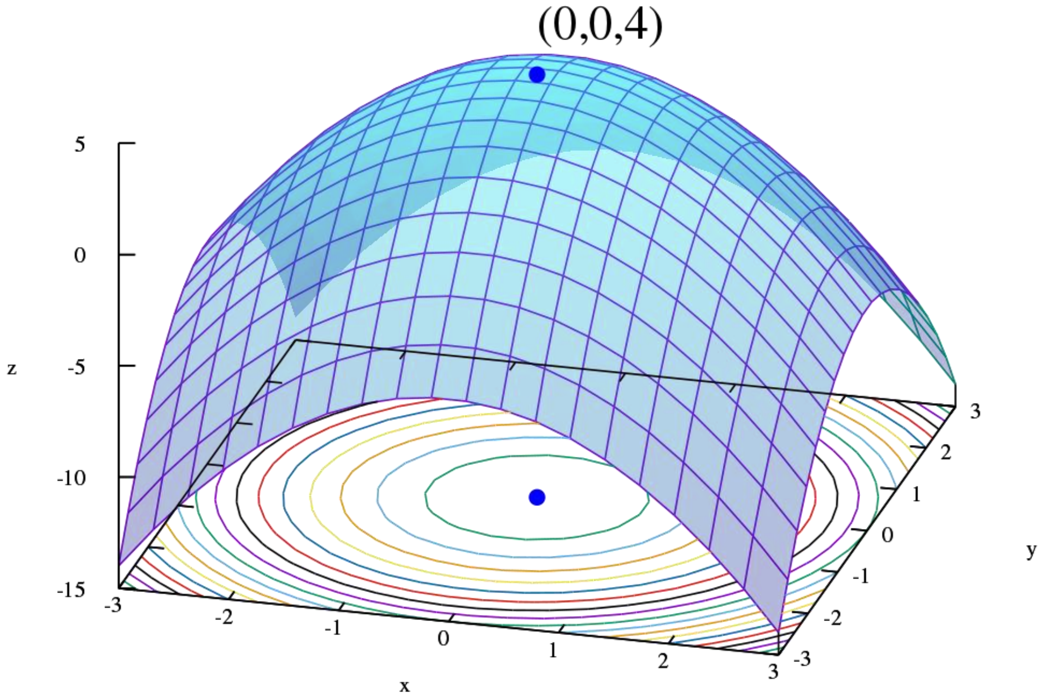
Optimization

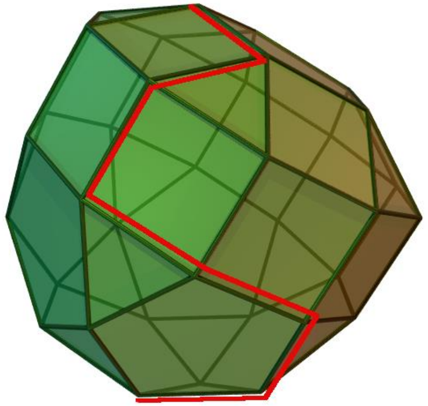
- Optimization allows us to find the best values for a given set of mathematical constraints.

- A common example is curve fitting.

- Useful in computational research, as well as data analysis for experimental work.

