Hw 3 Questions

1. No. The values on the table are based on the intrinsic properties of the game, not all the choices the players make. Therefore, Alice can still follow the table regardless of Bob's choice.

2.

a) Steps to compute $C(n, n/2)$:

1. Each computation of $C(n,k)$ makes 2 recursive calls: $C(n-1,k-1)$ and $C(n-1,k)$, making a binary recursion tree (BRT).
2. Since n is even, we are computing $C(n, n/2)$
3. For each recursion call, we reduce n and k by 1 or just n by 1. Therefore, this BRT has a depth of $n/2$ since we reach the base cases when $k=0$ || $k=n$.
4. Therefore, the tree has approximately $2^{n/2}$ nodes.

b) Steps:

1) Initialize a 2d Array, memo, such that memo[i][j] stores the result of $C(i,j)$

2) Base cases:

Set memo[n,0] = 1 for all n as $C(n,0)=1$

Set memo[n,n] = 1 for all n as $C(n,n)=1$

3) Recursion:

For each $0<k<n$, compute $C(n,k)=C(n-1,k-1)+C(n-1,k)$ and store the result in memo[n][k]

4. The result for $C(n,k)$ will be stored in memo[n][k]

With memorization, each unique subproblem is computed once. Therefore, the number of unique subproblems for $C(n,k)$ is $O(n*k)$ which is $O(n^2)$.

3. function palidrome(s):

```
n = length of s
dp = array [n][n] filled w/ 0's

Base case: Single char's are palindromes of length 1
for i from 0 to n-1:
  dp[i][i] = 1

for length from 2 to n:
  for i from 0 to n-length:
    j = i + length - 1
    if S[i] == S[j]:
      dp[i][j] = dp[i+1][j-1] + 2
    else:
      dp[i][j] = max(dp[i+1][j], dp[i][j-1])
return dp[0][n-1]
```

HW3 Questions cont.

4. In order to win, either team needs, at least, $(n+1)/2$ wins.
Therefore if the Anteaters win the series, then $V(\frac{n+1}{2}, j) = 1$ for any j
and if the Bears win the series, then $V(i, \frac{n+1}{2}) = 1$ for any i.
Recursive case:
For any game state (i, j) where $i < (n+1)/2$ and $j < (n+1)/2$, there are 2 possible outcomes:

Anteaters win next game: prob of reaching $(i+1, j)$ becomes $p \cdot V(i, j)$
Bears win next game: prob of reaching $(i, j+1)$ becomes $(1-p) \cdot V(i, j)$

Therefore, we can define $V(i, j) = p \cdot V(i-1, j) + (1-p) \cdot V(i, j-1)$

Algorithm:
```
function anteaters_win_prob(n, p):
    games_needed = (n+1)/2
    dp = array[games_needed+1][games_needed+1] filled w/ 0

    for i from 0 to games_needed - 1:
        dp[games_needed][j] = 1  # Anteaters win
    for j from 0 to games_needed - 1:
        dp[j][games_needed] = 0  # Bears win
    dp[0][0] = 1  # no games played

    for i from 1 to games_needed:
        for j from 1 to games_needed:
            dp[i][j] = p * dp[i-1][j] + (1-p) * dp[i][j-1]

    result = 0
    for i from games_needed - 1:
        result += dp[games_needed][j]
    return result
```