Even Subsequence

Even Subsequence

- **Problem**

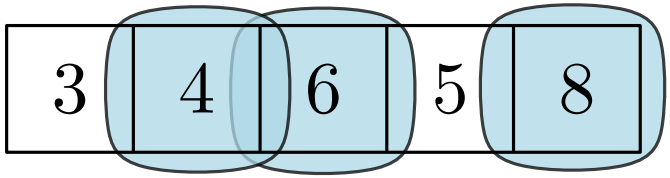
Given a sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$, where $a_i \in \mathbb{N}_0^+$, return the number of contiguous non-empty subsequences B of A whose elements sum to an even number.

3	4	6	5	8

Even Subsequence

- **Problem**

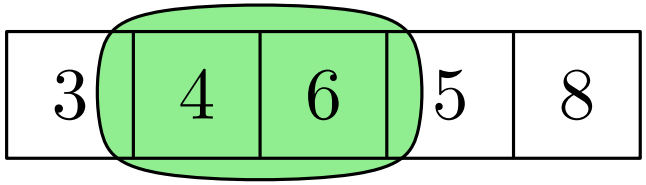
Given a sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$, where $a_i \in \mathbb{N}_0^+$, return the number of contiguous non-empty subsequences B of A whose elements sum to an even number.

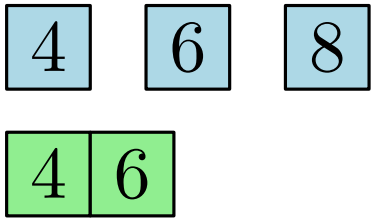


Even Subsequence

- **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.

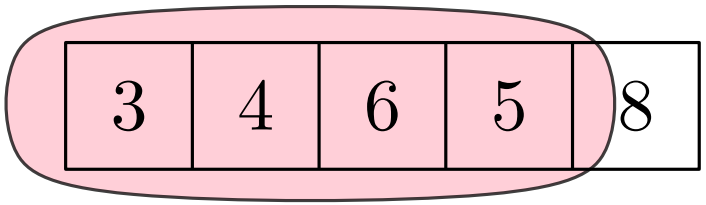


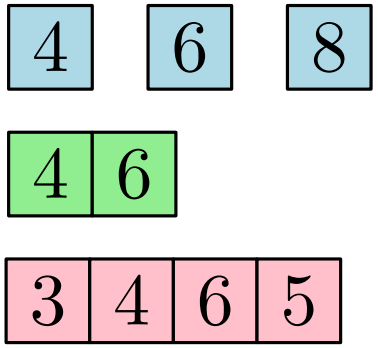


Even Subsequence

- **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.

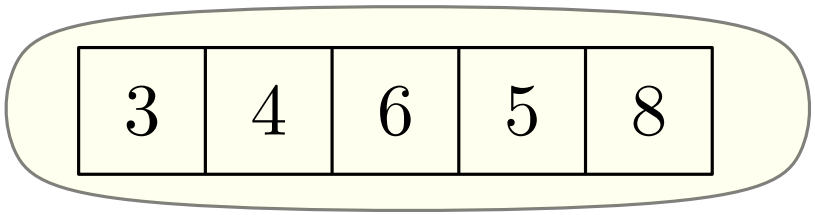


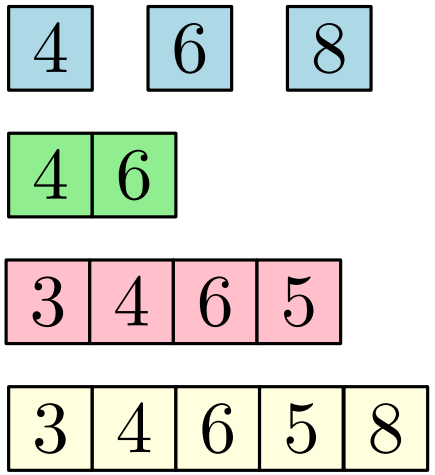


Even Subsequence

- **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.

