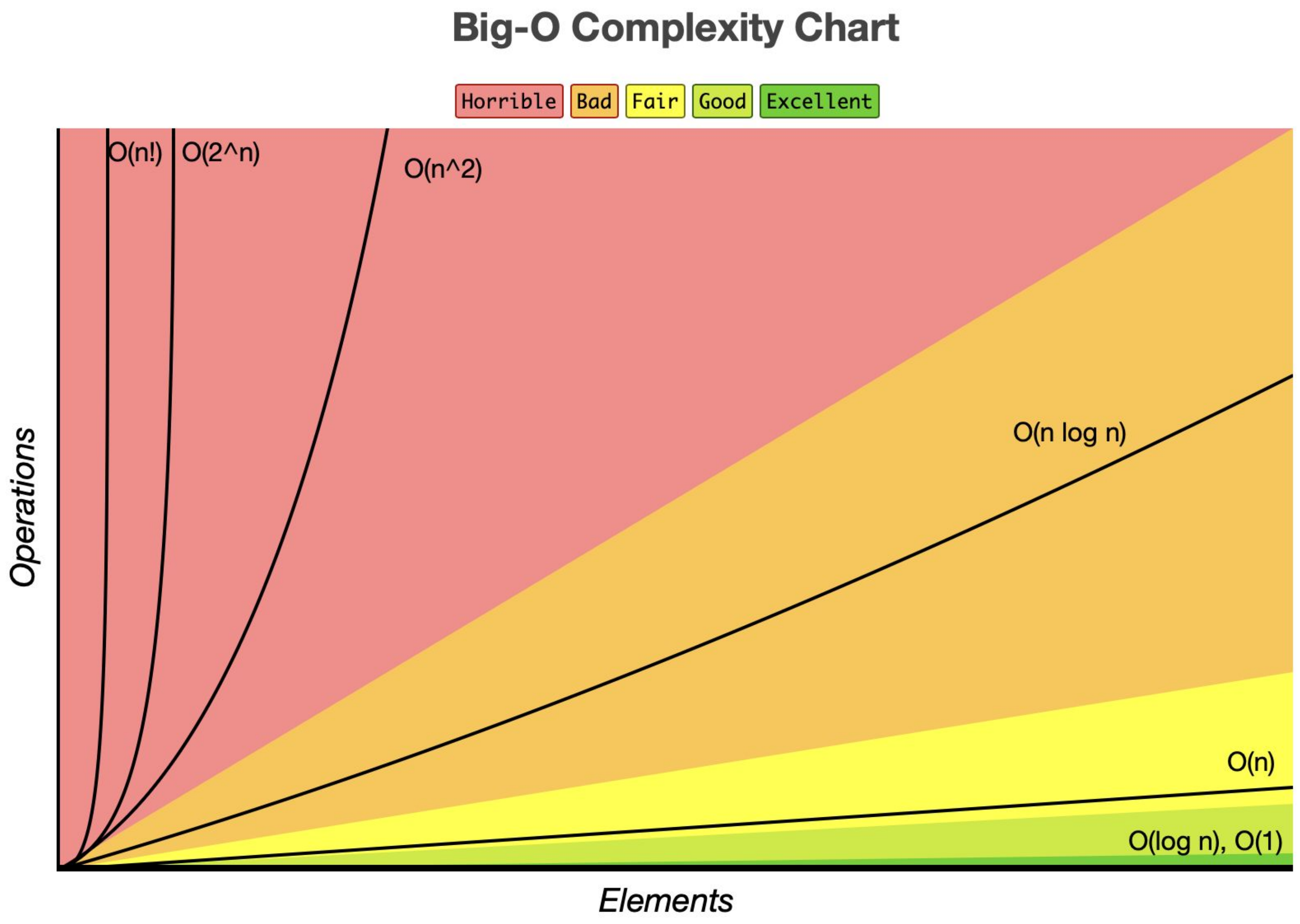
Lecture

Asymptotic Notation & Bubble Sort

Lecture

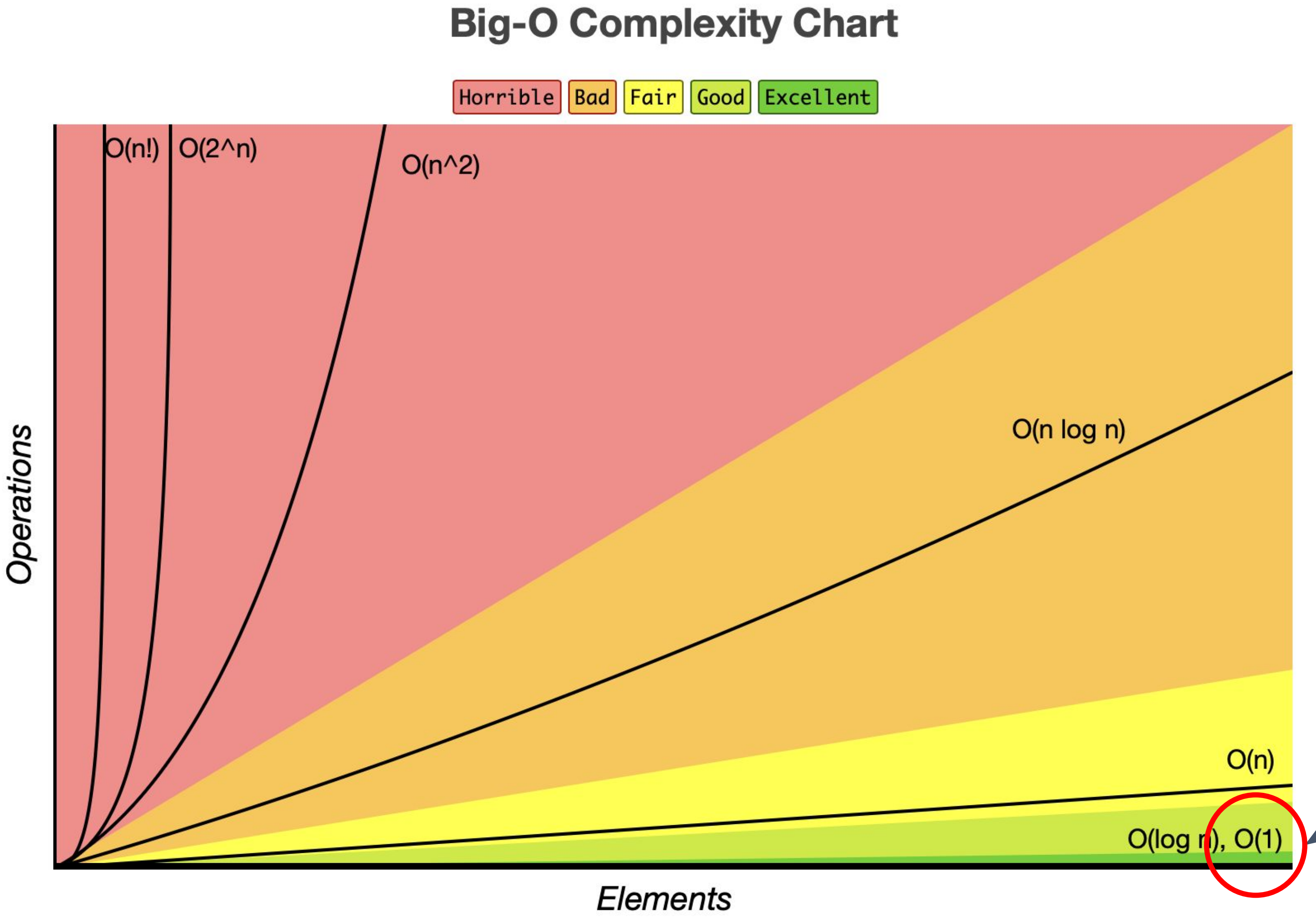
- ❖ Asymptotic Notation
- ❖ Bubble Sort
- ❖ Q/A project

