Even Subsequence

- **Problem**

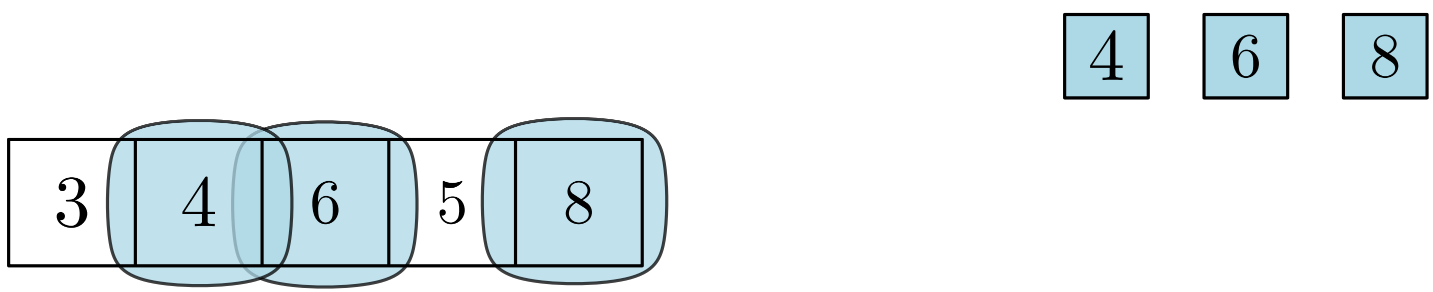
Given a sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$, where $a_i \in \mathbb{N}_0^+$, return the number of contiguous non-empty subsequences B of A whose elements sum to an even number.

3	4	6	5	8

Even Subsequence

- **Problem**

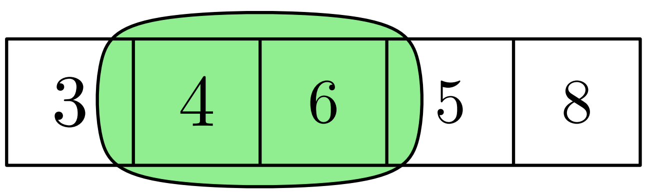
Given a sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$, where $a_i \in \mathbb{N}_0^+$, return the number of contiguous non-empty subsequences B of A whose elements sum to an even number.

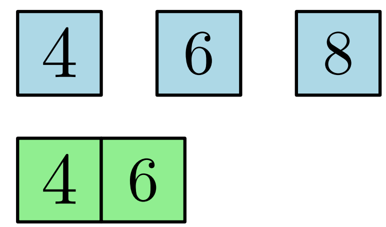


Even Subsequence

- **Problem**

Given a sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$, where $a_i \in \mathbb{N}_0^+$, return the number of contiguous non-empty subsequences B of A whose elements sum to an even number.

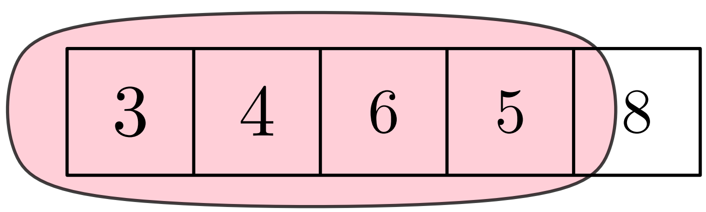


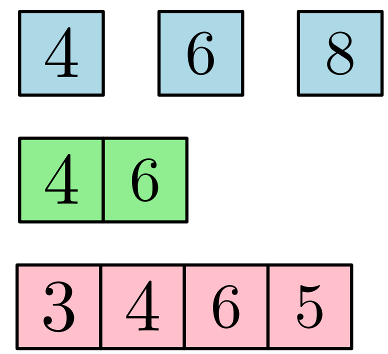


Even Subsequence

- **Problem**

Given a sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$, where $a_i \in \mathbb{N}_0^+$, return the number of contiguous non-empty subsequences B of A whose elements sum to an even number.

