

Міністерство освіти і науки, молоді та спорту України
Національний університет “Львівська політехніка”

Кафедра ЕОМ



Звіт

з домашнє завдання №25.2 з дисципліни:

“Алгоритми та моделі обчислень”

Варіант: № 24.

Виконав:

ст. групи КІ-203

Ширий Богдан Ігорович

Перевірів:

ст. викладач кафедри ЕОМ

Козак Назар Богданович

ЗАВДАННЯ:

УМОВА:

Одною з альтернатив Haskell при використанні парадигми функційного програмування є Erlang/Elixir.

Загалом, якщо Haskell можна вважати базовою стандартизованою мовою при застосуванні парадигми функційного програмування, то Erlang та Elixir володіють значною практичною цінністю. Це пов'язано з тим, що для них доступні потужні Web-фреймворки, які необхідні для швидкого написання сучасного масового програмного забезпечення. Також, отриманий байт-код після трансляції коду мовою Elixir виконується на віртуальній машині Erlang (BEAM), тому Elixir має сумісність з фреймворком Erlang/OTP та іншими бібліотеками і фреймворками мови Erlang.

Пропонується виконати домашнє завдання №25.1 повторно за допомогою Erlang або Elixir без використання готових бібліотечних реалізацій.

ВИБІР ВАРІАНТУ:

$$(N_{\text{ж}} + N_{\text{г}} + 1) \% 30 + 1 = (24 + 3 + 1) \% 30 + 1 = 28 \% 30 + 1 = 29,$$

де: $N_{\text{ж}}$ – порядковий номер студента в групі, а $N_{\text{г}}$ – номер групи.

Отож, мій шуканий варіант – це 685, 686, 681, 681, 677, 680, 681, 676, 676, 672, 679, 671, 676, 677, 672, 674, 666, 671, 672 та 667.

ВИКОНАННЯ:

Написав коди на мовах програмувань **Erlang**, виконання якого зображене на рисунку 1, та **Elixir**, виконання – на рисунку 2. При описі програм не використовував готових бібліотечних реалізацій, а використав такі парадигми функційного програмування:

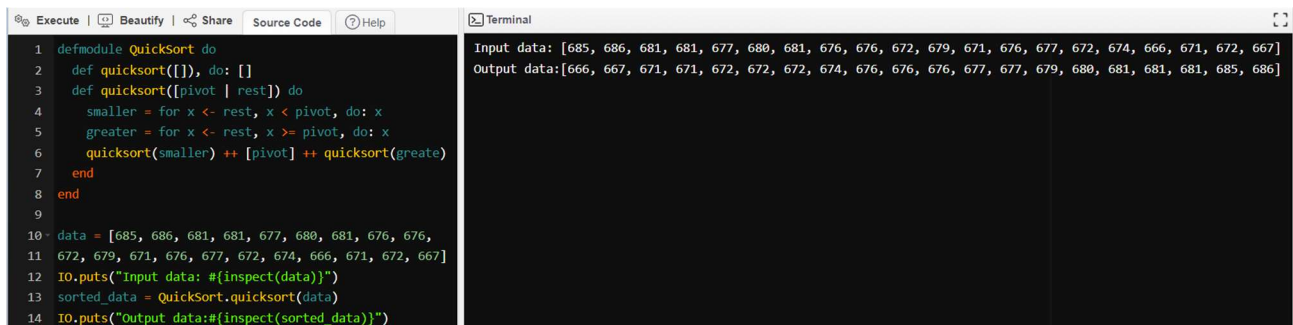
- **Функції вищого порядку:** У функції використовуються спискові включення для формування підписків, що задовольняють певні умови. Вони фільтрують елементи зі списку на основі умови, що визначена з опорним елементом.
- **Рекурсія:** Функція швидкого сортування в кодах викликає саму себе рекурсивно для сортування менших і більших підписків.
- **Імутабельність даних:** Усі дані, що передаються у функції, є незмінними. Кожен раз, коли створюється новий підсписок або рекурсивно викликається функція швидкого сортування, створюється нова структура даних, зберігаючи незмінність початкових даних.



```
1 -module(helloworld).
2 -export([start/0]).
3
4 start() ->
5     Data = [685, 686, 681, 681, 677, 680, 681, 676, 676,
6             672, 679, 671, 676, 677, 672, 674, 666, 671, 672, 667],
7     io:format("Input data: ~w ~n", [Data]),
8     SortedData = quicksort(Data),
9     io:format("Output data: ~w ~n", [SortedData]).
10
11 quicksort([]) -> [];
12 quicksort([Pivot | Rest]) ->
13     Smaller = [X || X <- Rest, X < Pivot],
14     Greater = [X || X <- Rest, X >= Pivot],
15     quicksort(Smaller) ++ [Pivot] ++ quicksort(Greater).
```

