

브루트포스

김영균

1 브루트 포스

- 브루트 포스란?

우리말로 완전 탐색이다. 즉 모든 경우의 수를 탐색해보는 거다.

- ┌ 반복문을 이용한 완전 탐색
- └ 재귀를 이용한 완전 탐색

- 코테에서는 구현력과 적절한 자료구조를 사용할 수 있는지 테스트용으로 자주 출제된다.

따라서 자신의 문제를 많이 풀어서 구현력을 기르는 게 답이긴 하다.

- 이번 시간에는 반복문을 이용한 완탐 중 몇 가지 테크닉을 알려주려고 한다.

2 누적 합

아래의 배열에서 인덱스의 번호가 3번째 원소부터 6번째 원소까지 합을 구하려면 어떻게 해야 할까?

index	0	1	2	3	4	5	6	7
arr[i]	1	33	8	19	6	9	10	13

완전 탐색으로 for문을 돌리면 된다.

```
int answer = 0;  
for(int i = 3; i < 6; i++) answer += arr[i];
```

배열에 포함된 원소의 개수가 N개일 때 위 방법의 시간복잡도는 $O(N)$ 이다.

$O(N)$ 에서 $O(1)$ 로 줄이는 방법을 알아보자.

2 누적 합

새로운 배열 psum[i]을 아래와 같이 정의해보자.

$\text{psum}[i] = \text{psum}[i - 1] + \text{arr}[i]$ (단, i 가 0일때 $\text{psum}[i] = \text{arr}[i]$ 이다.)

index	0	1	2	3	4	5	6	7
arr[i]	1	33	8	19	6	9	10	13

$\text{psum}[0] = \text{arr}[0]$

psum[0]	1
---------	---

2 누적 합

새로운 배열 psum[i]을 아래와 같이 정의해보자.

$psum[i] = psum[i - 1] + arr[i]$ (단, i 가 0일때 $psum[i] = arr[i]$ 이다.)

index	0	1	2	3	4	5	6	7
arr[i]	1	33	8	19	6	9	10	13

$psum[0] = arr[0]$

psum[0]	1
psum[1]	1 + 33

$psum[1] = psum[0] + arr[1]$

2 누적 합

새로운 배열 psum[i]을 아래와 같이 정의해보자.

$psum[i] = psum[i - 1] + arr[i]$ (단, i 가 0일때 $psum[i] = arr[i]$ 이다.)

index	0	1	2	3	4	5	6	7
arr[i]	1	33	8	19	6	9	10	13

$psum[0] = arr[0]$

$psum[1] = psum[0] + arr[1]$

$psum[2] = psum[1] + arr[2]$

psum[0]	1
psum[1]	34
psum[2]	34 + 8

2 누적 합

새로운 배열 psum[i]을 아래와 같이 정의해보자.

$psum[i] = psum[i - 1] + arr[i]$ (단, i 가 0일때 $psum[i] = arr[i]$ 이다.)

index	0	1	2	3	4	5	6	7
arr[i]	1	33	8	19	6	9	10	13
psum[i]	1	34	42	61	67	76	86	99

아 ~ psum[i]는 0번부터 i번째 까지 arr[i]의 합이구나~!

그러면 psum배열을 이용해서 어떻게 3~6번째 까지의 합을 $O(1)$ 구할까?

psum의 정의 $psum[i] = psum[i - 1] + arr[i]$ 로부터 $arr[i] = psum[i] - psum[i - 1]$ 을 도출 할 수 있다.

그렇다면 3번째부터 6번째까지의 합 $\sum_{i=3}^{i=6} arr[i]$ 를 구해보자

2 누적 합

$$\text{psum}[i] = \text{psum}[i - 1] + \text{arr}[i]$$

index	0	1	2	3	4	5	6	7
arr[i]	1	33	8	19	6	9	10	13
psum[i]	1	34	42	61	67	76	86	99

시그마 식을 전개해보자.