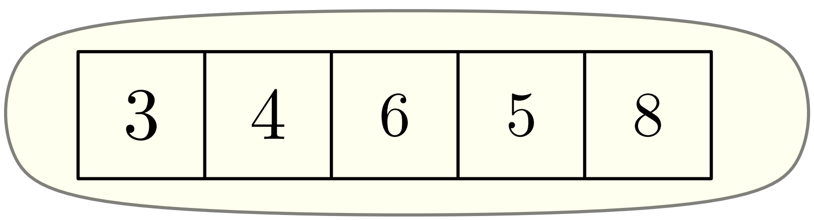


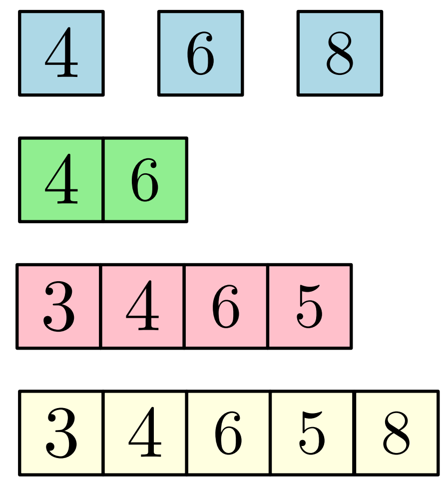


Even Subsequence

- **Problem**

Given a sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$, where $a_i \in \mathbb{N}_0^+$, return the number of contiguous non-empty subsequences B of A whose elements sum to an even number.