Asymptotic Notation



Check this out: https://www.bigocheatsheet.com/

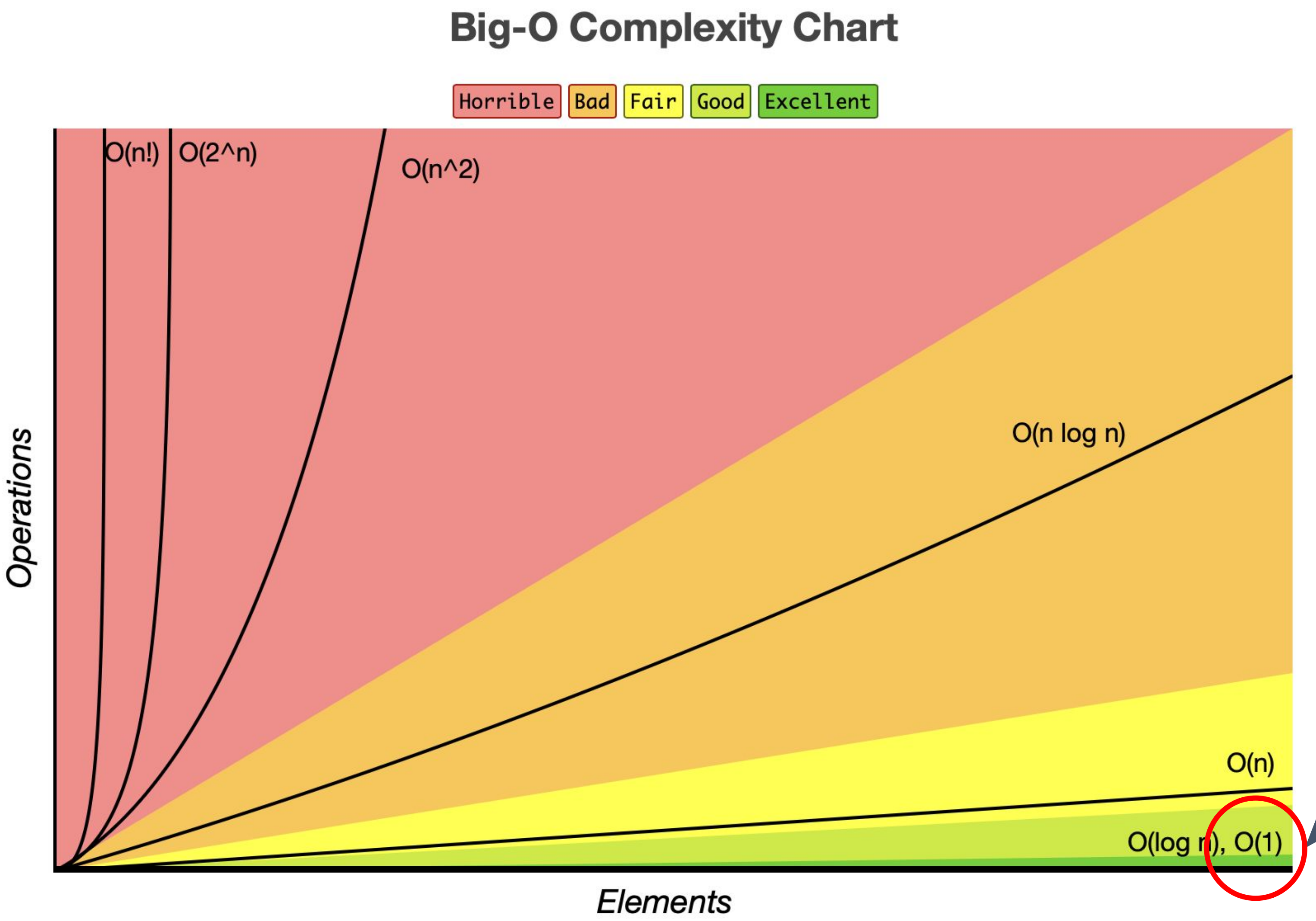
Asymptotic Notation



$O(1)$: An algorithm is said to have a constant execution time when it is not dependent on the input data (*n*)

Check this out: https://www.bigocheatsheet.com/

Asymptotic Notation

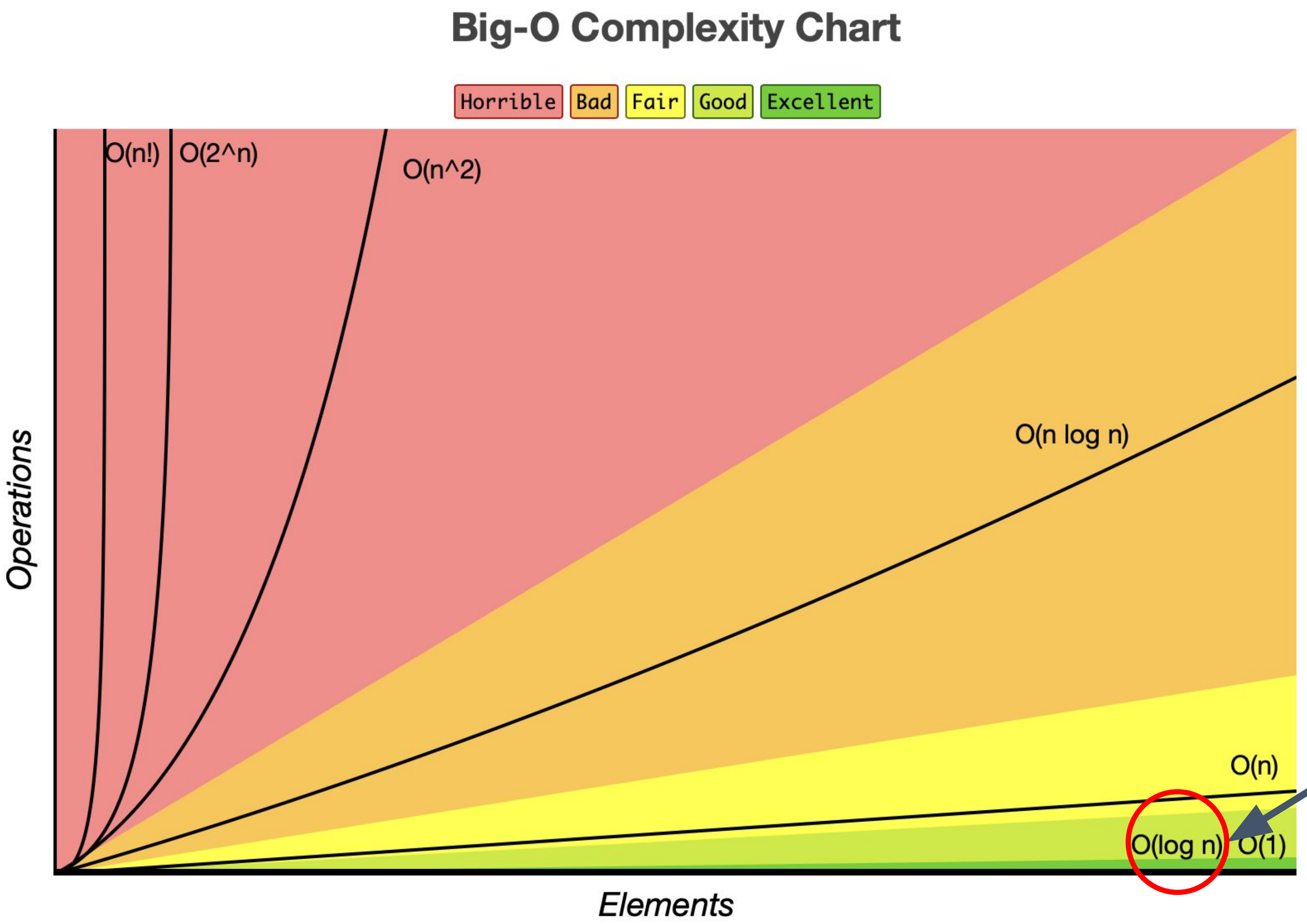


O(1) example:

```
def get_first(data):
    return data[0]
```

Check this out: https://www.bigocheatsheet.com/

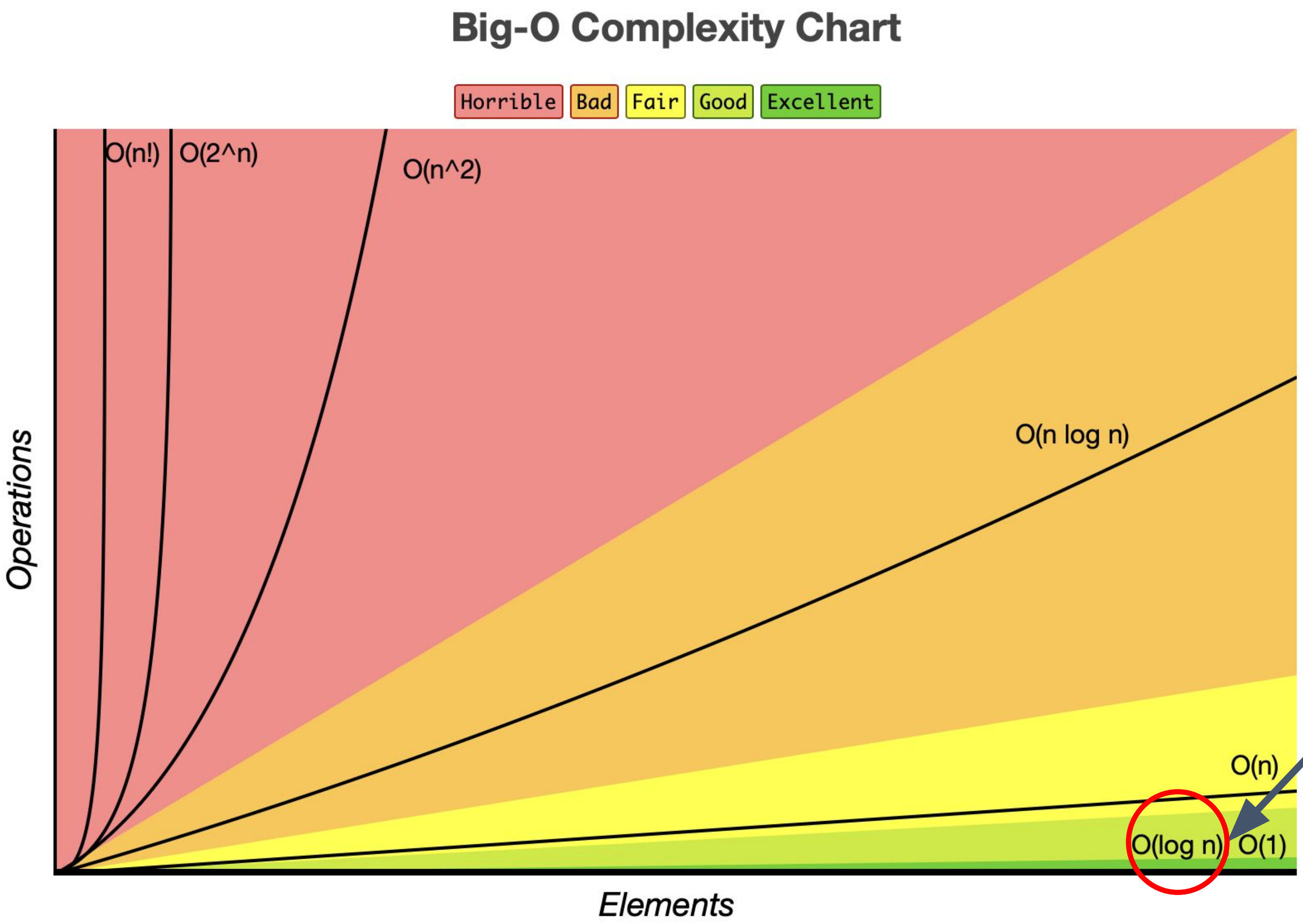
Asymptotic Notation



$O(log\ n)$: An algorithm is said to have a logarithmic time complexity when it reduces the size of the input data in each step

