1. explain why nim positions (1 2 3) and (2 1 3) are isomorphic.

2. for each nim position, give its value — win or loss for player-to-move? — and explain briefly how you know your answer.
   
   (0) (0 0 0 0) (936 936) (1 1 1) (1 2 3) (3 5 7)

3. in class, we discussed three options for finding nim values by searching the state space:
   
   1) minimax the tree of all possible continuations of the starting position, ignoring transposition and isomorphisms
   2) minimax the tree reduced by considering isomorphic positions to be the same, e.g. considering (1 2 3) to be the same position as (2 1 3).
   3) effectively reduce the tree in 2) to a directed acyclic graph by using a table to store results of positions that already computed

   options 1) and 2) are wasteful. by hand, compute the number of nodes in the search space for nim(2 2) using each option 1) 2) 3). Check your answer using program nimstates.py

   Using the program, for each option 1), 2), 3), give the number of nodes in the search space for nim(3, 3, 3).

4. function winning in nimnega.py uses a negamax-style algorithm to compute nim values using option 3) above.

   in tic-tac-toe, we returned -1, 0, 1 for loss, win, draw. what values does winning return? does it matter that they are not integers?

   how does winning manage to treat isomorphic permutations — e.g. (1 2 3) (1 3 2) (2 1 3) (2 3 1) (3 2 1) (3 1 2) — the same?

   when winning solves nim(3 3 3), how many entries are in the final dictionary? how do you know? explain briefly.

   explain why winning.py’s algorithm is closer to negamax than minimax. give details.

   explain why winning.py’s algorithm is closer to nega-alphabeta than negamax. give details.
5. Below are the lines of **winning**, scrambled. Put them back in order. Indent properly.

```python
# nim_psn not in dictionary, so update before we return
cchild = tuple(sorted(psn[:j] + (k,) + psn[j+1:]))
def winning(nim_psn, sd, depth): # tuple, dictionary, recursion depth
    for j in range(len(psn)): # each pile
        for k in range(psn[j]): # number of stones that will remain in that pile
            if all(p == 0 for p in nim_psn): # we lose if every pile empty
                if depth == 0: print(child) # show a winning move
            if nim_psn in sd:
                if not winning(child, sd, depth+1):
                    psn = tuple(sorted(nim_psn))
                    return False
                return True
            return sd[nim_psn]
        sd.update({ nim_psn: False }) # update before return
    return False
    return sd[nim_psn]
    sd.update({ nim_psn: False }) # update before return
    sd.update({ nim_psn: True }) # update before return
```

6. Here is output for nim(1, 2, 2) from current class repo version of `nimnega.py`.

```
(1, 2, 2)
(0, 2, 2)
    (0, 0, 2)
        (0, 0, 0)
(0, 1, 2)
    (0, 0, 1)
(0, 1, 1)

winning: move to (0, 2, 2)
```

(i) for each line of output, explain what is happening in the search.
(ii) show the output for nim(1, 2, 3)
(iii) show the first ten lines of output from nim(3, 5, 7)