Quadratic-Time Algo

```
def prefix_average(S):
    n = len(S)                  # executes in constant time, O(n)
    A = [0] * n                 # create list of n zeros → const # of primitive elements, thus O(n)
    for j in range(n):
        total = 0               # computing S[0]+...+S[j]
        for i in range(j+1):
            total += S[i]
        A[j] = total / (j+1)    # record average
    return A
```

→ there are 2 nested for-loops;

outer loop:
- outer loop (controlled by j) is executed n times for $j = 0, \ldots, n-1$
- thus, total = 0 AND A[j] = total/(j+1) executed n times each
- implies these statements contribute # of primitive operations proportional to n, thus $O(n)$ time

inner loop:
- body of inner loop (controlled by i) is executed j+1 times
- thus, total += S[i] is executed $1+2+3+\ldots+n$ times
- $1+2+3+\ldots+n = \frac{n(n+1)}{2}$ } $O(n^2)$

→ The running time is thus $O(n^2)$.

```
def prefix_average1(S):
    n = len(S)
    A = [0] * n
    for j in range(n):
        A[j] = sum(S[0:j+1]) / (j+1)
    return A
```

→ replaced inner loop using single expr. sum to compute partial sum
→ sum(S[0:j+1]) is a function call and takes O(j+1) time
→ the slice also takes O(j+1) time
→ thus, this implementation is also $O(n^2)$