Check this out: https://www.bigocheatsheet.com/

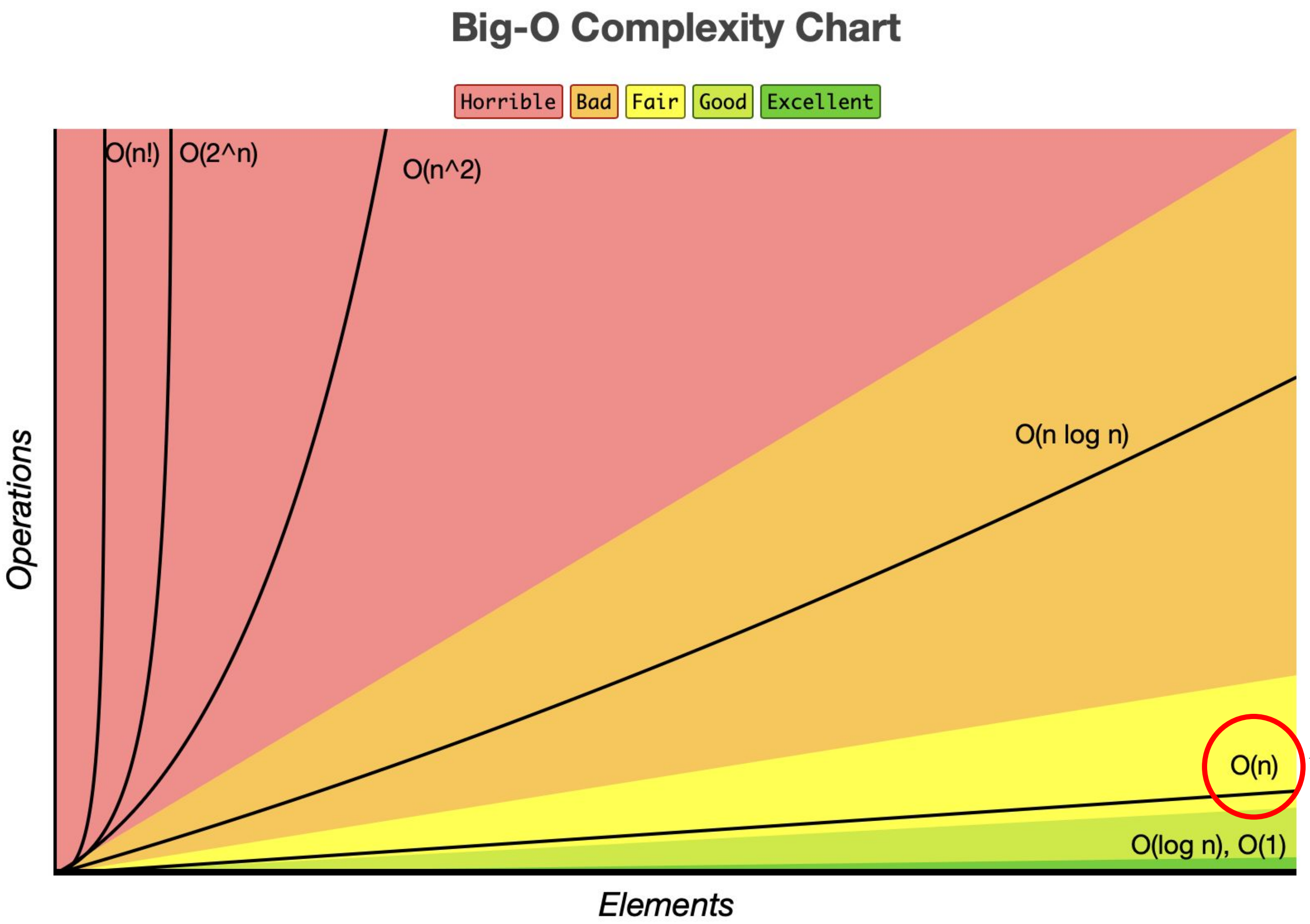
Asymptotic Notation



O(log n) example:
binary search

Check this out: https://www.bigocheatsheet.com/

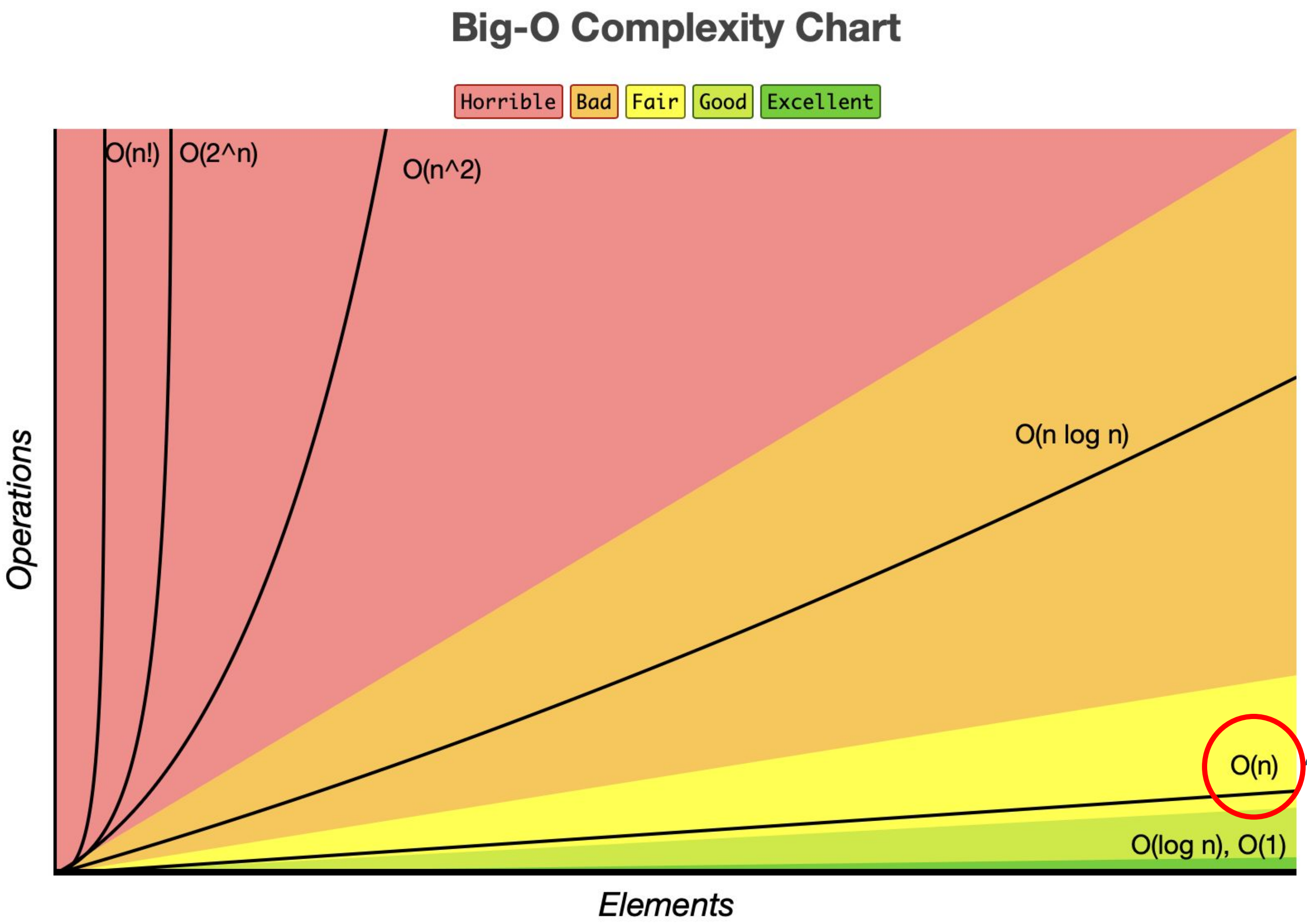
Asymptotic Notation



$O(n)$: An algorithm is said to have a linear time complexity when the running time increases at most linearly with the size of the input data.