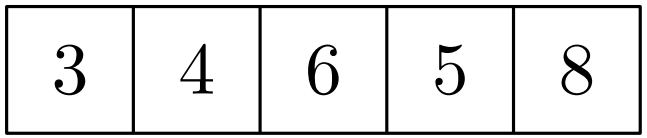


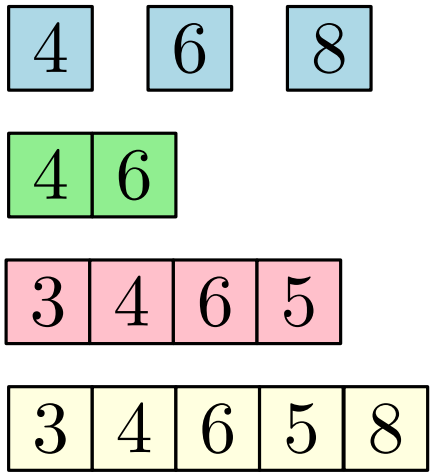


Even Subsequence

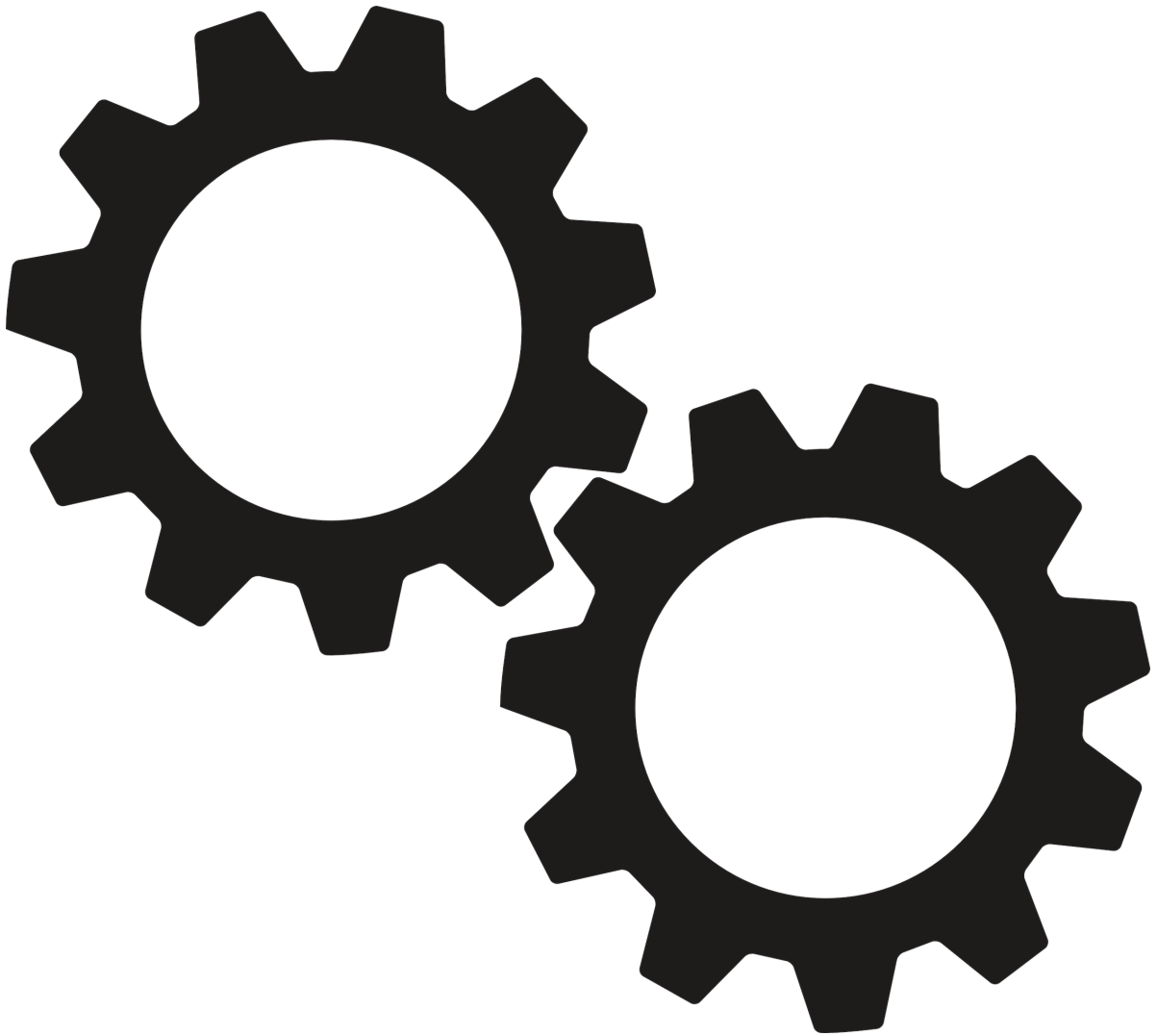
- **Problem**

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

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

Partial Sums Technique

$O(n)$

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

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

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

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

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

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

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

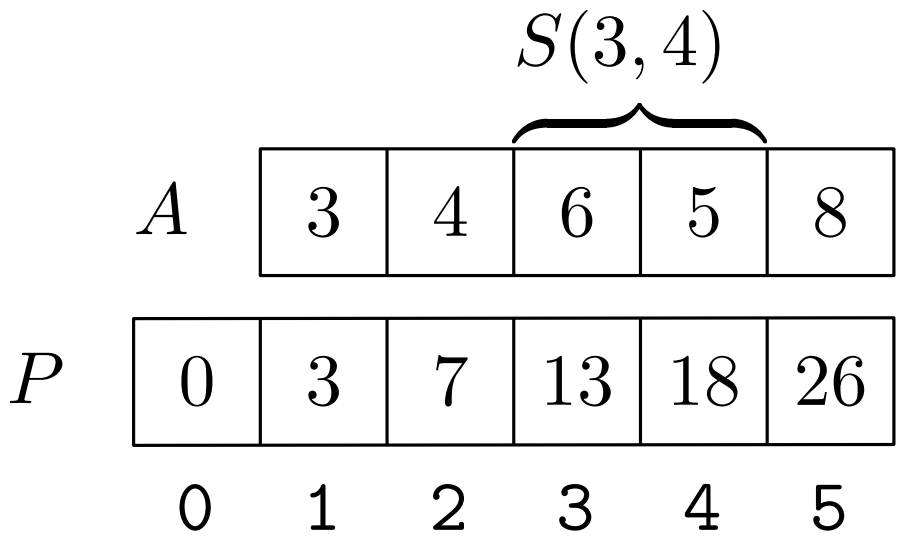
Observation: $S(i,j) = \sum_{k=i}^{j} a_k = P_j - P_{i-1}$ constant time!