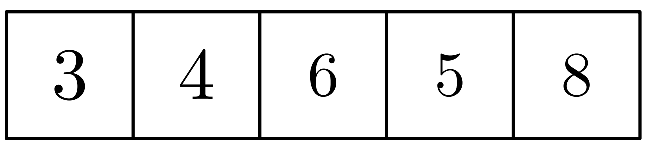


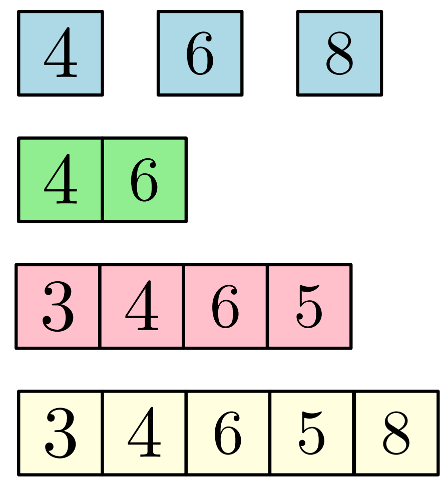


Even Subsequence

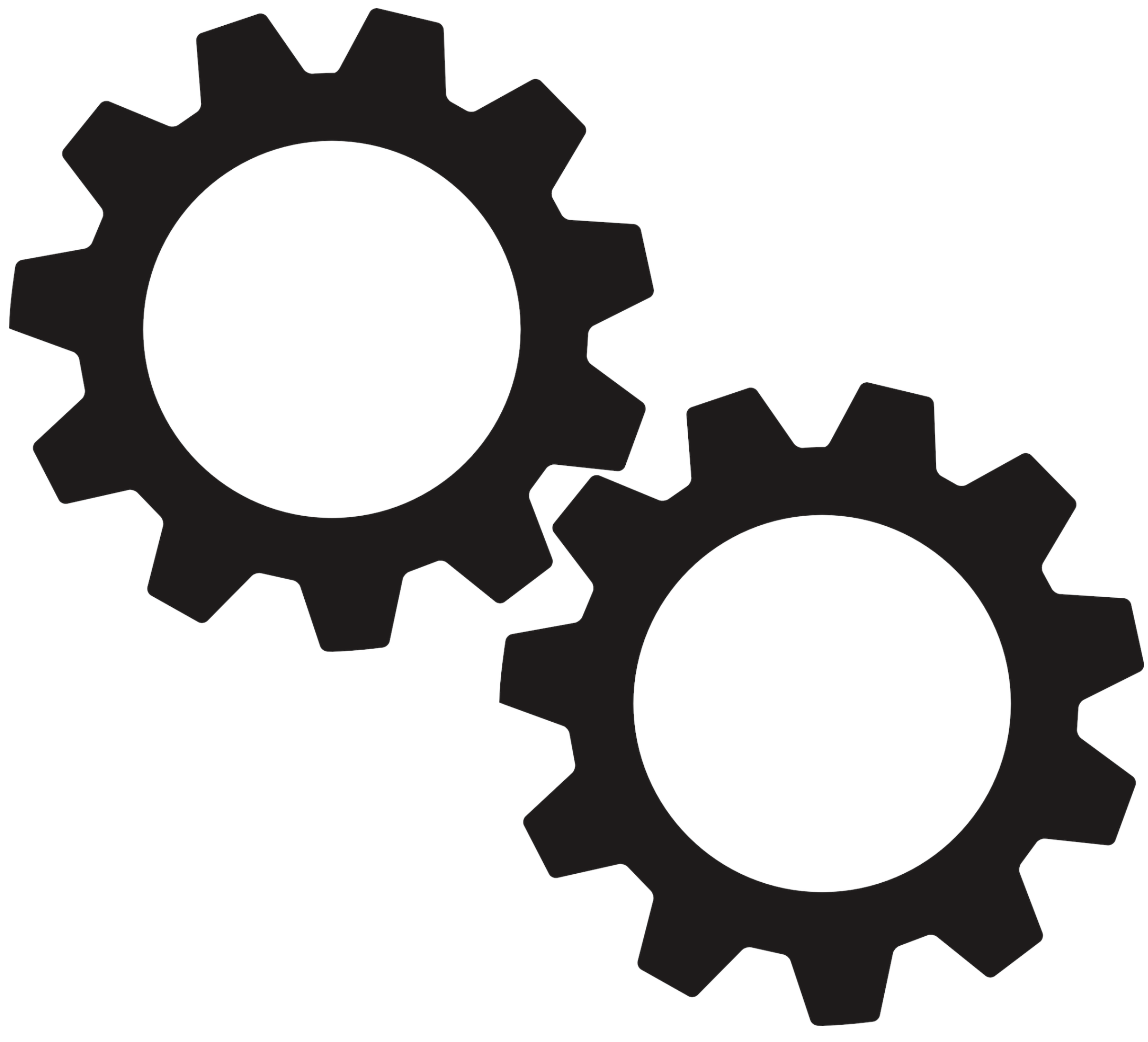
- **Problem**

Given a sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$, where $a_i \in \mathbb{N}_0^+$, return the number of contiguous non-empty subsequences B of A whose elements sum to an even number.





Answer: 6



Naive Solution

- For each pair of integers (i, j) with $i \leq j$
  - Check whether $\sum_{k=i}^{j} a_k$ is even

Naive Solution

- For each pair of integers (i, j) with $i \leq j$
  - Check whether $\sum_{k=i}^{j} a_k$ is even

```
unsigned int result = 0;
for(unsigned int i=0; i<A.size(); i++)
    for(unsigned int j=i; j<A.size(); j++)
        if(std::accumulate(A.begin()+i, A.begin()+j+1, 0)
                                                      %2 == 0)
            result++;

return result;
```

Naive Solution

- For each pair of integers (i, j) with $i \leq j$ $O(n^2)$
    - Check whether $\sum_{k=i}^{j} a_k$ is even $O(n)$

```
unsigned int result = 0;
for(unsigned int i=0; i<A.size(); i++)
    for(unsigned int j=i; j<A.size(); j++)
        if(std::accumulate(A.begin()+i, A.begin()+j+1, 0)
                                                   %2 == 0)
            result++;

return result;
```

Time: $\Theta(n^3)$.

Naive Solution

- For each pair of integers (i, j) with $i \leq j$ $O(n^2)$
  - Check whether $\sum_{k=i}^{j} a_k$ is even $O(n)$

```
unsigned int result = 0;
for(unsigned int i=0; i<A.size(); i++)
    for(unsigned int j=i; j<A.size(); j++)
        if(std::accumulate(A.begin()+i, A.begin()+j+1, 0)
                                                     %2 == 0)
            result++;

return result;
```

Time: $\Theta(n^3)$.

Can we do better?

Partial Sums Technique

$O(n)$

Compute the partial sums $P_k = \sum_{i=1}^{k} a_k$ in a single sweep

$$P_0 = 0 \qquad\qquad P_k = P_{k-1} + a_k$$

Partial Sums Technique

$O(n)$

Compute the partial sums $P_k = \sum_{i=1}^{k} a_k$ in a single sweep

$$P_0 = 0 \qquad\qquad P_k = P_{k-1} + a_k$$

A		3	4	6	5	8
P	0	3	7	13	18	26
	0	1	2	3	4	5

Partial Sums Technique

$O(n)$

Compute the partial sums $P_k = \sum_{i=1}^{k} a_k$ in a single sweep

$$P_0 = 0 \qquad\qquad P_k = P_{k-1} + a_k$$

A		3	4	6	5	8
P	0	3	7	13	18	26
	0	1	2	3	4	5

Observation: $S(i,j) = \sum_{k=i}^{j} a_k = P_j - P_{i-1}$

Partial Sums Technique

$O(n)$

Compute the partial sums $P_k = \sum_{i=1}^{k} a_k$ in a single sweep

$$P_0 = 0 \qquad\qquad P_k = P_{k-1} + a_k$$

A		3	4	6	5	8
P	0	3	7	13	18	26
	0	1	2	3	4	5

Observation: $S(i,j) = \sum_{k=i}^{j} a_k = P_j - P_{i-1}$ constant time!