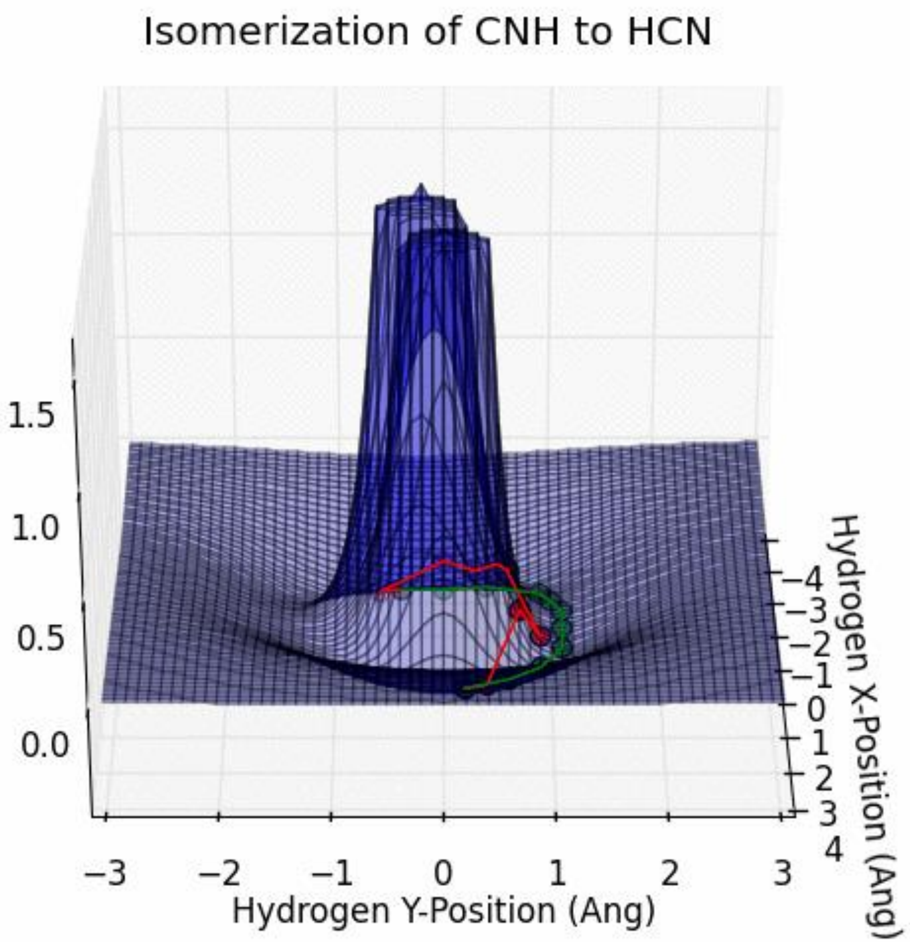
- In Python, there are optimization functions that are typically bundled together in software packages.