Input data: [685,686,681,681,677,680,681,676,676,672,679,671,676,677,672,674,666,671,672,667]
Output data: [666,667,671,671,672,672,672,674,676,676,676,677,677,679,680,681,681,681,685,686]

Рис. 1. Результат роботи написаного коду на Erlang.



```
1 defmodule QuickSort do
2   def quicksort([], do: [])
3   def quicksort([pivot | rest]) do
4     smaller = for x <- rest, x < pivot, do: x
5     greater = for x <- rest, x >= pivot, do: x
6     quicksort(smaller) ++ [pivot] ++ quicksort(greater)
7   end
8 end
9
10 data = [685, 686, 681, 681, 677, 680, 681, 676, 676,
11         672, 679, 671, 676, 677, 672, 674, 666, 671, 672, 667]
12 IO.puts("Input data: #{inspect(data)}")
13 sorted_data = QuickSort.quicksort(data)
14 IO.puts("Output data:#{inspect(sorted_data)}")
```

Input data: [685, 686, 681, 681, 677, 680, 681, 676, 676, 672, 679, 671, 676, 677, 672, 674, 666, 671, 672, 667]
Output data:[666, 667, 671, 671, 672, 672, 672, 674, 676, 676, 676, 677, 677, 679, 680, 681, 681, 681, 685, 686]

Рис. 2. Результат роботи написаного коду на Elixir.

Відповідно, навів текстові коди програм написаних: на мові Erlang у лістингу 1, а на мові Elixir – лістингу 2.

Лістинг 1. Код програми написаної на мові Erlang.

```
-module(helloworld).
-export([start/0]).

start() ->
    Data = [685, 686, 681, 681, 677, 680, 681, 676, 676,
            672, 679, 671, 676, 677, 672, 674, 666, 671, 672, 667],
    io:format("Input data: ~w~n", [Data]),
    SortedData = quicksort(Data),
    io:format("Output data: ~w~n", [SortedData]).

quicksort([]) -> [];
quicksort([Pivot | Rest]) ->
    Smaller = [X || X <- Rest, X < Pivot],
    Greater = [X || X <- Rest, X >= Pivot],
    quicksort(Smaller) ++ [Pivot] ++ quicksort(Greater).
```

Лістинг 2. Код програми написаної на мові Elixir.

```
defmodule QuickSort do
  def quicksort([], do: [])
  def quicksort([pivot | rest]) do
    smaller = for x <- rest, x < pivot, do: x
    greater = for x <- rest, x >= pivot, do: x
    quicksort(smaller) ++ [pivot] ++ quicksort(greater)
  end
end

data = [685, 686, 681, 681, 677, 680, 681, 676, 676,
        672, 679, 671, 676, 677, 672, 674, 666, 671, 672, 667]
IO.puts("Input data: #{inspect(data)}")
sorted_data = QuickSort.quicksort(data)
IO.puts("Output data:#{inspect(sorted_data)}")
```