Partial Sums Technique

$O(n)$

Compute the partial sums $P_k = \sum_{i=1}^{k} a_k$ in a single sweep

$$P_0 = 0 \qquad\qquad P_k = P_{k-1} + a_k$$

$S(3,4)$

A		3	4	6	5	8
P	0	3	7	13	18	26
	0	1	2	3	4	5

Observation: $S(i,j) = \sum_{k=i}^{j} a_k = P_j - P_{i-1}$ constant time!

Partial Sums Technique

$O(n)$

Compute the partial sums $P_k = \sum_{i=1}^{k} a_k$ in a single sweep

$$P_0 = 0 \qquad\qquad P_k = P_{k-1} + a_k$$

$S(3,4)$

A		3	4	6	5	8
P	0	3	7	13	18	26
	0	1	2	3	4	5

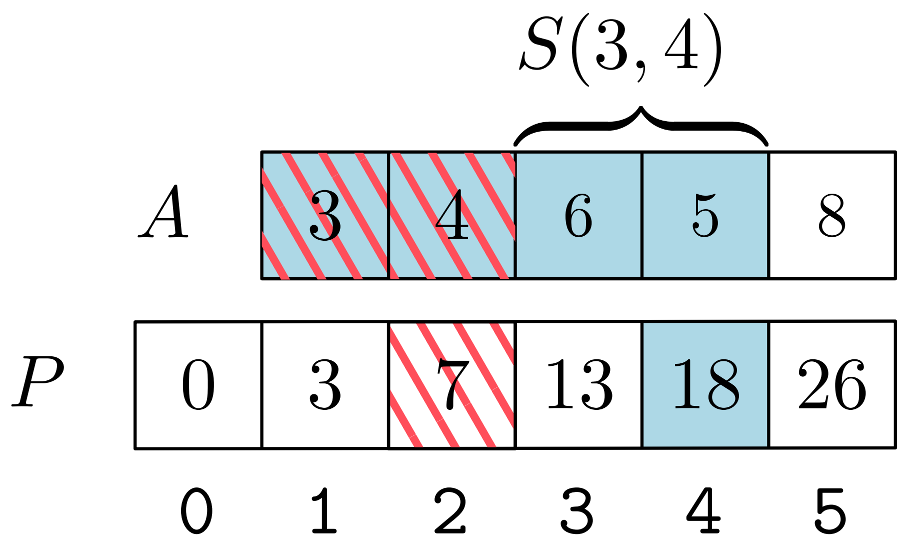
Observation: $S(i,j) = \sum_{k=i}^{j} a_k = P_j - P_{i-1}$ constant time!

Partial Sums Technique

$O(n)$

Compute the partial sums $P_k = \sum_{i=1}^{k} a_k$ in a single sweep

$$P_0 = 0 \qquad\qquad P_k = P_{k-1} + a_k$$



Observation: $S(i,j) = \sum_{k=i}^{j} a_k = P_j - P_{i-1}$ constant time!

A $O(n^2)$ algorithm

- Compute the vector P of prefix sums.
- For each pair of integers (i, j) with $i \leq j$
    - Check whether $S(i, j)$ is even

A $O(n^2)$ algorithm

- Compute the vector P of prefix sums. $O(n)$
- For each pair of integers (i, j) with $i \leq j$ $O(n^2)$
  - Check whether $S(i, j)$ is even $O(1)$

A $O(n^2)$ algorithm

- Compute the vector P of prefix sums. $O(n)$
- For each pair of integers (i, j) with $i \leq j$ $O(n^2)$
    - Check whether $S(i, j)$ is even $O(1)$

```
std::vector<int> P(A.size()+1);
std::partial_sum(A.begin(), A.end(), P.begin()+1);

unsigned int result = 0;
for(unsigned int i=1; i<=A.size(); i++)
    for(unsigned int j=i; j<=A.size(); j++)
        if((P[j] - P[i-1])%2 == 0)
            result++;

return result;
```