Partial Sums Technique

$O(n)$

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

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



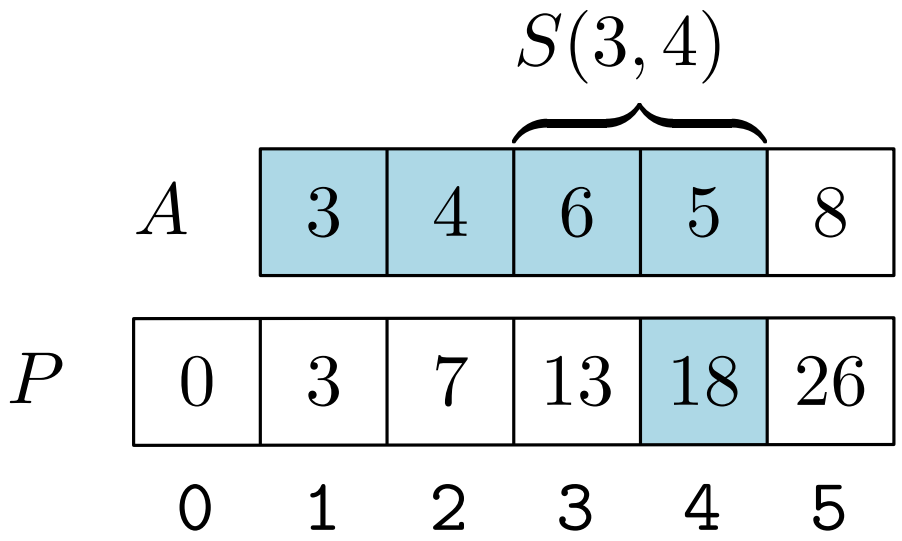
Observation: $S(i,j) = \sum_{k=i}^{j} a_k = P_j - P_{i-1}$ constant time!

Partial Sums Technique

$O(n)$

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

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



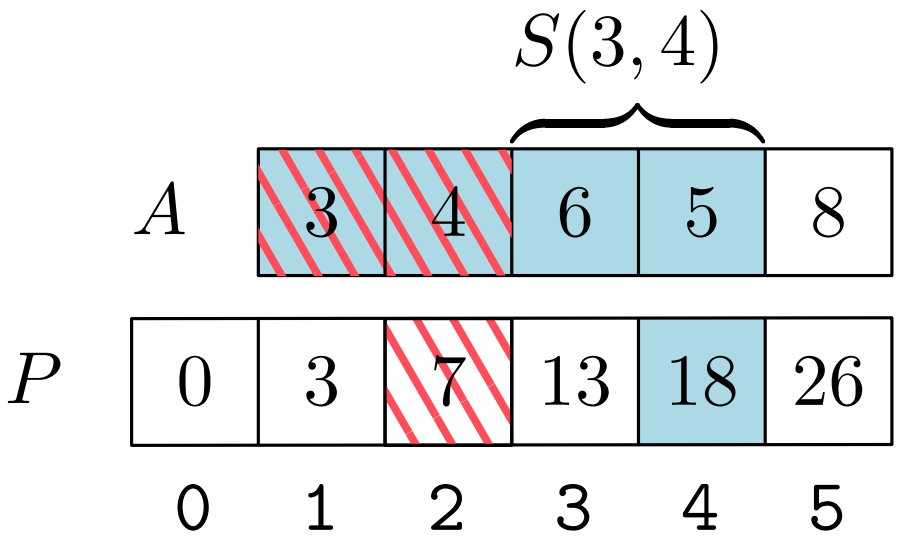
Observation: $S(i,j) = \sum_{k=i}^{j} a_k = P_j - P_{i-1}$ constant time!

Partial Sums Technique

$O(n)$

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

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



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 $O(1)$

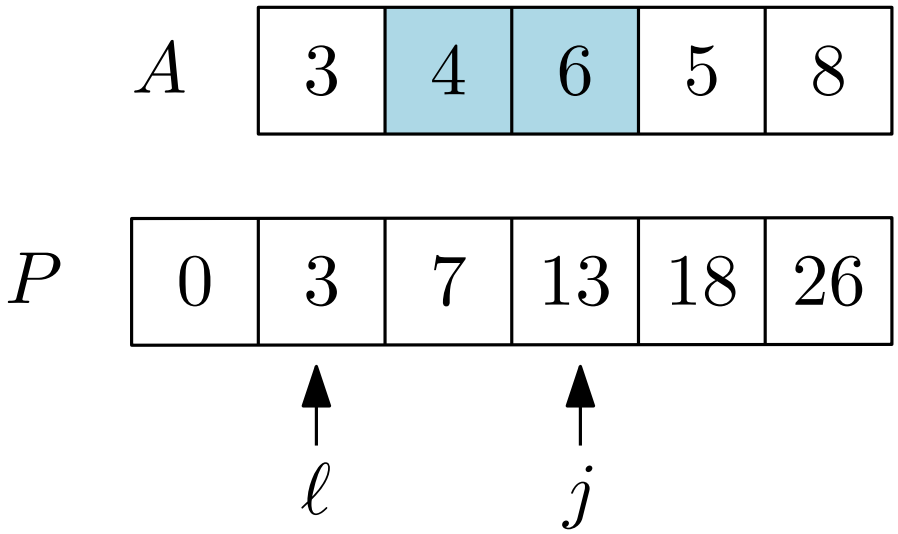
```
std::vector<int> P(A.size()+1);
std::partial_sum(A.begin(), A.end(), P.begin()+1);

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;
```

