

### Enum types

I need...	Function	Example	BigO
Execute without return	Enum.each/2	Enum.each([1,2,3], &IO.puts/1)	O(n)
Transform each element	Enum.map/2	Enum.map([1,2,3], &(&1 * 2)) # [2,4,6]	O(n)
Filter elements	Enum.filter/2	Enum.filter([1,2,3], &(&1 > 1)) # [2,3]	O(n)
Accumulate a value	Enum.reduce/2	Enum.reduce([1, 2, 3, 4], fn x, acc -> x * acc end)	O(n)
Find a value	Enum.find/2	Enum.find([1,2,3], &(&1 > 1)) # 2	O(n) (worst case)
Check if all match condition	Enum.all?/2	Enum.all?([2,4,6], &rem(&1, 2) == 0) # true	O(n) (worst case)
Check if exists inside	Enum.member?/2	Enum.member?([1,2,3], 2) # true	O(n)

### Data Structure Types

Data Structure	Access Complex	Observations
list	O(n)	sorted, duplicated keys
tupple	O(1)	
map	O(1) map > 32 / O(n) map < 32	unsorted, no key duplication
keyword list	O(n)	Inserting with prepend is O(1)

Maps < 32 are sorted lists  
 Maps > 32 are hash-tree (HAMT)  
 Lists, when insert prepend for O(1), append is O(n)

### Sort Algorithms

Name	Time Complex	Comments
Merge Sort	$\square$ O(n log n) / O(n log n) / O(n log n)	Recommended in Elixir (recursive and stable)
Insertion Sort	$\checkmark$ O(n) / $\square$ O(n <sup>2</sup> ) / O(n <sup>2</sup> )	Good for small or semi-ordered lists
Radix Sort	O(nk) / O(nk) / O(nk)	Very fast on small integers
Bubble Sort	$\checkmark$ O(n) / $\square$ O(n <sup>2</sup> ) / $\times$ O(n <sup>2</sup> )	Inefficient, useful only for small or already sorted lists
Selection Sort	$\square$ O(n <sup>2</sup> ) / O(n <sup>2</sup> ) / O(n <sup>2</sup> )	Simple, but slow
QuickSort	$\checkmark$ O(n log n) / O(n log n) / $\times$ O(n <sup>2</sup> )	Better than MergeSort on average, but can degrade to O(n <sup>2</sup> )
Heap Sort	O(n log n) / O(n log n) / O(n log n)	
Counting Sort	O(n + k) / O(n + k) / O(n + k)	Only works with small numbers and limited range



By **zokratez**  
[cheatography.com/zokratez/](https://cheatography.com/zokratez/)

Published 12th March, 2025.  
 Last updated 3rd March, 2025.  
 Page 1 of 5.

Sponsored by **Readable.com**  
 Measure your website readability!  
<https://readable.com>

### Sort Algorithms (cont)

Bucket Sort                       $O(n + k) / O(n + k) / O(n^2)$                       Useful if data is evenly distributed

Avoid:

QuickSort because we can get  $O(n^2)$  if the list is already sorted.

Insertion ( $O(n^2)$ ) → If the list is inverted.

### Merge Sort

```
defmodule MergeSort do
  def sort([], do: [])
  def sort([x], do: [x])
  def sort(list) do
    mid_point = div(length(list), 2)
    {left, right} = Enum.split(list, mid_point)

    new_left = sort(left)
    new_right = sort(right)

    merge(new_left, new_right)
  end
  def merge(left, []), do: left
  def merge([], right), do: right
  def merge([l | left], [r | right]) when l <= r do
    [l | merge(left, [r | right])]
  end
  def merge([l | left], [r | right]) do
    [r | merge([l | left], right)]
  end
end
MergeSort.sort([2, 4, 6, 7, 1, 2, 3, 5])
```

### Insertion Sort

```
defmodule Sort do
  def insert([], x), do: [x]
  def insert([h | t], x) when x <= h, do: [x, h | t] # Insert before if is lower

  def insert([h | t], x), do: [h | insert(t, x)] # Continue searching if is greater
  def insertion_sort([], do: [])

  def insertion_sort([h | t]) do
    sorted_tail = insertion_sort(t)
    insert(sorted_tail, h)
  end
end
```