Check this out: https://www.bigocheatsheet.com/

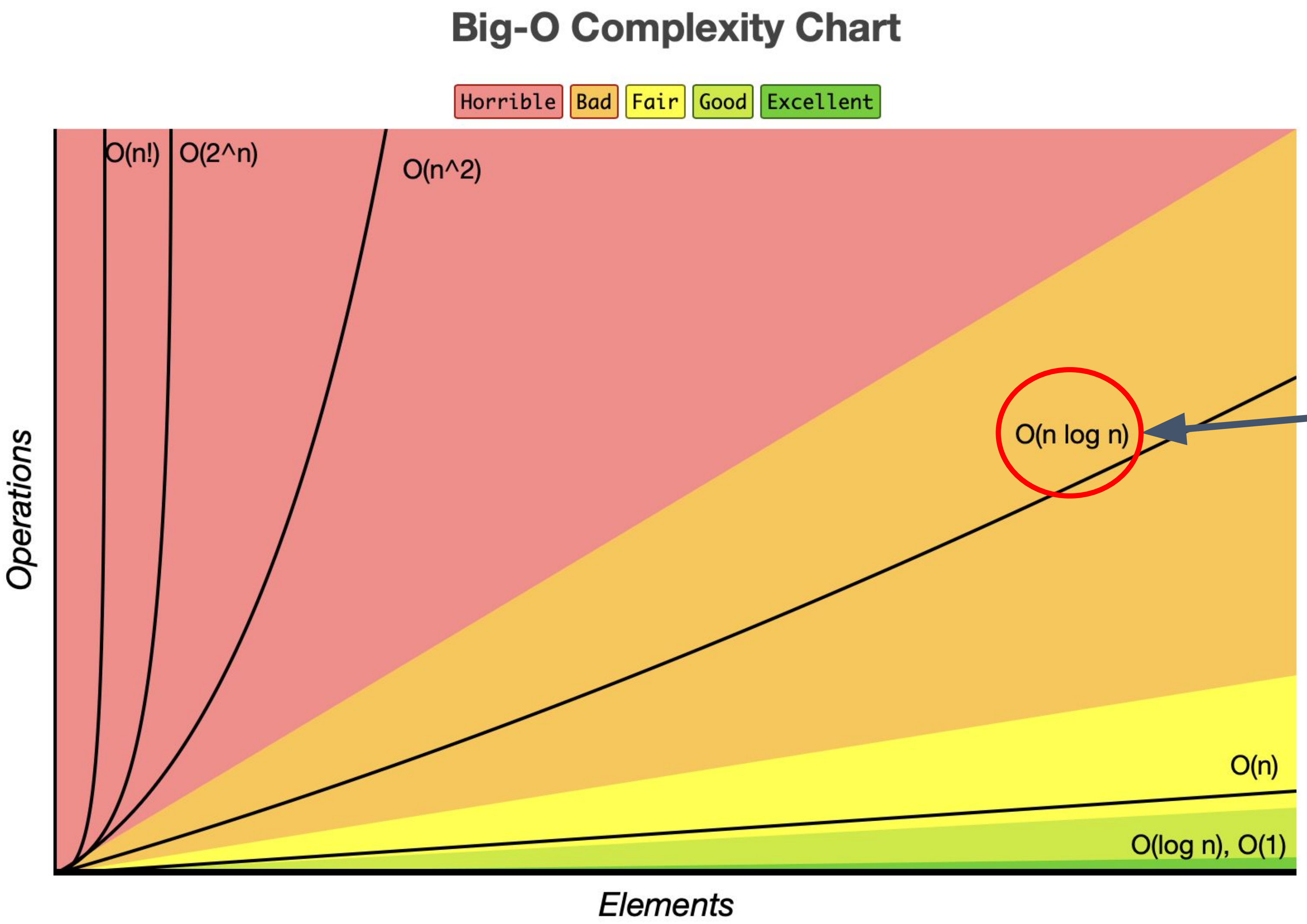
Asymptotic Notation



$O(n)$ example:
linear search in a list

Check this out: https://www.bigocheatsheet.com/

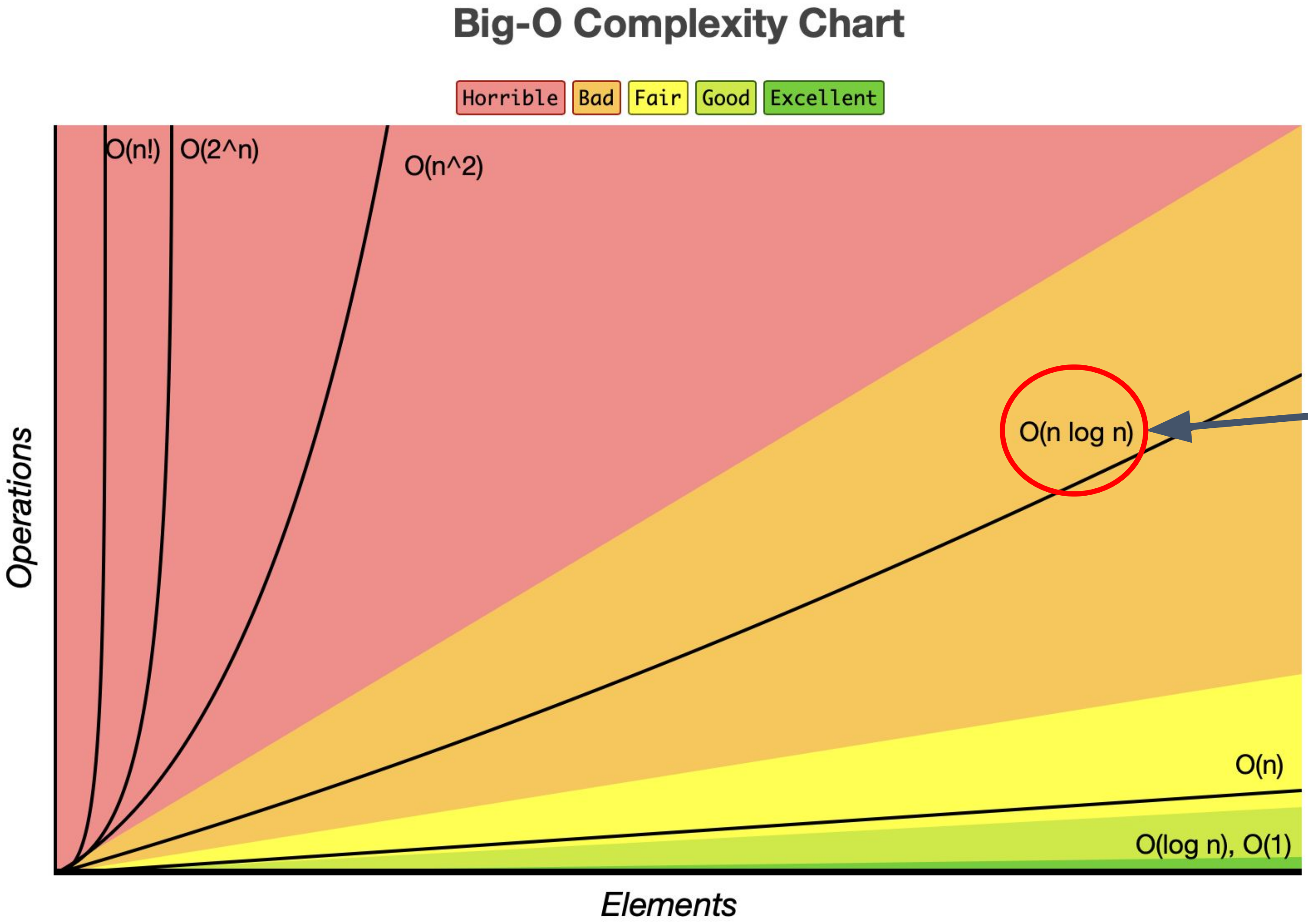
Asymptotic Notation



O(n log n):
An algorithm is said to have a quasilinear time complexity when each operation in the input data have a logarithm time complexity.