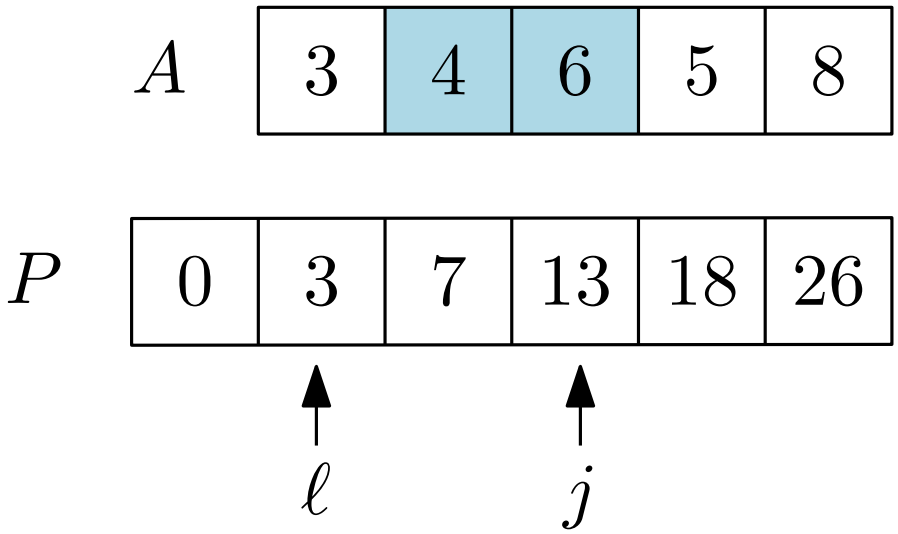
Can we do better?

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.

