## Largest Sum of Averages

We partition a row of numbers `A` into at most `K` adjacent (non-empty) groups, then our score is the sum of the average of each group. What is the largest score we can achieve?

Note that our partition must use every number in A, and that scores are not necessarily integers.

**Example:**
**Input:**
A = [9,1,2,3,9]
K = 3
**Output:** 20
**Explanation:**
The best choice is to partition A into [9], [1, 2, 3], [9]. The answer is 9 + (1 + 2 + 3) / 3 + 9 = 20.
We could have also partitioned A into [9, 1], [2], [3, 9], for example.
That partition would lead to a score of 5 + 2 + 6 = 13, which is worse.

### Note:

- `1 <= A.length <= 100`.
- `1 <= A[i] <= 10000`.
- `1 <= K <= A.length`.
- Answers within `10^-6` of the correct answer will be accepted as correct.

## Solution 1

`search` return the result for `n` first numbers to `k` groups.
It's top-down solution and it keeps all process to memory.
So it's like a DP solution while DP is bottom-up.

Time complexity: `O(KN^2)`

C++:

```
    double memo[200][200];
    double largestSumOfAverages(vector<int>& A, int K) {
        memset(memo, 0, sizeof(memo));
        int N = A.size();
        double cur = 0;
        for (int i = 0; i < N; ++i) {
            cur += A[i];
            memo[i + 1][1] = cur / (i + 1);
        }
        return search(N, K, A);
    }

    double search(int n, int k, vector<int>& A) {
        if (memo[n][k] > 0) return memo[n][k];
        double cur = 0;
        for (int i = n - 1; i > 0; --i) {
            cur += A[i];
            memo[n][k] = max(memo[n][k], search(i, k - 1, A) + cur / (n - i));
        }
        return memo[n][k];
    }
```

Java:

```
    public double largestSumOfAverages(int[] A, int K) {
        int N = A.length;
        double[][] memo = new double[N+1][N+1];
        double cur = 0;
        for (int i = 0; i < N; ++i) {
            cur += A[i];
            memo[i + 1][1] = cur / (i + 1);
        }
        return search(N, K, A, memo);
    }

    public double search(int n, int k, int[] A, double[][] memo) {
        if (memo[n][k] > 0) return memo[n][k];
        double cur = 0;
        for (int i = n - 1; i > 0; --i) {
            cur += A[i];
            memo[n][k] = Math.max(memo[n][k], search(i, k - 1, A, memo) + cur / (n
- i));
        }
        return memo[n][k];
    }
```

Python

```python
    def largestSumOfAverages(self, A, K):
        memo = {}
        def search(n, k):
            if (n, k) in memo: return memo[n, k]
            if k == 1:
                memo[n, k] = sum(A[:n]) / float(n)
                return memo[n, k]
            cur, memo[n, k] = 0, 0
            for i in range(n - 1, 0, -1):
                cur += A[i]
                memo[n, k] = max(memo[n, k], search(i, k - 1) + cur / float(n - i))
            return memo[n, k]
        return search(len(A), K)
```

written by lee215 original link here

## Solution 2

Let `f[i][j]` be the largest sum of averages for first `i + 1` numbers `(A[0], A[1], ... , A[i])` to `j` groups. `f[i][j]` consists of two parts: first `j-1` groups' averages and the last group' s average. Considering the last group, its last number must be `A[i]` and its first number can be from `A[1]` to `A[i]`. Suppose the last group starts from `A[p+1]`, we can easily get the average form `A[p+1]` to `A[i]`. The sum of first `j-1` groups' average is `f[p][j-1]` which we have got before. So now we can write the DP equation:
`f[i][j] = max {f[p][j-1] + (A[p+1] + A[p+2] + ... + A[i]) / (i - p)}, p = 0,1,...,i-1`

```
class Solution {
    public double largestSumOfAverages(int[] A, int K) {
        if (K == 0 || A.length == 0) {
            return 0;
        }
        int l = A.length;
        double[][] f = new double[l][K + 1];
        double[] s = new double[l + 1];
        for (int i = 1; i <= l; i++) {
            s[i] = s[i - 1] + A[i - 1];
            f[i - 1][1] =  s[i] / i;
        }
        for (int j = 2; j <= K; j++) {
            for (int i = 0; i < l; i++) {
                double max = Double.MIN_VALUE;
                for (int p = 0; p < i; p++) {
                    double sum = f[p][j - 1] + (s[i + 1] - s[p + 1]) / (i - p);
                    max = Double.max(sum, max);
                }
                f[i][j] = max;
            }
        }
        return f[l - 1][K];
    }
}
```

written by Kimulsanne original link here

## Solution 3

```
double largestSumOfAverages(vector<int>& A, int K) {
    if(A.empty() || K == 0)
        return 0;
    vector<vector<double>> dp(K+1,vector<double>(A.size(),0));
    vector<int> sum;
    sum.push_back(A[0]);
    for(int i = 1; i < A.size(); i++)
        sum.push_back(A[i] + sum.back());

    for(int k = 1; k <= K; k++){
        for(int i = k-1; i < A.size(); i++){
            if(k == 1)
                dp[k][i] = double(sum[i])/(i+1);
            else{
                for(int j = k-2 ; j < i; j++){
                    dp[k][i] = max(dp[k-1][j] + double(sum[i] - sum[j]) / (i - j),d
p[k][i]);
                }
            }
        }
    }
    return dp[K][A.size()-1];
}
```

written by lgcly original link here