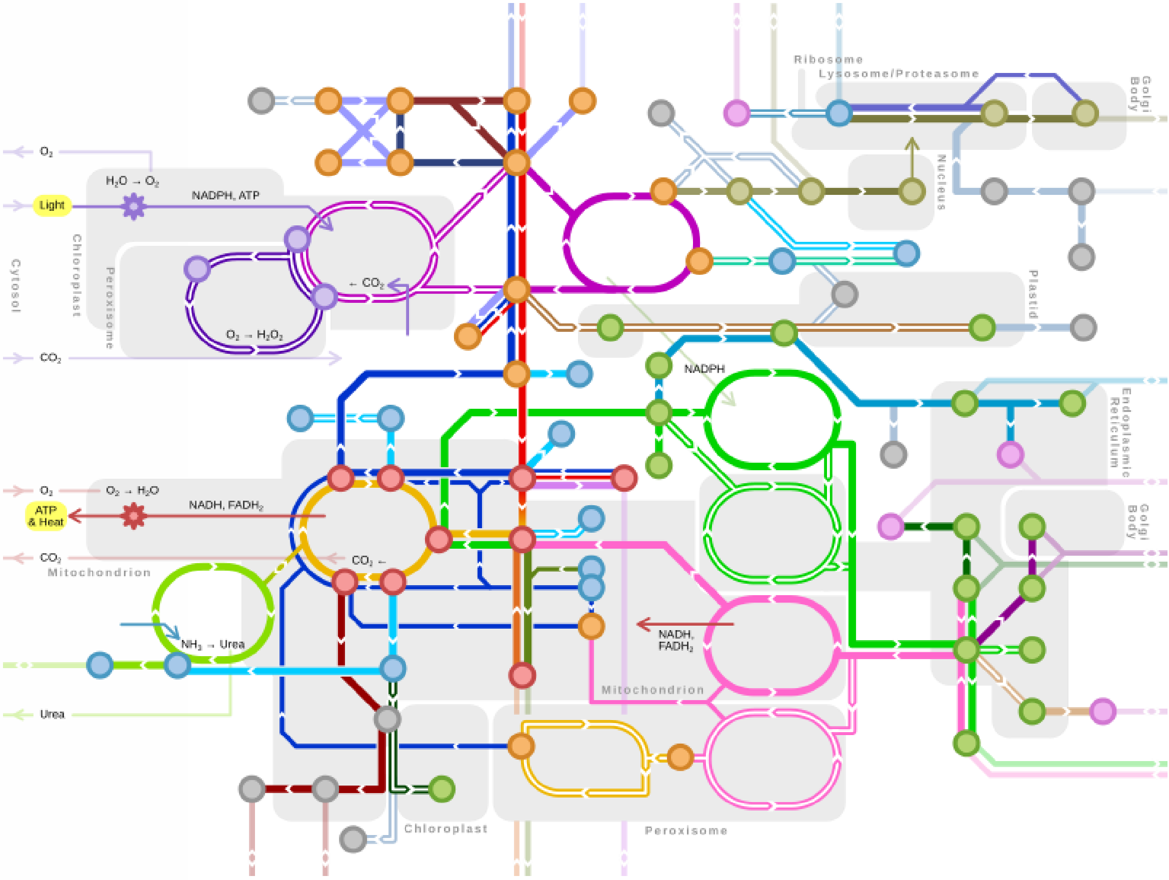


Optimization Examples

Nudged elastic band (NEB):



Optimizing Reaction Fluxes in a Metabolic Network:



Optimization Techniques and Algorithms

- In this class, we'll be focusing on some iterative methods that make use of Hessians and gradients.

- Some examples include:
  - Newton's method
  - Quasi-Newton methods
  - Conjugate gradient
  - Gradient descent/steepest descent

General Formulation for Gradient-based Problems

- We have a function f(x), and we want to find a value x such that f(x) is minimized.

- Typically, f(x) is (twice) differentiable, convex, and exists within the Euclidean space $\mathbb{R}^n$, where n is the dimension. There may also be a set of constraints that we have to adhere to.

- In an iterative optimization method, we guess an initial value x_0 and the update our value using the gradient in some way until we converge at the minimum:

$$x_{k+1} = x_k + s_k \nabla f(x_k)$$

Example of a Quasi-Newton Method

JOHNS HOPKINS WHITING SCHOOL of ENGINEERING

1. Choose a starting point, x_0
2. Calculate the search by approximating the inverse Hessian, H^{-1}
3. Calculate the change in x by the following expression:

$$x_{k+1} = x_k - [H^{-1}]_k \nabla f(x_k)$$

4. Determine x_{k+1} from the expression above.
5. Check if the method has converged – we see if the gradient is equal to 0.
6. Repeat from step 2 until we have converged.

Other Gradient-Based Methods

- There are other methods where there are different ways to approximate the inverse Hessian (or not use it at all).

