# EECS 281, Week 7: Wednesday

## Winning Games

Suppose you are playing a card game where you deal *n* cards. Each player picks a card from the left end or from the right end until no cards remain. The player whose cards sum up to the highest number wins.

Design an optimal strategy for playing this game against Maxim, who always uses the greedy approach by selecting the card with the highest number. Implement your strategy in `optimalStrategy`, which should return the highest sum you can get by optimally selecting cards when playing against Maxim.

```
const int SENTINEL = -1;

int optimalStrategyRecursive(const vector<int> &cards) {
    vector<vector<int>> cache = vector<vector<int>>(cards.size() + 1,
                                    vector<int>(cards.size() + 1, SENTINEL));
    return optimalHelper(cards, 0, cards.size(), cache);
}

int optimalHelper(const vector<int> &cards, size_t start, size_t end,
                  vector<vector<int>> &cache) {
    // ensure even cards
    assert((end - start) % 2 == 0);
    if (end - start == 0) {
        return 0;
    } else if (cache[start][end] == SENTINEL) {
        int chooseLeft = cards[start] +
                         min(optimalHelper(cards, start + 1, end - 1, cache),
                             optimalHelper(cards, start + 2, end, cache));
        int chooseRight = cards[end - 1] +
                          min(optimalHelper(cards, start, end - 2, cache)
                              optimalHelper(cards, start + 1, end - 1, cache));
        cache[start][end] = max(chooseLeft, chooseRight);
    }
    return cache[start][end];
}

int optimalStrategyIterative(const vector<int> &cards) {
    int n = cards.size();
    vector<vector<int>> table = vector<vector<int>>(n, vector<int>(n));
    for (int gap = 0; gap < n; gap += 1) {
        int start = 0;
        int end = gap;
        while (end < n) {
            // cards without two on left
            int right = ((start + 2) <= end) ? table[start + 2][end] : 0;
            // cards without one on left and one on right
            int middle = ((start + 1) <= (end - 1)) ? table[start + 1][end - 1] : 0;
            // cards without two on right
            int left = (start <= (end - 2))? table[start][end - 2]: 0;

            table[start][end] = max(cards[start] + min(right, middle),
                                    cards[end] + min(middle, left));
            start += 1;
            end += 1;
        }
    }
    return table[0][n-1];
}
```

*When finished, take a picture of your answer and email it to Maxim.*