$$\sum_{i=3}^6 \text{arr}[i] = \text{arr}[3] + \text{arr}[4] + \text{arr}[5] + \text{arr}[6]$$

$$\text{arr}[3] = \text{psum}[3] - \text{psum}[2]$$

$$\text{arr}[4] = \text{psum}[4] - \text{psum}[3]$$

$$\text{arr}[5] = \text{psum}[5] - \text{psum}[4]$$

$$\text{arr}[6] = \text{psum}[6] - \text{psum}[5]$$

+) _____

2 누적 합

$$\text{psum}[i] = \text{psum}[i - 1] + \text{arr}[i]$$

index	0	1	2	3	4	5	6	7
arr[i]	1	33	8	19	6	9	10	13
psum[i]	1	34	42	61	67	76	86	99

시그마 식을 전개해보자.

$$\sum_{i=3}^6 \text{arr}[i] = \text{arr}[3] + \text{arr}[4] + \text{arr}[5] + \text{arr}[6]$$

$$\text{arr}[3] = \cancel{\text{psum}[3]} - \text{psum}[2]$$

$$\text{arr}[4] = \cancel{\text{psum}[4]} - \cancel{\text{psum}[3]}$$

$$\text{arr}[5] = \cancel{\text{psum}[5]} - \cancel{\text{psum}[4]}$$

$$\text{arr}[6] = \text{psum}[6] - \cancel{\text{psum}[5]}$$

+))

$$= \text{psum}[6] - \text{psum}[2]$$

간단하게 일반화를 하면 배열 arr의 어떤 구간 i, j ($i \leq j$) 사이의 합은
 $\text{arr}[i] + \text{arr}[i + 1] + \dots + \text{arr}[j - 1] + \text{arr}[j] = \text{psum}[j] - \text{psum}[i - 1]$ 와 같다.

2 누적 합

간단하게 일반화를 하면 배열 arr의 어떤 구간 i, j ($i \leq j$) 사이의 합은 $arr[i] + arr[i + 1] + \dots + arr[j - 1] + arr[j] = psum[j] - psum[i - 1]$ 와 같다.

주의할 점) 만약 i 가 0이라면? $psum[i - 1]$ 은 잘못된 인덱스 접근이므로 i 가 0일 때는 $psum[i - 1]$ 은 0과 같다고 따로 처리해주어야한다.

```
int partial_sum(int* psum, int i, int j) {  
    if(i == 0) return psum[j];  
    return psum[j] - psum[i - 1];  
}
```

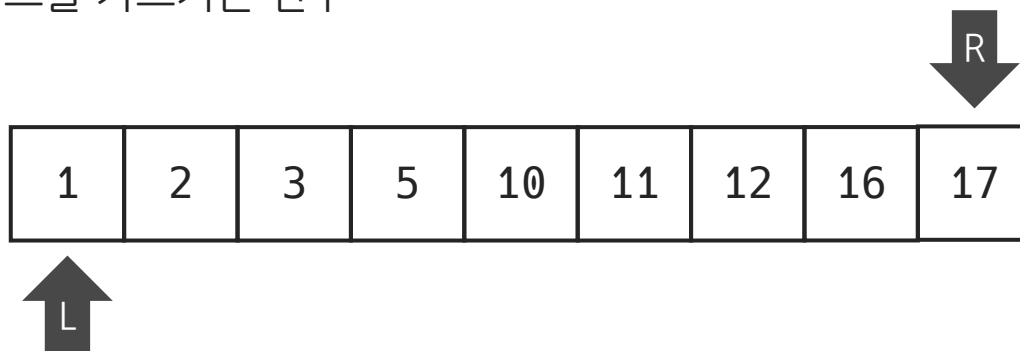
3 투 포인터

- 투 포인터란?

두 개의 포인터(커서)를 움직이면서 $O(N)$ 에 배열을 탐색하는 테크닉

- 포인터(커서)란?

C언어의 포인터가 아닌, 배열에 인덱스를 가르키는 변수



- 언제 사용?

대다수는 정렬된 배열에서 사용한다. 정렬된 배열에서 $O(N^2)$ 탐색 중 필요없는 연산을 버려서 $O(N)$ 으로 줄인다.

3 투 포인터

9개의 서로 다른 양의 정수 a_0, a_1, \dots, a_8 으로 이루어진 정렬된 수열이 있다. $a_i + a_j = 13$ ($0 \leq i < j \leq 8$)을 만족하는 (a_i, a_j) 쌍의 수를 구하는 프로그램을 작성하시오.