Check this out: https://www.bigocheatsheet.com/

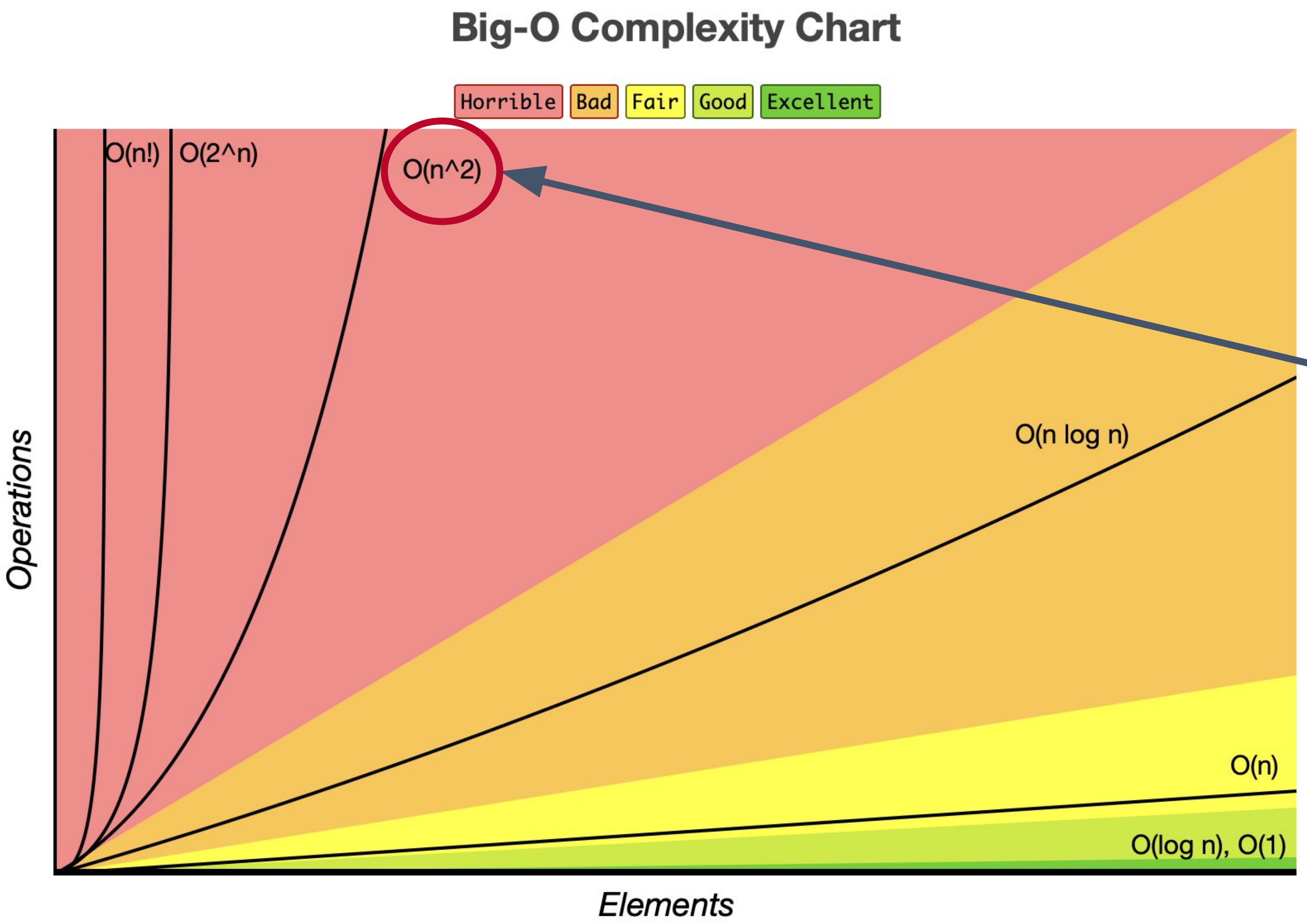
Asymptotic Notation



O(n log n) example: merge sort or linear algorithm + binary search for each element

Check this out: https://www.bigocheatsheet.com/

Asymptotic Notation

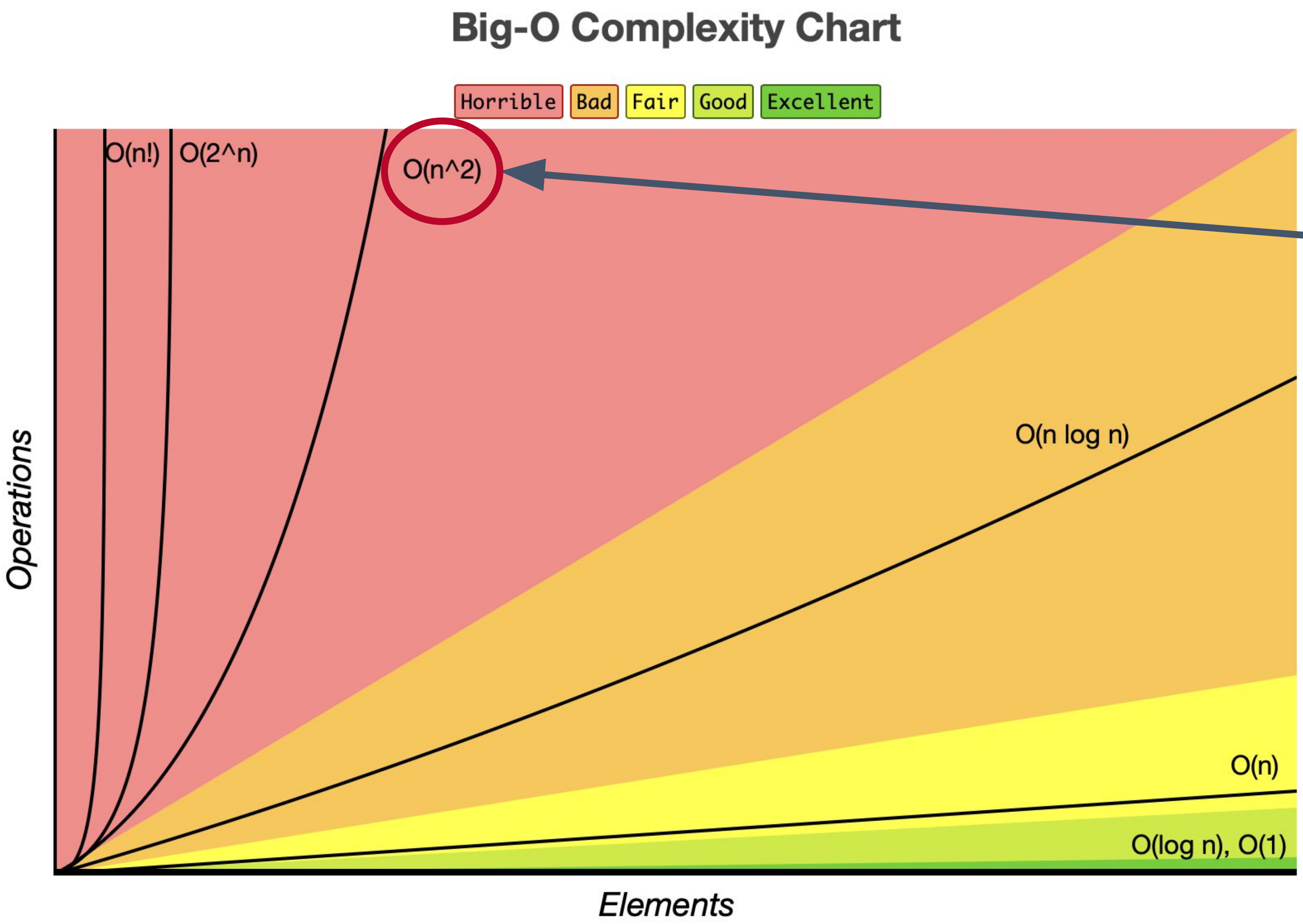


$O(n^2)$:

An algorithm is said to have a quadratic time complexity when it needs to perform a linear time operation for each value in the input data

