Lista 2

1. Liczby Fibonacciego są zdefiniowane następująco:

$f(0) = 0$
$f(1) = 1$
$f(n+2) = f(n) + f(n+1)$

Napisz dwie funkcje, które dla danego n znajdują n-tą liczbę Fibonacciego: pierwszą opartą bezpośrednio na powyższej definicji i drugą, wykorzystującą rekursję ogonową. Porównaj ich szybkość wykonania, obliczając np. 42-gą liczbę Fibonacciego.

Funkcja, mierząca czas wykonania zadanej funkcji f dla argumentu x, może wyglądać tak:

```
let time f x =
  let t = Sys.time()
  and fx = f x
  in let _ = Printf.printf "Czas wykonania: %fs\n" (Sys.time() -. t)
     in fx
```

2. Dla zadanej liczby rzeczywistej a oraz dokładności ε można znaleźć pierwiastek trzeciego stopnia z a wyliczając kolejne przybliżenia x_i tego pierwiastka (metoda Newtona-Raphsona):

$x_0 = a/3$ dla $a > 1$
$x_0 = a$ dla $a \leq 1$
$x_{i+1} = x_i + (a/x_i^2 - x_i)/3$

Dokładność (względna) jest osiągnięta, jeśli $|x_i^3 - a| \leq \varepsilon * |a|$.
Napisz efektywną (wykorzystującą rekursję ogonową) funkcję root3: float -> float, która dla zadanej liczby a znajduje pierwiastek trzeciego stopnia z dokładnością 10^{-55}.
Uwaga. Pamiętaj, że OCaml nie wykonuje automatycznie żadnych koercji typów.

W poniższych funkcjach wykorzystaj mechanizm dopasowania do wzorca.

3. Zdefiniuj operator infiksowy <-> , który dla dwóch wektorów trójwymiarowych (w postaci krotki) zwróci ich odległość euklidesową, np. (1.,1.,1.) <-> (1.,1.,0.) => 1.

4. Zdefiniuj operator infiksowy <--, który do listy uporządkowanej niemalejąco wstawia w odpowiednie miejsce dany element, np.: [1;3;5;5;7] <-- 3 => [1;3;3;5;5;7]
Jaka jest złożoność obliczeniowa tej funkcji? Zilustruj rysunkiem reprezentację wewnętrzną obu list (patrz wykład, str. 23 – 26).

5. Napisz funkcję take: int -> 'a list -> 'a list, gdzie take k [x_1; ...; x_n]) == [x_1; ...; x_k].
Np. take 2 [1;2;3;5;6] => [1;2], take (-2) [1;2;3;5;6] => [], take 8 [1;2;3;5;6] => [1;2;3;5;6].

6 Napisz funkcję drop: : int -> 'a list -> 'a list, gdzie drop k [x_1; ...; x_n] == [x_{k+1}; ...; x_n].
Np. drop 2 [1;2;3;5;6] => [3;5;6], drop (-2) [1;2;3;5;6] => [1;2;3;5;6], drop 8 [1;2;3;5;6] =>[].

7. Zdefiniuj funkcję replicate: int list -> int list, która z danej listy liczb naturalnych tworzy listę, w której każdy element wejściowej listy jest tyle razy powtórzony, jaką ma wartość, np. replicate [1;0;4;-2;3] => [1;4;4;4;4;3;3;3].