1. (10p) Write a recursive function `flip` that takes a binary tree as input and returns a binary tree that it is its mirror image. You can represent binary trees as nested structures:

Nested (recursive) representation: `(root (left subtree) (right subtree))`

Examples:

```sml
(flip '(1 2 3)) should return (1 3 2)
(flip '(1 (2 3 4) ())) should return (1 () (2 4 3))
(flip '(1 (2 (3 4 5) (10 11 12)) (6 () (7 () 8))) should return
   (1 (6 (7 8 () ()) (2 (10 12 11) (3 5 4))))
```
2. (20p)

a) (5p) Write a lisp function `funny_first` that takes a list of flat lists and returns a new list composed of the first elements of the original flat lists.

b) (5p) Write a lisp function `funny_last` that takes a list of flat lists as its argument and returns a new list composed of the last elements of the original flat lists.

c) (5p) Write a lisp function `funny_len` that takes a list of flat lists as its argument and returns the sum of the lengths of the nested lists.

d) (5p) Write a lisp function `funny_sum` that takes a list of flat lists of numbers and returns the sum of the elements of the nested lists.

Examples:

(funny_first '((A B) (C) (D E) (F G H)))  should return (A C D F)
(funny_last '((A B) (C) (D E) (F G H)))  should return (B C E H)
(funny_len '((A B) (C) (D E) (F G H)))  should return 8
(funny_sum '((1 2) (3) (4 5) (10 20 30)))  should return 75
3. (15p) List the order in which nodes are visited in the tree below for each of the following search strategies (choosing leftmost branch in all cases):

a. Depth-first search

b. Depth-first iterative-deepening search (increasing the depth by 1 each iteration)

c. Breadth-first search
4. (35p) A simple sliding-tile puzzle consists of 4 numbered tiles and an empty space. The puzzle has two legal moves with associate costs: (i) A tile may move into an adjacent empty location. This has a cost of 1. (ii) A tile can move over one other tile into the empty position. This has a cost of 2. The start and goal positions are given below.

\[
\begin{array}{ccc}
1 & 2 & 3 \\
\hline
\text{start} & & \\
\end{array} \quad \begin{array}{ccc}
3 & 1 & 2 \\
\hline
\text{goal} & & \\
\end{array}
\]

Set up a state-space search formulation for this puzzle:

- **a)** (2p) Specify the form of state descriptions, the starting state, and the goal state for this problem.
- **b)** (5p) Name the operators on states and describe what each operator does to a state description.
- **c)** (3p) Propose a heuristic function $\hat{h}$ for solving this problem.
- **d)** (20p) Use the $A^*$ algorithm to find a solution path.
- **e)** (5p) Is your solution path optimal? Give a formal argument.
5. (30p) A simple version of the nim game is played as follows: Two players alternate in removing stones from three piles initially containing one, one, and five stones, respectively. The player who picks up the last stone wins. Each player can pick one or more stones from a single pile; at least one stone has to be picked every time. At every turn the players can pick from different piles.

   a) (20p) Show, by drawing a game tree, which player can always win. You may use either DFS or BFS method to generate nodes.

   b) (10p) Is it necessary to generate the whole tree to find a winning strategy? Explain why or why not. Is it possible to use the alpha-beta pruning on this game tree?