Řešení domácích úkolů - sort

```
let msort list =
  let rec split accx accy = function
    | []      -> accx, accy
    | x :: xs -> split accy (x :: accx) xs

  let rec merge up acc = function
    | x::xs, y::ys when(if up then x>y else x<y)-> merge up (x::acc) (xs, y::ys)
    | x::xs, y::ys                              -> merge up (y::acc) (x::xs, ys)
    | [], xs | xs, []                           -> List.rev xs @ acc

  let rec msort’ asc = function
    | []  -> []
    | [x] -> [x]
    | l   -> let left, right = split [] [] l
             merge up [] (msort’ (not up) left, msort’ (not up) right)

  msort’ true list
```

Řešení domácích úkolů - opti

```
type ’a tree = Nil | Node of ’a * (’a tree) * (’a tree)
type hodnota = { root : int; cena : int }

let opti xs =
  let a, c = xs |> List.sort_by fst |> Array.of_list |> Array.unzip
  let n = a.Length

  let csum = Array.zero_create (n+1)
  for i = 1 to n do csum.[i] <- csum.[i−1] + c.[i−1]
  let csum i j = csum.[j+1] − csum.[i]

  let ceny = Array2.create_based (−1) (−1) (n+2) (n+2) { root = 0; cena = 0; }
  let update_ceny i j =
    let best = ref { root = 0; cena = int.MaxValue }
    for k = i to j do
      if ceny.[i, k−1].cena + ceny.[k+1, j].cena <= (!best).cena then
        best := { root = k; cena = ceny.[i, k−1].cena + ceny.[k+1, j].cena }

    ceny.[i, j] <- { root = (!best).root; cena = (!best).cena + csum i j}

  for i = 0 to n−1 do ceny.[i, i] <- { root = i; cena = c.[i]; }
  for len = 1 to n−1 do
    for i = 0 to n−len−1 do
      update_ceny i (i+len)

  let rec build i j = if i > j then Nil
                                else let r = ceny.[i, j].root
                                     Node (a.[r], build i (r−1), build (r+1) j)

  build 0 (n−1)
```

To je O(N^3), O(N^2) je jenom malá změna:

```
  let update_ceny i j =
    ...
    for k = ceny.[i, j−1].root to ceny.[i+1, j].root do
    ...
```

F# - základní typy

```
bool    System.Boolean   true, false
  && , || : bool -> bool -> bool
  =, !=, <, >, <=, >= : ’a -> ’a -> bool
  min, max : ’a -> ’a -> ’a
```

```
byte           System.Byte     0uy        sbyte        System.SByte    0y
int16          System.Int16    0s         uint16       System.UInt16   0us
int,int32      System.Int32    0          uint32       System.UInt32   0u
int64          System.Int64    0L         uint64       System.UInt64   0UL
nativeint      System.IntPtr   0n         unativeint   System.UIntPtr  0un
single,float32 System.Single   0.0f       double,float System.Double   0.0
decimal        System.Decimal  0M
bigint         Math.BigInt     0I         bignum       Math.BigNum     0N
unit           Core.Unit       ()
  +, -, *, **, /, %, ~- : overloaded
  &&&, |||, ^^^, ~~~, <<<, >>> : overloaded
  abs, acos, asin, atan, atan2, ceil, cos, cosh, exp, floor, log, log10,
  pown, round, sign, sin, sinh, sqrt, tan, tanh, truncate : overloaded
  nan, infinity : double        nanf, infinityf : float
  byte, sbyte, int16, uint16, int, int32, ..., decimal : conversions

char    System.Char    'a', '\t', ...; konverze pomoci char
string  System.String  "ahoj", "C:\\c", @"C:\c", "abc"B : byte[]
  konverze pomoci string
  + , ^ : string -> string -> string
  System.Text.StringBuilder, metody
    Append, Insert, Remove, Replace, EnsureCapacity, ToString, Chars
  printf, printfn, sprintf

                            F# - strukturované typy
                            -----------------------
'a option   nebo   option<'a>;    type 'a option = Option<'a>
Option.{get is_some, is_none, length, map, iter, to_array, to_list}

'a list nebo list<'a>;            type 'a list = List<'a>
[], x :: xs, [1; 2; 3], xs @ ys
List.{length, hd, tl, init, append, (min, max, sort, sum)[_by]}
List.{filter, map, map2, mapi, mapi2, iter, iter2, iteri, iteri2}
List.{fold_left, fold_right, scan_left, scan_right, reduce_left, reduce_right}
List.{zip, zip3, unzip, unzip3, concat, map_concat}
List.{to_array, of_array, to_seq, of_seq}

'a []
[||]; [|1; 2; 3|]
Array.{length, init, create, zero_create, append, (min, max, sort, sum)[_by]}
Array.{filter, map, map2, mapi, mapi2, iter, iter2, iteri, iteri2}
Array.{fold_left, fold_right, scan_left, scan_right, reduce_left, reduce_right}
Array.{zip, zip3, unzip, unzip3, concat, map_concat}
Array.{to_list, of_list, to_seq, of_seq}
Array.empty<'a> : 'a []

'a [,], 'a [,,]
Array[23].{length1, length2, ?length3?, create, zero_create, init}
Array[23].{iter, iteri, map, mapi}

'a * 'b * 'c,   fst,   snd
'a -> 'b

'a lazy, lazy<'a>;   type 'a lazy = Lazy<'a>
(lazy exp : 'a lazy).Force ()
let force (a : Lazy<'a>) = a.Force ()

'a ref
ref : 'a -> 'a ref
(!) : 'a ref -> 'a
(:=) : 'a ref -> 'a -> unit
incr, decr : 'a ref -> unit

'a seq nebo seq<'a>;   type 'a seq = IEnumerable<'a>
IEnumerable<'a> ma funkci GetEnumerator vracejici IEnumerator<'a>
IEnumerator<'a> ma vlastnost Current a funkci MoveNext
Seq.{length, append, concat, filter, fold, hd, skip, skipWhile, take, takeWhile}
Seq.{map, map2, mapi, iter, iter2, iteri, fold, reduce, scan, zip, zip3}
Seq.{(max, min, sort, sum)[_by], cache}
```