Linear-Time Algorithm

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

running time analysis

→ single for loop controlled by j: $O(N)$

→ body of loop executed n times

- thus, total += S[j] and A[j] = total / (j+1) are executed n times each
- they are each $O(1)$ so contribution is $O(N)$

→ Overall $O(N)$

Three-Way Set Disjointness

→ given three sequences of #'s A, B, C. Determine if the intersection of the three sequences is empty, namely, there is no element x such that $x \in A$, $x \in B$, $x \in C$.

```
def disjoint1(A, B, C):
    for a in A:
        for b in B:
            for c in C:
                if a == b == c:
                    return False
    return True
```

→ loops through each possible triple of values from 3 sets to see if they're equivalent

→ if each original set has size n, then the worst-case running time is $O(n^3)$

→ we can improve with one understanding: once inside loop B, if a and b are not equal, it is a waste of time to iterate through C

```
def disjoint2(A, B, C):
    for a in A:
        for b in B:
            if a == b:
                for c in C:
                    if a == c:
                        return False
    return True
```

→ there are quadratically many (a, b) pairs

→ if A and B are sets of distinct elements, there can be at most $O(n)$ pairs where $a = b$

→ therefore, the innermost loop, over C, executes at most n times