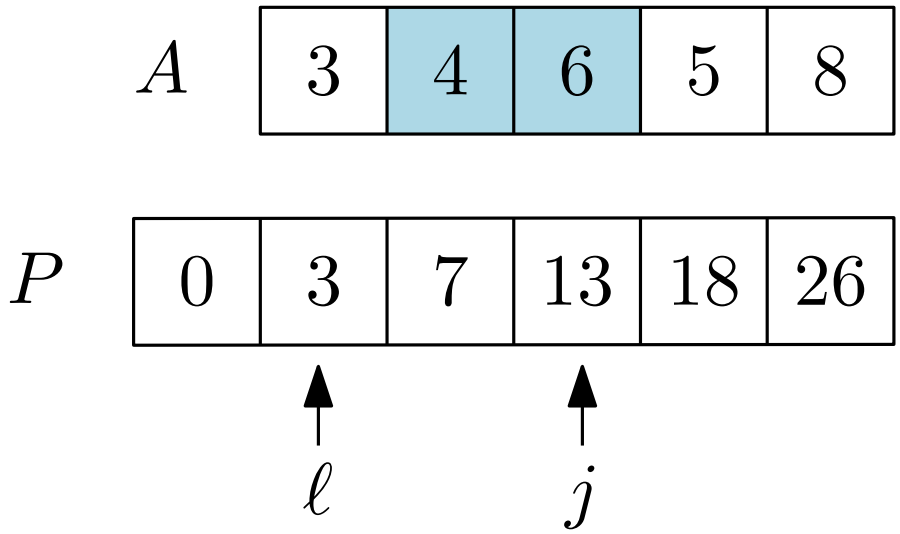
A Linear-Time Solution



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



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";
```

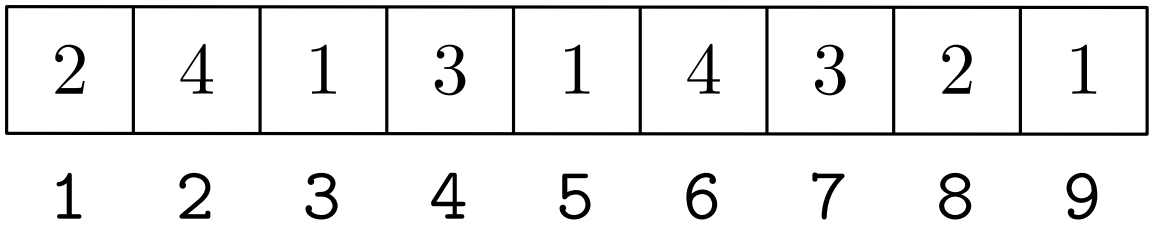
Streaming algorithm. $O(n)$ time $O(1)$ space

Balanced Array

Balanced Array

- **Problem**

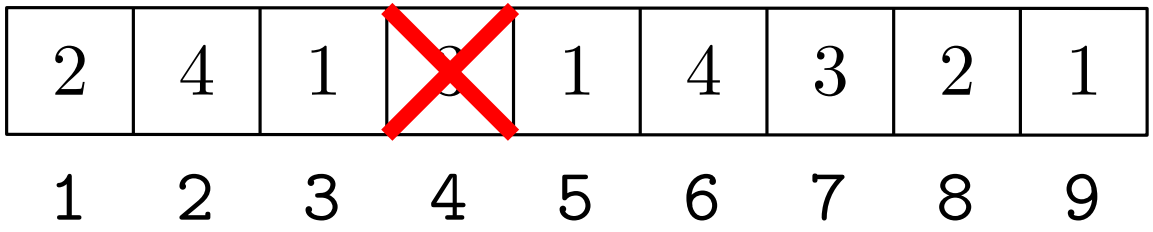
You are given an array of n integers and you need to delete exactly one element



Balanced Array

- **Problem**

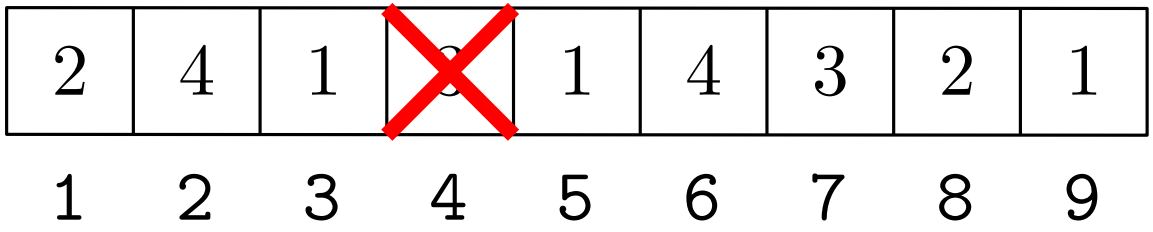
You are given an array of n integers and you need to delete exactly one element



Balanced Array

- **Problem**

You are given an array of n integers and you need to delete exactly one element



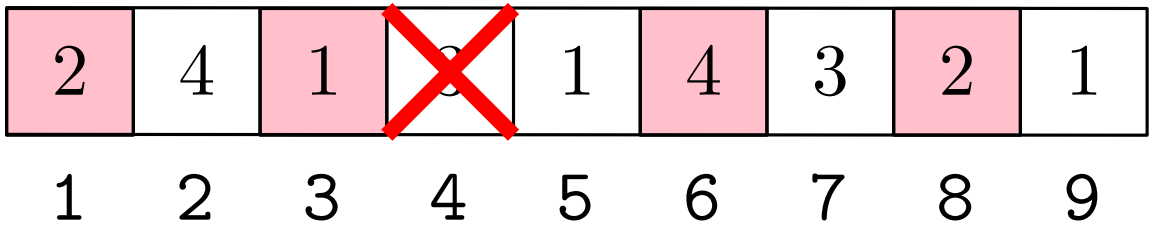
The resulting array is **balanced** if, **after the deletion**, the sum of the elements in even positions is equal to the sum of the elements in odd positions

Find the number of choices for the element to delete that result in a balanced array.

Balanced Array

- **Problem**

You are given an array of n integers and you need to delete exactly one element



The resulting array is **balanced** if, **after the deletion**, the sum of the elements in even positions is equal to the sum of the elements in odd positions

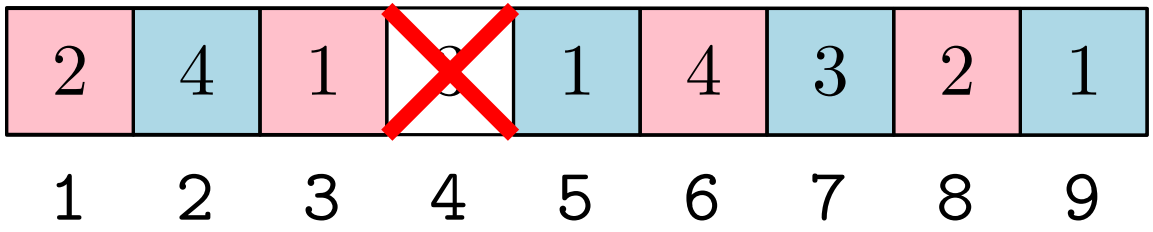
Find the number of choices for the element to delete that result in a balanced array.

Odd sum: 9

Balanced Array

- **Problem**

You are given an array of n integers and you need to delete exactly one element



The resulting array is **balanced** if, **after the deletion**, the sum of the elements in even positions is equal to the sum of the elements in odd positions

Find the number of choices for the element to delete that result in a balanced array.

Odd sum: 9

Even sum: 9

Naive Solution

- Result $= 0$
- For each index i that can be deleted:
  - Odd_Sum $= 0$, Even_Sum $= 0$
  - For each odd $j < i$: Odd_Sum $+=$ a_j
  - For each even $j > i$: Odd_Sum $+=$ a_j
  - For each even $j < i$: Even_Sum $+=$ a_j
  - For each odd $j > i$: Even_Sum $+=$ a_j
  - If Odd_Sum $==$ Even_Sum: Result++;
- Return Result

Naive Solution

- Result $= 0$
- For each index i that can be deleted: $O(n)$
  - Odd_Sum $= 0$, Even_Sum $= 0$
  - For each odd $j < i$: Odd_Sum $+= a_j$
  - For each even $j > i$: Odd_Sum $+= a_j$

  } $O(n)$
  - For each even $j < i$: Even_Sum $+= a_j$
  - For each odd $j > i$: Even_Sum $+= a_j$

  } $O(n)$
  - If Odd_Sum $==$ Even_Sum: Result++;
- Return Result

Time: $\Theta(n^2)$.

A Possible Implementation

