

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	$\checkmark O(n)$ / $\square O(n^2)$ / $O(n^2)$	Good for small or semi-ordered lists
Radix Sort	$O(nk)$ / $O(nk)$ / $O(nk)$	Very fast on small integers
Bubble Sort	$\checkmark O(n)$ / $\square O(n^2)$ / $\times O(n^2)$	Inefficient, useful only for small or already sorted lists
Selection Sort	$\square O(n^2)$ / $O(n^2)$ / $O(n^2)$	Simple, but slow
QuickSort	$\checkmark O(n \log n)$ / $O(n \log n)$ / $\times O(n^2)$	Better than MergeSort on average, but can degrade to $O(n^2)$
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
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



By **zokratez**
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>

Merge Sort (cont)

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

    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 insert ion _sort([], do: []

  def insert ion _sort([h | t]) do
    sorted_tail = insert ion _sort(t)
    insert(sorted_tail, h)
  end
end
Sort.insert ion _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
```

Detect Anagram (cont)

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

O(n log n) because Enum.sort
```

Palindrome

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

Palindrom.e.is_polindrome?("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 <table><tr><td>Acumulate</td><td>reduce</td></tr><tr><td>Return in map or diff type than map</td><td>reduce</td></tr><tr><td>Return list</td><td>map</td></tr><tr><td>Discard / Clean</td><td>filter</td></tr></table>	Acumulate	reduce	Return in map or diff type than map	reduce	Return list	map	Discard / Clean	filter	GenServer Callback <table><tr><td colspan="2">init</td></tr><tr><td colspan="2">handle_call</td></tr><tr><td colspan="2">handle_cast</td></tr><tr><td colspan="2">handle_info</td></tr><tr><td>code_c-hange</td><td>Hot code update</td></tr><tr><td colspan="2">terminate</td></tr></table> Best for inserts / updates <table><tr><td>Lists (Prepend)</td><td>[new_item list]</td></tr><tr><td>Maps insert</td><td>Map.put O(1) except hash collisions</td></tr><tr><td>Maps update</td><td>%{map a: 42} better than Map.put</td></tr><tr><td>Keyword List (Prepend)</td><td>[{:key, val} kw_list]</td></tr></table> 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 2-Node1 ask for the Node2 port using EPMD	init		handle_call		handle_cast		handle_info		code_c-hange	Hot code update	terminate		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 (cont) 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 <pre>#Directly using sigil ~c ~c"a bc" #from symbol def foo(name) do to_ charli st(- name) end</pre> Imports, alias, use, require <table><tr><td>use</td><td>Extend modules and add functionality</td><td>use GenServer</td></tr><tr><td>require</td><td>Use macros from another module</td><td>require Logger</td></tr><tr><td>import</td><td>Use functions without module prefix</td><td>import Enum, only: [map: 2]</td></tr><tr><td>alias</td><td>Short modules names</td><td>alias MyApp.Foo2</td></tr></table>	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 backpr esure. String to List ("AB"=>["A", "B"]) <pre>String.gr aph eme s("A - B")</pre> Interfaces with Elixir (polymorp-hism) 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 <table><tr><td>Caract erist</td><td>OBAN</td><td>Exq</td></tr><tr><td>DB</td><td>Postgres</td><td>Redis</td></tr><tr><td>Distri- buted</td><td>Multi-node</td><td>Needs Redis Cluster</td></tr><tr><td>Retries</td><td>Config- urable with backoff</td><td>Less flexible</td></tr></table>	Caract erist	OBAN	Exq	DB	Postgres	Redis	Distri- buted	Multi-node	Needs Redis Cluster	Retries	Config- urable with backoff	Less flexible
Acumulate	reduce																																																						
Return in map or diff type than map	reduce																																																						
Return list	map																																																						
Discard / Clean	filter																																																						
init																																																							
handle_call																																																							
handle_cast																																																							
handle_info																																																							
code_c-hange	Hot code update																																																						
terminate																																																							
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]																																																						
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																																																					
Caract erist	OBAN	Exq																																																					
DB	Postgres	Redis																																																					
Distri- buted	Multi-node	Needs Redis Cluster																																																					
Retries	Config- urable with backoff	Less flexible																																																					
Actor model The actor model is a concu- rency 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.																																																							



OBAN vs EXq (cont)

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

C

By **zokratez**
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>