Check this out: https://www.bigocheatsheet.com/

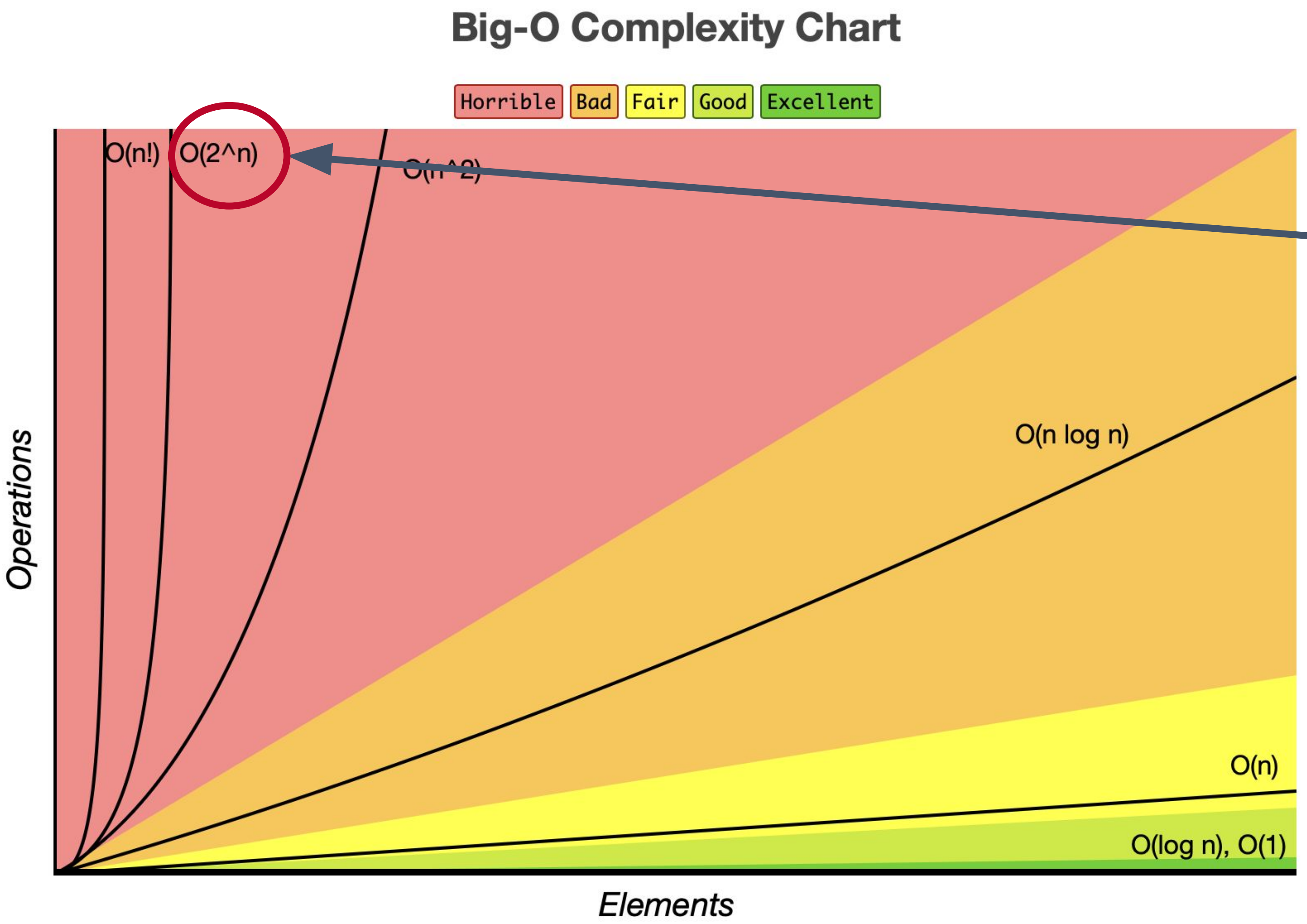
Asymptotic Notation



$O(n^2)$ example:
explore a square matrix

Check this out: https://www.bigocheatsheet.com/

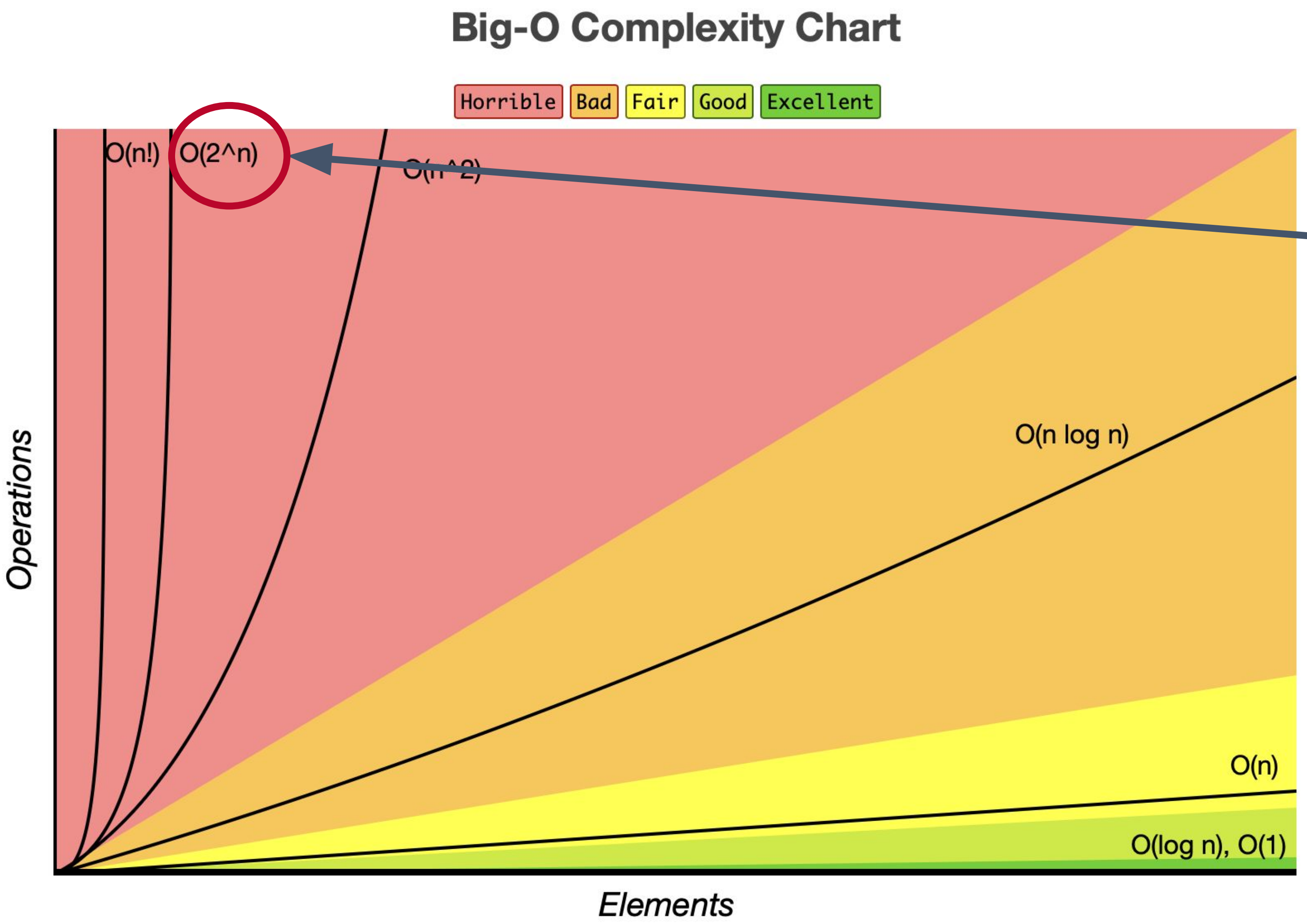
Asymptotic Notation



$O(2^n)$:
An algorithm is said to have an exponential time complexity when the growth doubles with each addition to the input data set.