```
unsigned int result = 0;
for(unsigned int i=1; i<=A.size(); i++)
{
    int even_sum=0, odd_sum=0;
    for(unsigned int j=1; j<i; j+=2) odd_sum+=A[j];
    for(unsigned int j=i + 2 - i%2; j<i; j+=2) odd_sum+=A[j];

    for(unsigned int j=2; j<i; j+=2) even_sum+=A[j];
    for(unsigned int j=i + 1 + i%2; j<i; j+=2) even_sum+=A[j];

    result += (odd_sum == even_sum)
}

return result;
```

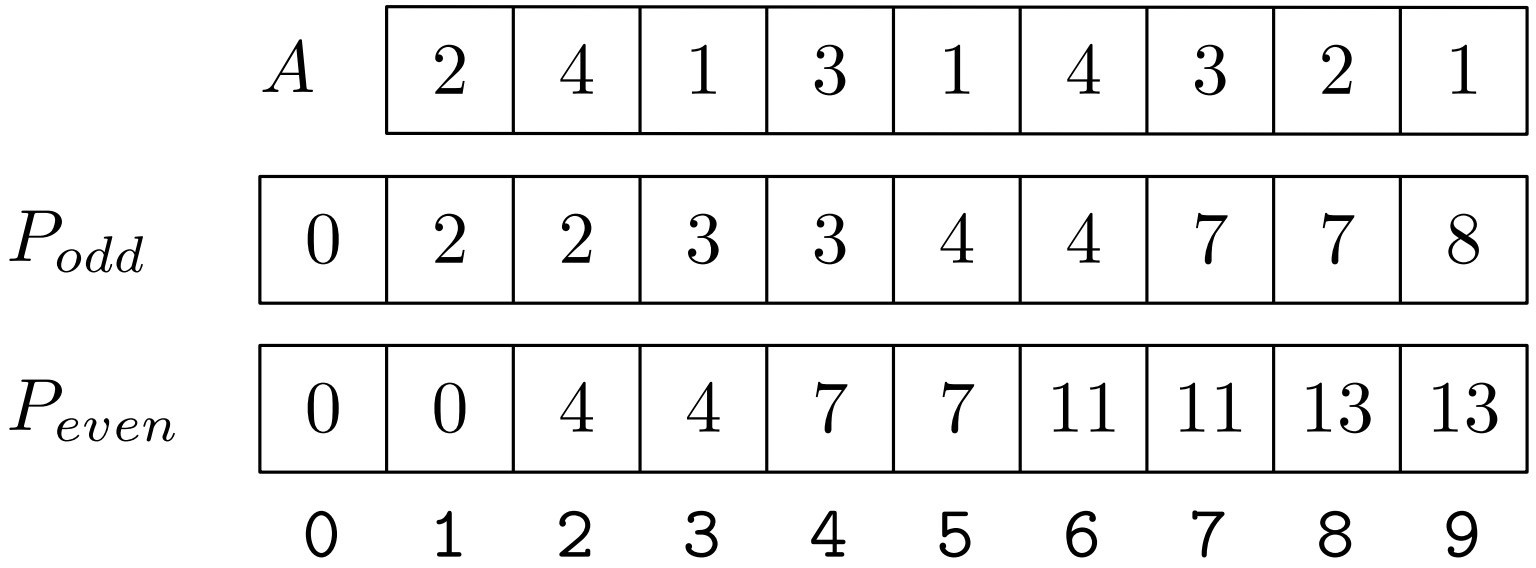
Prefix Sums:

Idea:

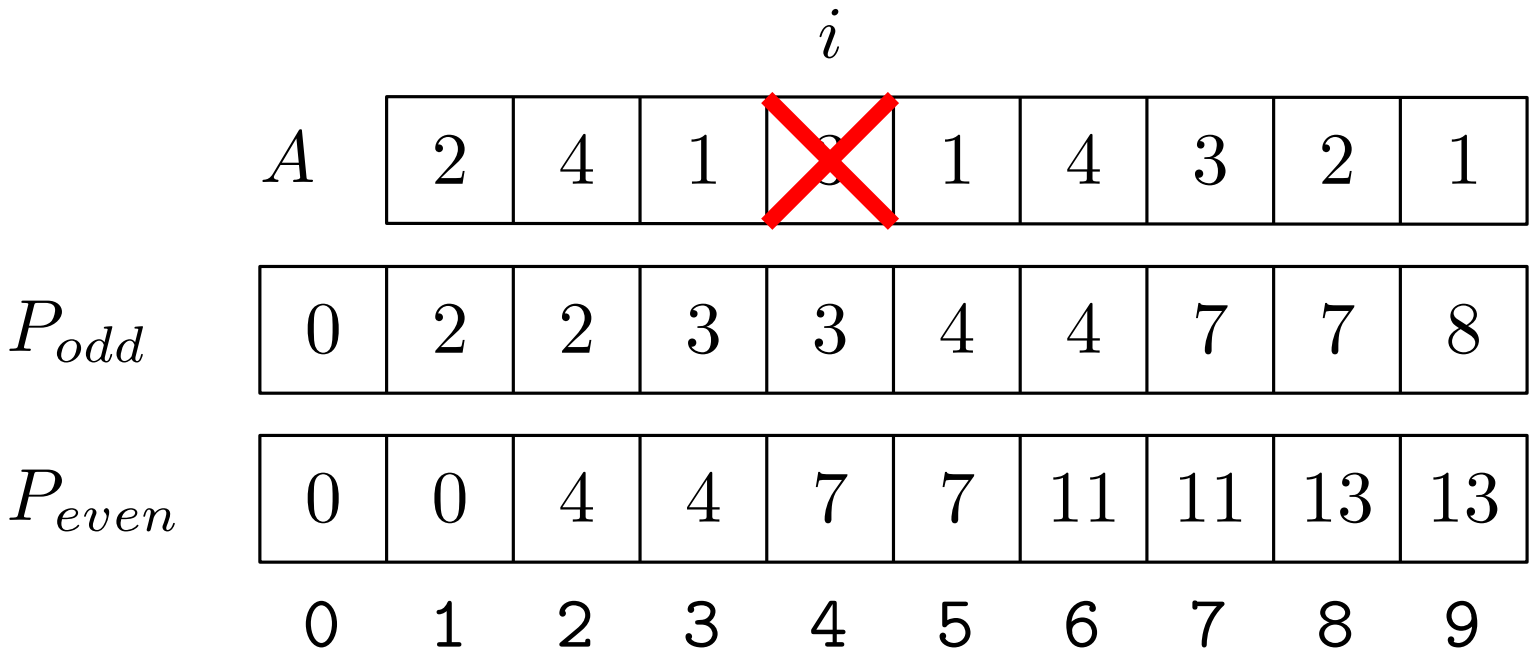
- pre-compute all prefix sums restricted to all elements in odd positions (in the original array!)
- pre-compute all prefix sums restricted to all elements in even positions (in the original array!)