A $O(n^2)$ algorithm

- Compute the vector P of prefix sums. $O(n)$
- For each pair of integers (i, j) with $i \leq j$ $O(n^2)$
    - Check whether $S(i, j)$ is even $O(1)$

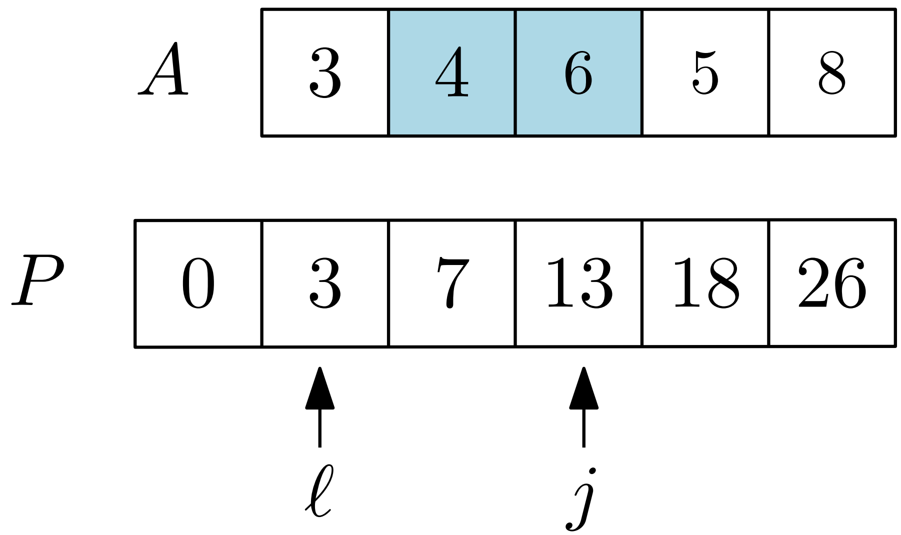
```
std::vector<int> P(A.size()+1);
std::partial_sum(A.begin(), A.end(), P.begin()+1);

unsigned int result = 0;
for(unsigned int i=1; i<=A.size(); i++)
    for(unsigned int j=i; j<=A.size(); j++)
        if((P[j] - P[i-1])%2 == 0)
            result++;

return result;
```

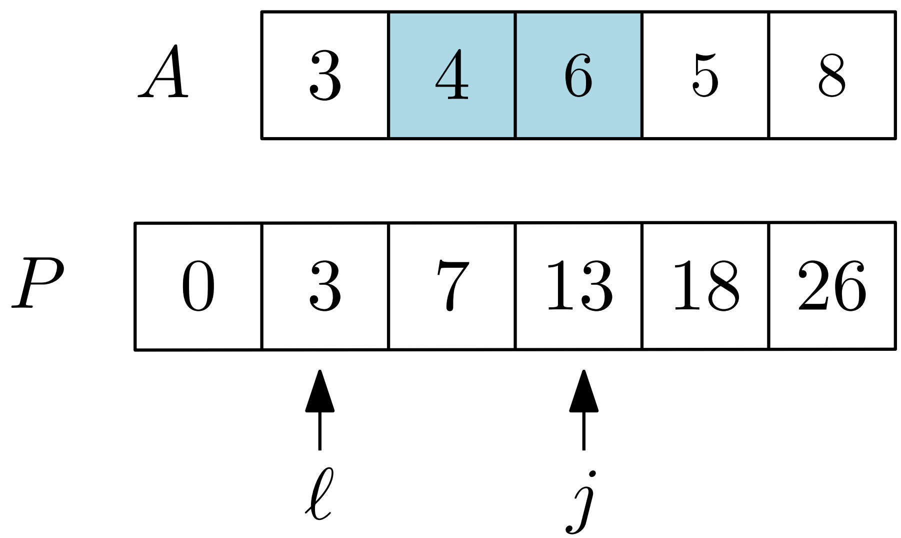
Can we do better?

A Linear-Time Solution



Observation 1: Each pair ℓ, j with $\ell < j$ for which $P_j - P_\ell$ is even contributes 1 to the result.

A Linear-Time Solution



Observation 1: Each pair ℓ, j with $\ell < j$ for which $P_j - P_\ell$ is even contributes 1 to the result.