### Insertion Sort (cont)

```
> Sort.insertion_sort([3, 1, 4, 2])
```

### Detect Anagram

```
defmodule Anagram do
  def anagram?(word1, word2) do
    Enum.sort(String.graphemes(word1)) == Enum.sort(String.graphemes(word2))
  end
end

IO.inspect(Anagram.anagram?("listen", "silent")) # true
```

$O(n \log n)$  because Enum.sort

### Palindrome

```
defmodule Palindrome do
  def is_palindrome?(word) do
    word_to_charlist = to_charlist(word)
    (word_to_charlist == Enum.reverse(word_to_charlist))
  end
end

Palindrome.is_palindrome?("abcba")
```

Use recursive option is less performance. It uses List.last/1 and Enum.drop/2 ( $O(n)$  each), making the complexity  $O(n^2)$ .

### Fibonacci

```
defmodule Fibonacci do
  def fib(n), do: fib(n, 0, 1)
  defp fib(0, a, _b), do: a
  defp fib(n, a, b), do: fib(n - 1, b, a + b)
end

IO.inspect(Fibonacci.fib(10)) # => 55
```

$O(n)$

Fast Enum Reference	
Accumulate	reduce
Return in map or diff type than map	reduce
Return list	map
Discard / Clean	filter

Liveview callbacks	
mount	
handle_event	clicks, forms, keys, hooks.
handle_info	messages from other processes (send)
handle_params	Respond to URL changes

**Actor model**

The actor model is a concurrency paradigm in which **processes (called actors)** communicate exclusively through message passing, **without sharing memory**. The actor model uses a **FIFO queue-like** structure to process messages, but encapsulated in each actor. It is not a traditional queue because each actor is an autonomous and concurrent system.

GenServer Callback	
init	
handle_call	
handle_cast	
handle_info	
code_change	Hot code update
terminate	

Best for inserts / updates	
Lists (Prepend)	[new_item   list]
Maps insert	Map.put O(1) except hash collisions
Maps update	%{map   a: 42} better than Map.put
Keyword List (Prepend)	[{:key, val}   kw_list]

**Process Communication Elixir/Erlang**

Interprocess communication in Erlang and Elixir is based on the **actor model** and is completely **asynchronous**. Each process is lightweight, running on the BEAM virtual machine, and has its **own state** and **message queue**. Processes **do not share memory**, allowing for **lock-free concurrency** and fault tolerance. 1-Erlang cookie is same in 2 nodes

**Process Communication Elixir/Erlang (cont)**

2-Node1 ask for the Node2 port using **EPMD**

3-Node 1 send a message to the **EDP** (Erlang Distribution Protocol)

**Difference with process messaging in Java and why is better?**

Because Java uses OS threads, these can share memory, every thread use stack and OS resources.

**Strings to charlist**

```
#Directly using sigil ~c
~c"a bc"
#from symbol
def foo(name) do
  to_charlist(name)
end
```

Imports, alias, use, require		
use	Extend modules and add functionality	use GenServer
require	Use macros from another module	require Logger
import	Use functions without module prefix	import Enum, only: [map: 2]
alias	Short modules names	alias MyApp.Foo2

**Difference GenServer VS GenState**

GenServer has state and response to casts. GenState is data-flow with backpressure.

**String to List ("AB"=>["A", "B"])**

```
String.graphemes("A - B")
```

**Interfaces with Elixir (polymorphism)**

first defines the **@callbacks** in an interface module... Then implement in a module using **@behavior** Lastly, when we "override" a function defined in interface or which has a **callback** use **@impl true**

**inheritance in Elixir**

The most similar to "inheritance" is to use the **use** macro. This extends functions in the module.

OBAN vs EXq		
Characterist	OBAN	Exq
DB	Postgres	Redis
Distributed	Multi-node	Needs Redis Cluster
Retries	Configurable with backoff	Less flexible



### OBAN vs EXq (cont)

Scalable	Excelent for big systems	Good, but depends on Redis
----------	--------------------------------	----------------------------------



By [zokratez](#)  
[cheatography.com/zokratez/](https://cheatography.com/zokratez/)

Published 12th March, 2025.  
Last updated 3rd March, 2025.  
Page 5 of 5.

Sponsored by [Readable.com](#)  
Measure your website readability!  
<https://readable.com>