A	2	4	1	3	1	4	3	2	1
P_{odd}	0	2	2	3	3	4	4	7	7	8
P_{even}	0	0	4	4	7	7	11	11	13	13
	0	1	2	3	4	5	6	7	8	9

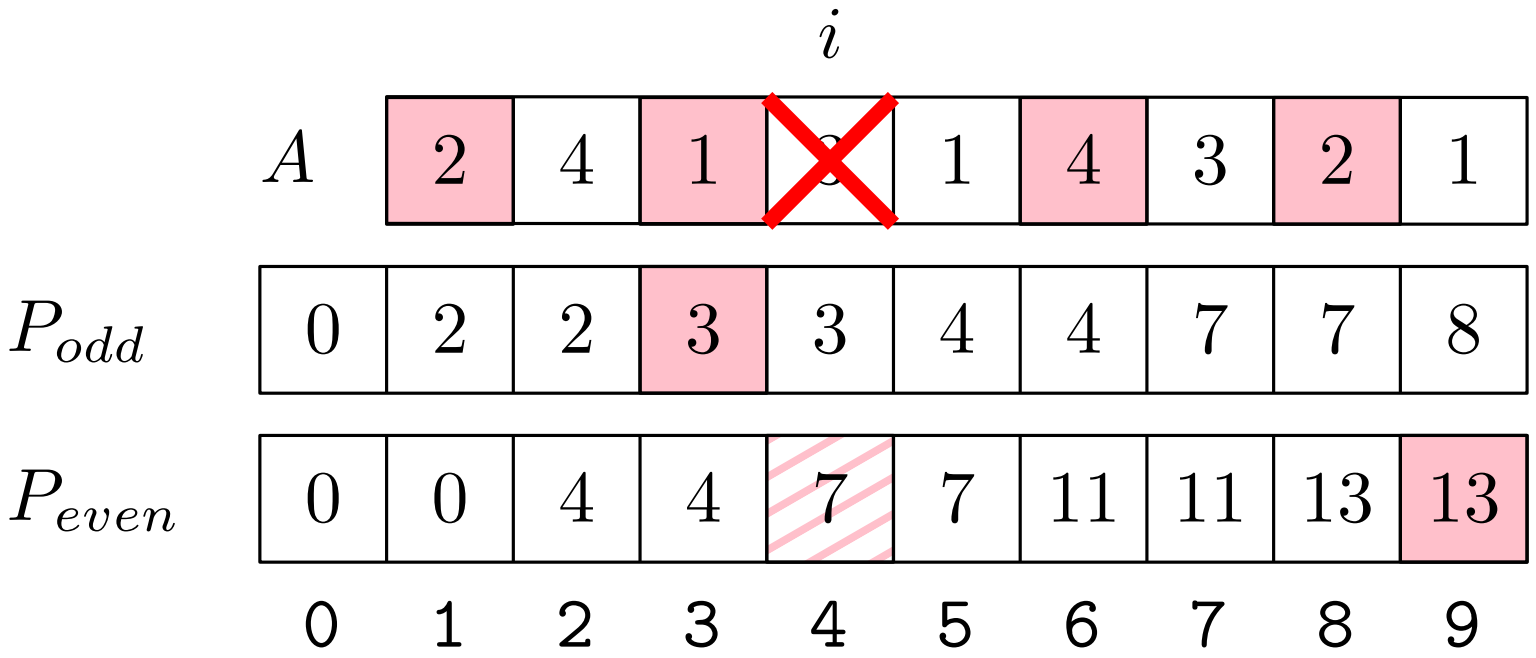
Prefix Sums:


A
2 4 1 3 1 4 3 2 1
P_{odd}
0 2 2 3 3 4 4 7 7 8
P_{even}
0 0 4 4 7 7 11 11 13 13
0 1 2 3 4 5 6 7 8 9

Prefix Sums:

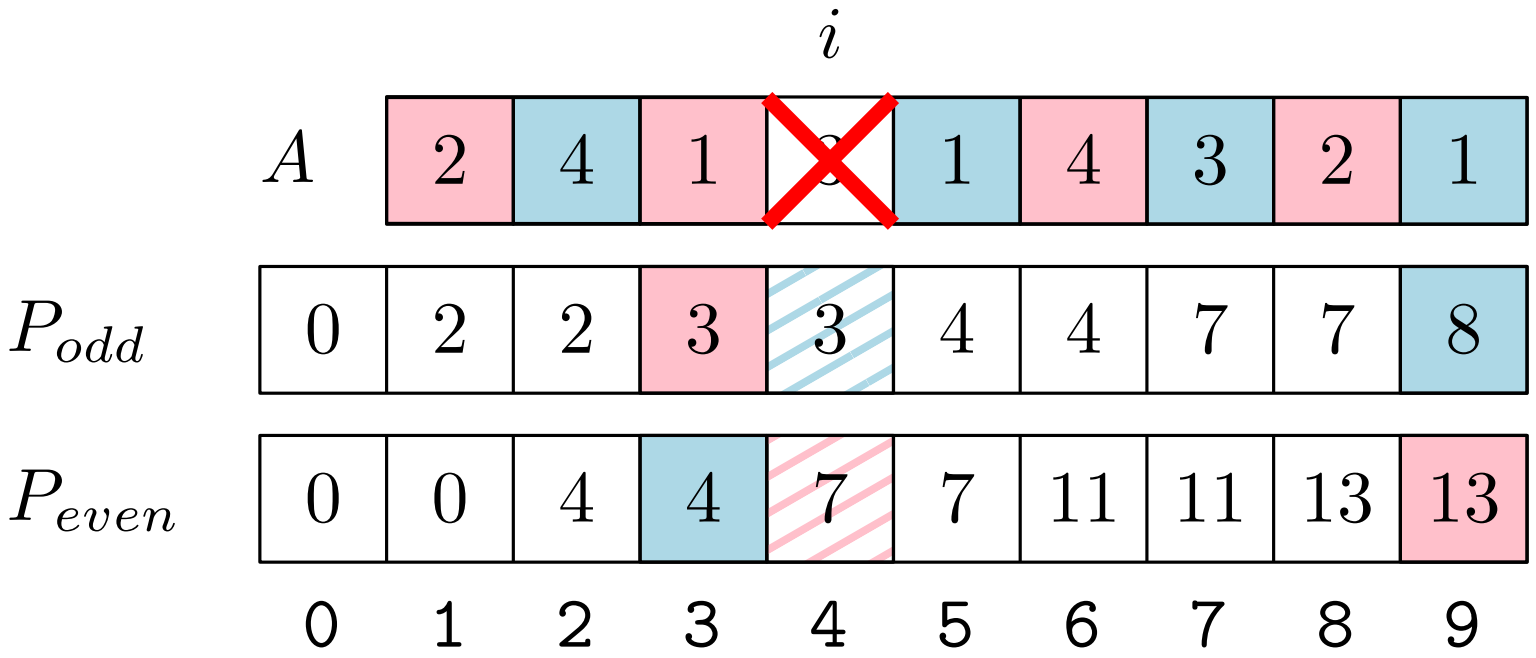

i
A
2 4 1 3 1 4 3 2 1
P_{odd}
0 2 2 3 3 4 4 7 7 8
P_{even}
0 0 4 4 7 7 11 11 13 13
0 1 2 3 4 5 6 7 8 9

Prefix Sums:



Odd_Sum: $= P_{odd}[i-1] + P_{even}[n] - P_{even}[i]$

Prefix Sums:



Odd_Sum: $= P_{odd}[i-1] + P_{even}[n] - P_{even}[i]$

Even_Sum: $= P_{even}[i-1] + P_{odd}[n] - P_{odd}[i]$

Prefix Sums:

- Compute the vectors P_{odd} and P_{even} of prefix sums — $O(n)$
- Result $= 0$
- For each index i that can be deleted: — $O(n)$
  - Odd_Sum $= P_{odd}[i-1] + (P_{even}[n] - P_{even}[i])$
  - Even_Sum $= P_{even}[i-1] + (P_{odd}[n] - P_{odd}[i])$

  } $O(1)$
  - If Odd_Sum == Even_Sum: Result++;
- Return Result

Time: $\Theta(n)$.

A Possible Implementation

```
std::vector<int> Peven(A.size()+1)
std::vector<int> Podd(A.size()+1);
for(int i=1; i<=A.size(); A++)
{
    Peven[i] = Peven[i-1] + (i%2)?0:A[i];
    Podd[i] = Podd[i-1] + (i%2)?A[i]:0;
}

unsigned int result = 0;
for(unsigned int i=1; i<=A.size(); i++)
{
    int even_sum = Peven[i-1] + Podd[n] - Podd[i];
    int odd_sum = Podd[i-1] + Peven[n] - Peven[i];
    result += (odd_sum == even_sum)
}

return result;
```