Still Another Idea from CS61A

- The problem is that we are recomputing intermediate results many times.
- Solution: *memoize* the intermediate results. Here, we pass in an $N \times N$ array ($N =$ `V.length`) of memoized results, initialized to -1.

```
int bestSum (int[] V, int left, int right, int total, int[][] memo) {
  if (left > right)
    return 0;
  else if (memo[left][right] == -1) {
    int L = total - bestSum (V, left+1, right, total-V[left], memo);
    int R = total - bestSum (V, left, right-1, total-V[right], memo);
    memo[left][right] = Math.max (L, R);
  }
  return memo[left][right];
 }
}
```

- Now the number of recursive calls to `bestSum` must be $O(N^2)$, for $N =$ the length of V, an enormous improvement from $\Theta(2^N)$!