Observation 2: $P_j - P_\ell$ is even iff P_ℓ and P_j are either both even, or both odd

A Linear-Time Solution

A		3	4	6	5	8
P	0	3	7	13	18	26
		ℓ		j		

Observation 1: Each pair ℓ, j with $\ell < j$ for which $P_j - P_\ell$ is even contributes 1 to the result.

Observation 2: $P_j - P_\ell$ is even iff P_ℓ and P_j are either both even, or both odd

Let E and F be the number of even and odd values is P, respectively.

Number of even pairs in P: $\binom{E}{2} = E(E-1)/2$.

Number of odd pairs in P: $\binom{F}{2} = F(F-1)/2$.

A Linear-Time Solution

- Compute the vector P of prefix sums $O(n)$
- Compute F from P $O(n)$
- Return $F(F-1)/2 + \overbrace{(n+1-F)}^{E}(n-F)/2$ $O(1)$

A Linear-Time Solution

- Compute the vector P of prefix sums $O(n)$
- Compute F from P $O(n)$
- Return $F(F-1)/2 + \overbrace{(n+1-F)}^{E}(n-F)/2$ $O(1)$

```
std::partial_sum(A.begin(), A.end(), A.begin());
int odd = std::count_if(A.begin(), A.end(),
                                        [](int x) {return x%2;});

return odd*(odd-1)/2 + (A.size()-odd)*(A.size()+1-odd)/2;
```

A Linear-Time Solution

- Compute the vector P of prefix sums $O(n)$
- Compute F from P $O(n)$
- Return $F(F-1)/2 + \overbrace{(n+1-F)}^{E}(n-F)/2$ $O(1)$

```
std::partial_sum(A.begin(), A.end(), A.begin());
int odd = std::count_if(A.begin(), A.end(),
                        [](int x) {return x%2;});

return odd*(odd-1)/2 + (A.size()-odd)*(A.size()+1-odd)/2;
```

$O(n)$ time $O(n)$ space

A Linear-Time Solution

- Compute the vector P of prefix sums $O(n)$
- Compute F from P $O(n)$
- Return $F(F-1)/2 + \overbrace{(n+1-F)}^{E}(n-F)/2$ $O(1)$

```
std::partial_sum(A.begin(), A.end(), A.begin());
int odd = std::count_if(A.begin(), A.end(),
                                 [](int x) {return x%2;});

return odd*(odd-1)/2 + (A.size()-odd)*(A.size()+1-odd)/2;
```

$O(n)$ time $O(n)$ space

Can we do better?

A Linear-Time Solution

- Compute the vector P of prefix sums $O(n)$
- Compute F from P $O(n)$
- Return $F(F-1)/2 + \overbrace{(n+1-F)}^{E}(n-F)/2$ $O(1)$

```
std::partial_sum(A.begin(), A.end(), A.begin());
int odd = std::count_if(A.begin(), A.end(),
                                 [](int x) {return x%2;});

return odd*(odd-1)/2 + (A.size()-odd)*(A.size()+1-odd)/2;
```

$O(n)$ time $O(n)$ space

Can we do better?

Observation: Each partial sum is used only once.

A Linear-Time Solution

- Scan the input and keep track of the current sum $O(n)$
- Keep track of F $O(n)$
- Return $F(F-1)/2 + \overbrace{(n+1-F)}^{E}(n-F)/2$ $O(1)$

```
unsigned int n;
std::cin >> n;

int x, odd=0, prefix_sum=0;
for(unsigned int i=0; i<n; i++)
{
    std::cin >> x;
    prefix_sum += x;
    odd += prefix_sum%2;
}

std::cout << odd*(odd-1)/2 + (n-odd)*(n+1-odd)/2 << "\n";
```

A Linear-Time Solution

- Scan the input and keep track of the current sum $O(n)$
- Keep track of F $O(n)$
- Return $F(F-1)/2 + \overbrace{(n+1-F)}^{E}(n-F)/2$ $O(1)$

```
unsigned int n;
std::cin >> n;

int x, odd=0, prefix_sum=0;
for(unsigned int i=0; i<n; i++)
{
    std::cin >> x;
    prefix_sum += x;
    odd += prefix_sum%2;
}

std::cout << odd*(odd-1)/2 + (n-odd)*(n+1-odd)/2 << "\n";
```

Streaming algorithm. $O(n)$ time $O(1)$ space