Check this out: https://www.bigocheatsheet.com/

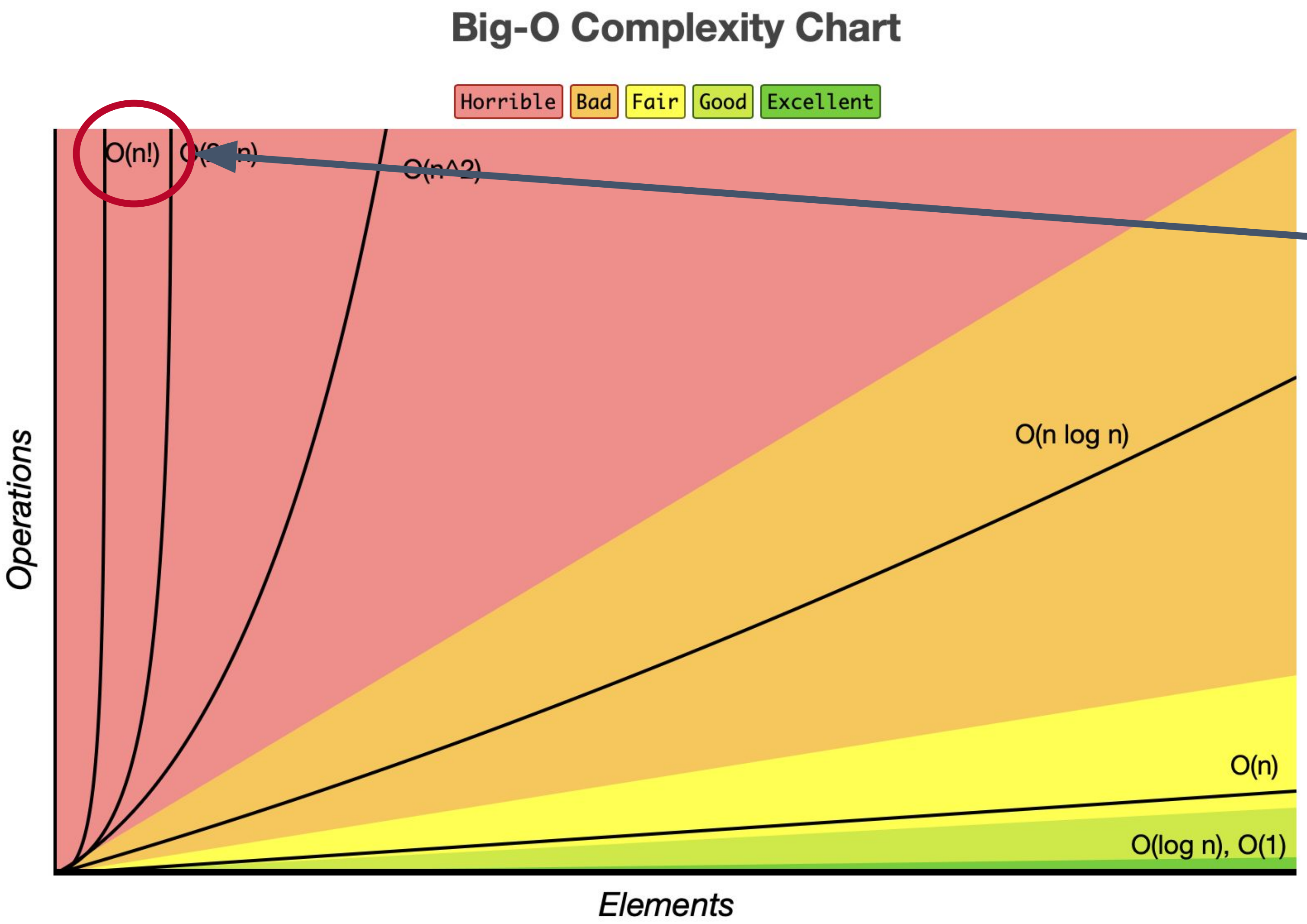
Asymptotic Notation



$O(2^n)$ example: fibonacci

Check this out: https://www.bigocheatsheet.com/

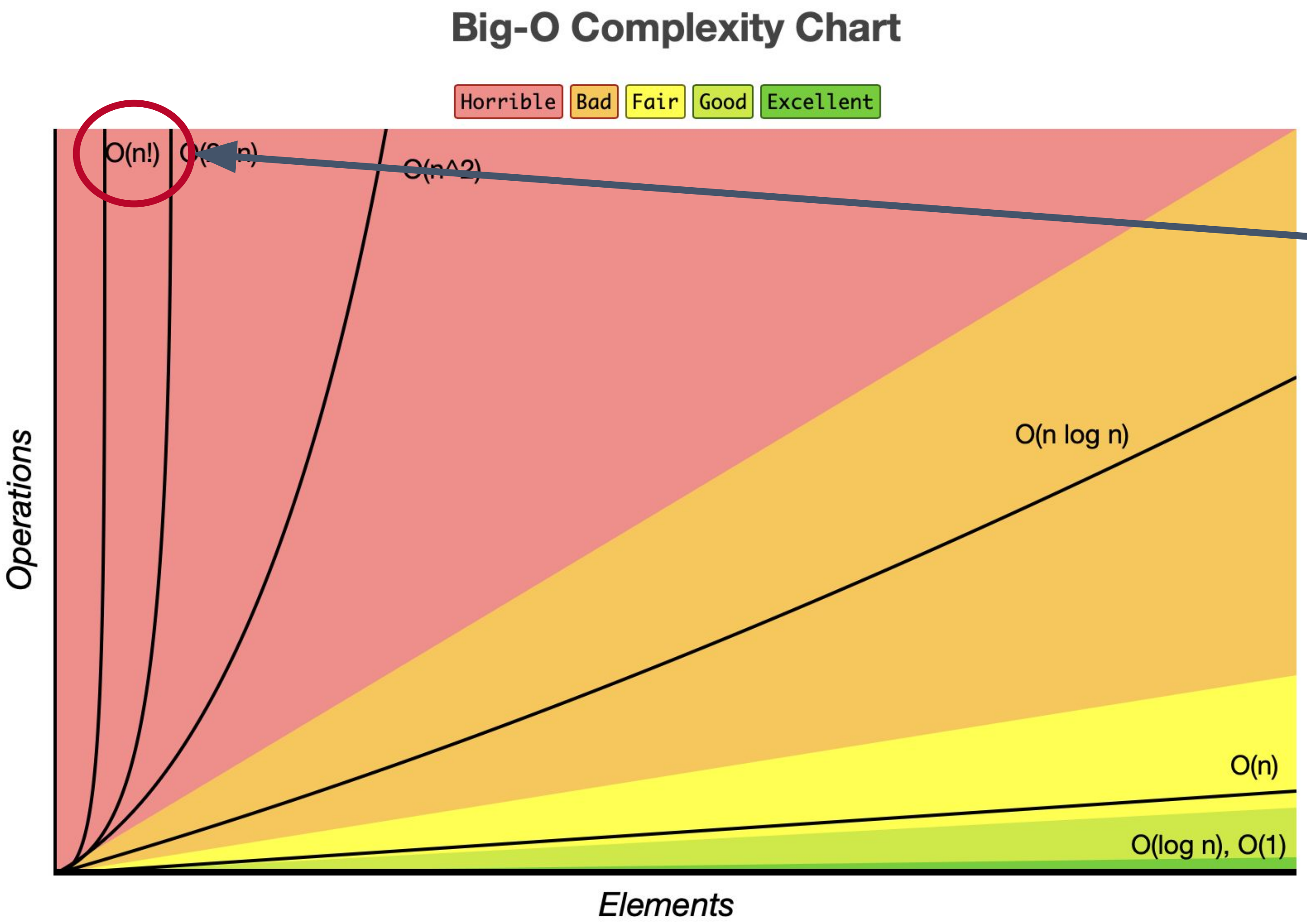
Asymptotic Notation



O(n!):
An algorithm is said to have a factorial time complexity when it grows in a factorial way based on the size of the input data

Check this out: https://www.bigocheatsheet.com/

Asymptotic Notation

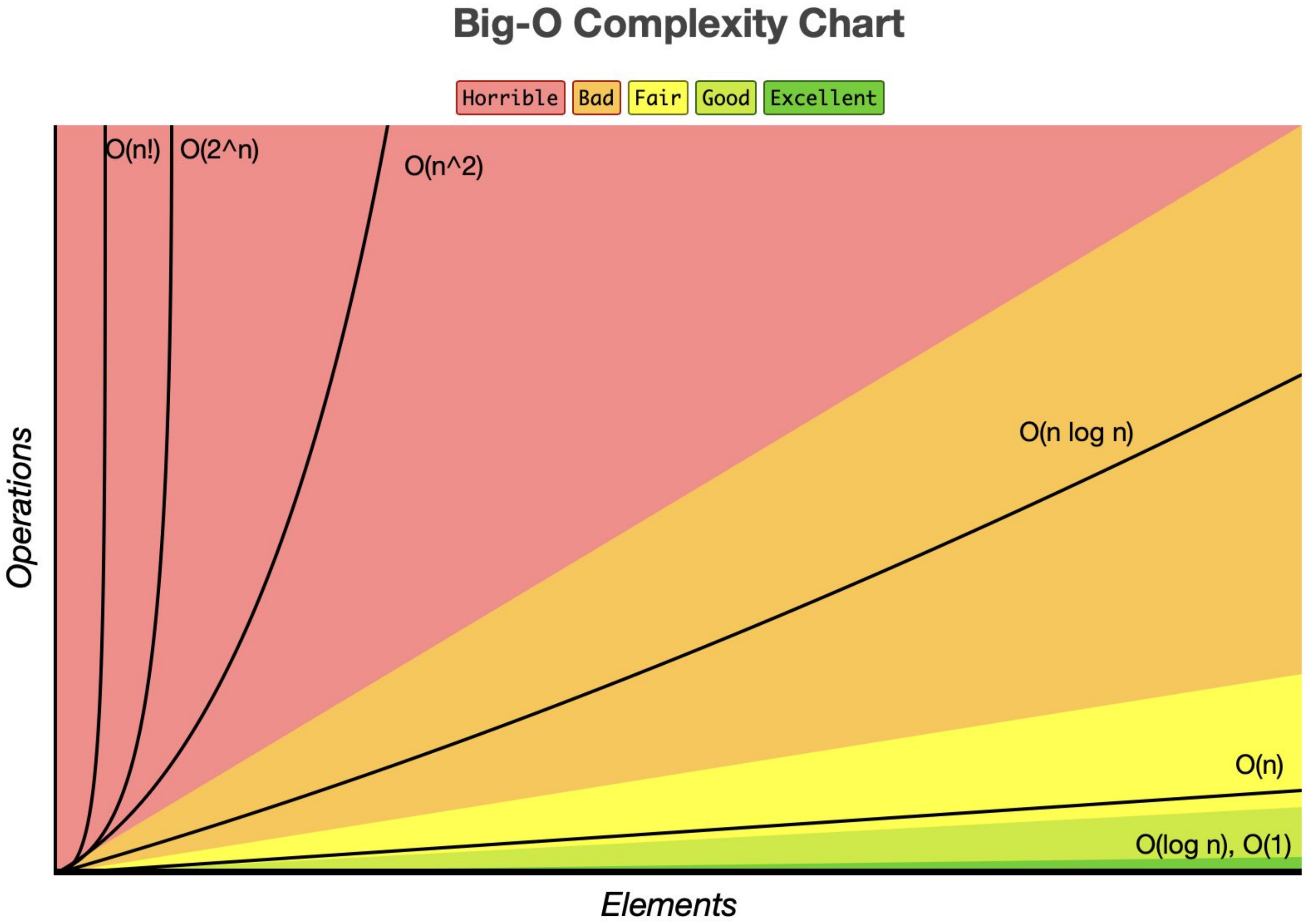


$O(n!)$ example: compute all the permutation of n elements. Factorial function grows **very** rapidly. Just to compare:

$2^{10} = 1024$
$10! = 3628800$

Check this out: https://www.bigocheatsheet.com/

Asymptotic Notation



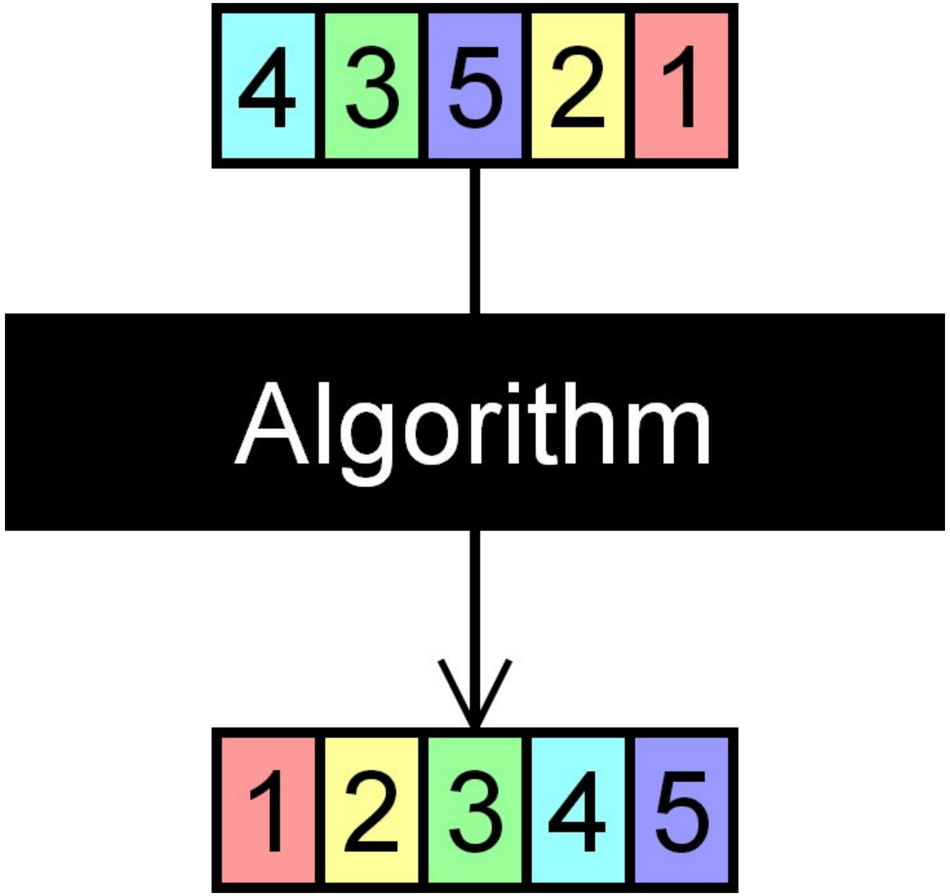
Fun fact:
<u>Unfortunately</u> a lot of interesting problems can be solved only using algorithm that run in $O(n!)$ or $O(2^n)$

Check this out: https://www.bigocheatsheet.com/

Sorting

Sorting Algorithms:

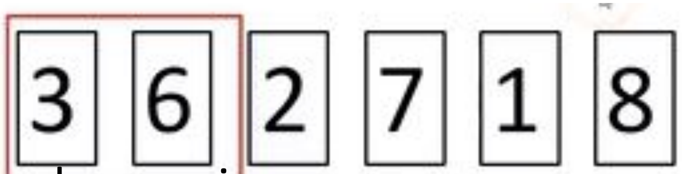
- *Bubble Sort*
- Insertion Sort
- Merge Sort
- Quick Sort
- …



Bubble Sort

General Idea:

Traverse a collection of elements moving from the start to the end



Move the largest value toward the end using pairwise comparisons and swapping

Check this out:
https://dfordeveloper.github.io/study-sorting/

Bubble Sort

Bubble Sort takes an unordered collection and makes it an ordered one.

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**35**	**12**	**101**	**5**

Before applying bubble sort

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**5**	**12**	**35**	**42**	**77**	**101**

After applying bubble sort

Bubble Sort

How does it work?

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**35**	**12**	**101**	**5**

Before applying bubble sort

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**5**	**12**	**35**	**42**	**77**	**101**

After applying bubble sort

Bubble Sort

First pass: Let's Start!

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**35**	**12**	**101**	**5**

Before applying bubble sort

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**35**	**12**	**101**	**5**

Algorithm applied

Bubble Sort

First pass: check if index 0 and 1 must be swapped

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**35**	**12**	**101**	**5**

Before applying bubble sort

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**35**	**12**	**101**	**5**

Algorithm applied

Bubble Sort

First pass: Yes! because 77 > 42

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**35**	**12**	**101**	**5**

Before applying bubble sort

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**42**	**77**	**35**	**12**	**101**	**5**

Algorithm applied

Bubble Sort

First pass: check if index 1 and 2 must be swapped

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**35**	**12**	**101**	**5**

Before applying bubble sort

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**42**	**77**	**35**	**12**	**101**	**5**

Algorithm applied

Bubble Sort

First pass: Yes! Because 77 > 35

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**35**	**12**	**101**	**5**

Before applying bubble sort

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**42**	**35**	**77**	**12**	**101**	**5**

Algorithm applied

Bubble Sort

First pass: check if index 2 and 3 must be swapped

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**35**	**12**	**101**	**5**

Before applying bubble sort

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**42**	**35**	**77**	**12**	**101**	**5**

Algorithm applied

Bubble Sort

First pass: Yes! Because 77 > 12

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**35**	**12**	**101**	**5**

Before applying bubble sort

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**42**	**35**	**12**	**77**	**101**	**5**

Algorithm applied

Bubble Sort

First pass: check if index 3 and 4 must be swapped

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**35**	**12**	**101**	**5**

Before applying bubble sort

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**42**	**35**	**12**	**77**	**101**	**5**

Algorithm applied

Bubble Sort

First pass: No! Because 77 < 105

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**35**	**12**	**101**	**5**

Before applying bubble sort

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**42**	**35**	**12**	**77**	**101**	**5**

Algorithm applied

Bubble Sort

First pass: check if index 4 and 5 must be swapped

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**35**	**12**	**101**	**5**

Before applying bubble sort

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**42**	**35**	**12**	**77**	**101**	**5**

Algorithm applied

Bubble Sort

First pass: Yes! because 101 > 5

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**77**	**42**	**35**	**12**	**101**	**5**

Before applying bubble sort

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**42**	**35**	**12**	**77**	**5**	**101**

Algorithm applied

Bubble Sort

Now, we need to repeat this process over and over until the list is ordered!

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**42**	**35**	**12**	**77**	**5**	**101**

Naive Bubble Sort Pseudocode

```
Naive Bubble Sort(A Array)
  for i in range(len(A)):
    for j in range(len(A) - 1):
      if(A[j] > A[j+1]):
       Swap(A[j], A[j+1])
    endloop
  endloop
  Return A
```

Inner loop

Outer loop

Bubble Sort

Exercise at home: Starting from the result of the first pass complete the algorithm execution to get the correct result

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**42**	**35**	**12**	**77**	**5**	**101**

Organize your workspace as follow:

Second pass:

Comparison 1: 42 > 35? Yes, Result:

42	35	12	77	5	101

Etc…

Third pass:

…

…

Final pass:

Naive Bubble Sort

Question:

- Which is the **computational complexity** ?

Naive Bubble Sort

Question:

- Which is the **computational complexity** ?

Answer:

- The **computational complexity** is $O(n^2)$

Exercise at home: formally prove the computational complexity of $O(n^2)$

Naive Bubble Sort

It seems like the naive version is a way too naive!

Question:

- Can you came up with an idea to reduce the amount of operations, just modifying **<u>the inner for loop</u>**?

Improved Bubble Sort Pseudocode

```
Improved Bubble Sort(A Array)
  for i in range(len(A)):
    for j in range(len(A) - i - 1):
      if(A[j] > A[j + 1]):
        Swap(A[j], A[j + 1])
    endloop
  endloop
  Return A
```

Inner loop

Outer loop

Improved Bubble Sort

Questions:

- Which is the computational complexity in this case?

Improved Bubble Sort

Questions:

- Which is the computational complexity in this case?

Answer:

- Asymptotically it is **always the same**! $O(n^2)$

Improved Bubble Sort

It seems like even this version can be improved!

Question:

- Can you came up with an idea to reduce the amount of operations, just using a **particular exit condition**?

A further Improvement in Bubble Sort Pseudocode

```
Flag Bubble Sort(A Array)
  for i in range(len(A)):
     swap_flag = False
     for j in range(len(A) - i - 1):
        if(A[j] > A[j + 1]):
           swap_flag = True
           Swap(A[j], A[j + 1])
     endloop
     if swap_flag is False:
        return A
  endloop
  Return A
```

Inner loop

Outer loop

Bubble Sort

Question:

- Which is the **best** case?
- What is the complexity in that case?

Bubble Sort

Question:

- Which is the **best** case?
- What is the complexity in that case?

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**5**	**12**	**35**	**42**	**77**	**101**

Answer: if the list is ordered, the complexity is $O(n)$, because we need just a single pass

Bubble Sort

Question:

- ❑ Which is the **worst** case?
- ❑ What is the complexity in that case?

Bubble Sort

Question:

- Which is the **worst** case?
- What is the complexity in that case?

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**101**	**77**	**42**	**35**	**12**	**5**

Answer: if the list is in reverse order, the complexity is $O(n^2)$, because we need compare each element with any other element within the list

Bubble Sort

Question:

- Which is the **average** case?
- What is the complexity in that case?

Bubble Sort

Question:

- ❑ Which is the **average** case?
- ❑ What is the complexity in that case?

Index:	*0*	*1*	*2*	*3*	*4*	*5*
Value:	**35**	**5**	**42**	**101**	**12**	**77**

Answer: in the average case the complexity is $O(n^2)$

Bubble Sort

Question:

- What about the **space complexity**?

Bubble Sort

Question:

- What about the **space complexity**?

Answer: in all the three versions of Bubble Sort the space complexity is $O(1)$.

Bubble sort requires only a fixed amount of extra space for the flag, and the other variables.

It is an in-place sorting algorithm, which modifies the original array's elements to sort the given array. It doesn't need extra space!