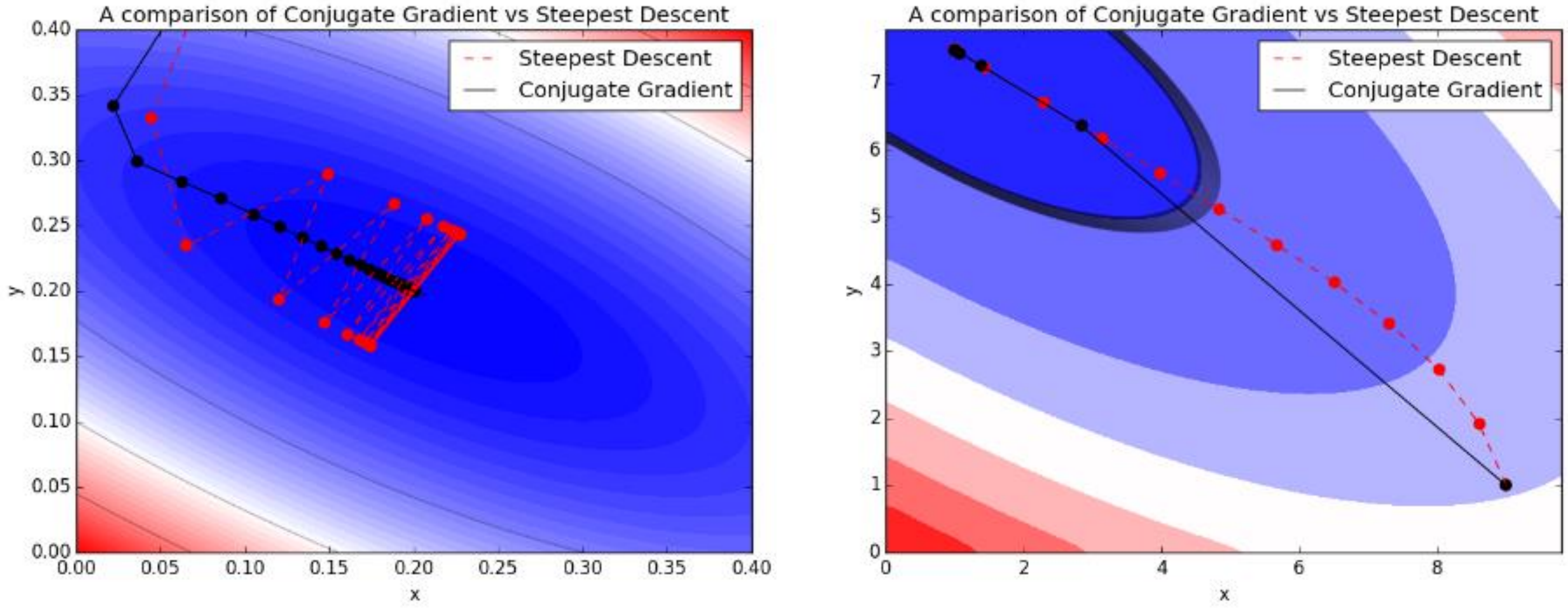
- Gradient descent/steepest descent: $\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n),\ n \geq 0.$

- Broyden-Fletcher-Goldfarb-Shanno (BFGS):

$$H_{k+1} = H_k + \frac{yy^T}{y^T s} - \frac{H_k s s^T H_k}{s^T H_k s}$$

$$s = x_{k+1} - x_k, \qquad y = \nabla f(x_{k+1}) - \nabla f(x_k)$$

Conjugate Gradient



Conjugate Gradient: $|s_i\rangle = -|g_i\rangle + \beta|s_{i-1}\rangle$

$$\beta = \frac{\langle g_i|g_i\rangle}{\langle g_{i-1}|g_{i-1}\rangle}$$

Fletcher-Reeves Method

SciPy's Optimize Function

JOHNS HOPKINS
WHITING SCHOOL of ENGINEERING

- SciPy has an optimize function that has many different solving algorithms built in.

scipy.optimize.minimize(*fun, x0, args=(), method=None, jac=None, hess=None, hessp=None, bounds=None, constraints=(), tol=None, callback=None, options=None*) [source]

Minimization of scalar function of one or more variables.

Parameters:

fun : *callable*

The objective function to be minimized.

```
fun(x, *args) -> float
```

where x is an 1-D array with shape (n,) and *args* is a tuple of the fixed parameters needed to completely specify the function.

x0 : *ndarray, shape (n,)*

Initial guess. Array of real elements of size (n,), where 'n' is the number of independent variables.