1	2	3	5	10	11	12	16	17
---	---	---	---	----	----	----	----	----



완전탐색으로 풀 경우 코드는 아래와 같고 $O(N^2)$ 이다.

```
int answer = 0;
for(int i = 0; i < 9; i++) {
    for(int j = i + 1; j < 9; j++) {
        if(a[i] + a[j] == 13) answer++;
    }
}
```

3 두 포인터

완전 탐색을 잠깐 진행해보자.

1. 인덱스 i, j 가 $i = 0, j = 6$ 일 때 $a[0] + a[6] = 13$ 이므로
정답의 개수를 증가시킨다. → 이후 j 가 1 증가된다

									
index	0	1	2	3	4	5	6	7	8
a[i]	1	2	3	5	10	11	12	16	17



3 투 포인터

완전 탐색을 잠깐 진행해보자.

- 1. 인덱스 i, j 가 $i = 0, j = 6$ 일 때 $a[0] + a[6] = 13$ 이므로 정답의 개수를 증가시킨다. → 이후 j 가 1 증가된다.
- 2. j 가 1 증가하였고 $a[0] + a[7] > 13$ 이 되었다.

여기서 필요 없는 연산을 찾을 수 있다.

$a[0] + a[7] > 13$ 이므로 정렬된 배열에선 $a[0] + a[8] > 13$ 임을 j 를 증가시키지 않고도 알 수 있다.
마찬가지로, $a[2] + a[6] > 13$ 이므로 $a[2] + a[7] > 13$ 임을 알 수 있다.
즉. $a[i] + a[j] > 13$ 이면 j 를 증가시킬 필요가 없다.

									
index	0	1	2	3	4	5	6	7	8
a[i]	1	2	3	5	10	11	12	16	17

3 투 포인터

두 개의 포인터를 양 끝을 가르키게 하자.

$a[0] + a[8] > 13$ 이므로 $right$ 을 감소시켜 $a[left] + a[right]$ 합을 줄여보자.

left	right	$a[left] + a[right]$	결과
0	8	18	right 감소

	left ↓								right ↓
index	0	1	2	3	4	5	6	7	8
$a[i]$	1	2	3	5	10	11	12	16	17

왜 $right$ 을 줄이는 가?

예시: $(0, 6) \rightarrow (1, 5) \rightarrow (2, 4)$

주어진 배열은 오름차순이므로 $left$ 가 증가할수록 $a[left] + a[right] = 13$ 을 만족시키는 $right$ 는 줄어든다.

따라서 초기 $left$ 에 대해서 어떤 k 가 $a[0] + a[k] > 13$ 이라면 이후 증가한 $left$ 들은 $a[k + 1], a[k + 2] \dots$ 은 탐색할 필요가 없다.

3 투 포인터

left = 0, right = 7이 되었다.

$a[0] + a[7] > 13$ 이므로, 마찬가지로 right을 줄이자.

left	right	$a[\text{left}] + a[\text{right}]$	결과
0	8	18	right 감소
0	7	17	right 감소

	left ↓				right ↓				
index	0	1	2	3	4	5	6	7	8
a[i]	1	2	3	5	10	11	12	16	17

3 투 포인터

left = 0, right = 6이 되었다. $a[0] + a[6] = 13$ 이므로 정답을 갱신하자.

이제 어떤 것을 옮겨야할까? 사실 상관없다.

앞으로 정답일 경우에는 right을 줄이자.

left	right	$a[\text{left}] + a[\text{right}]$	결과
0	8	18	right 감소
0	7	17	right 감소
0	6	13	정답 갱신 right 감소

	left						right		
	↓						↓		
index	0	1	2	3	4	5	6	7	8
a[i]	1	2	3	5	10	11	12	16	17

3 투 포인터

left = 0, right = 5이 되었다.

$arr[0] + arr[5] < 13$ 이다.

이 경우에는 left을 증가시켜 $a[left] + a[right]$ 값을 늘려보자.

left	right	$a[left] + a[right]$	결과
0	8	18	right 감소
0	7	17	right 감소
0	6	13	정답 갱신 right 감소
0	5	12	left 증가

left
↓

right
↓

index	0	1	2	3	4	5	6	7	8
a[i]	1	2	3	5	10	11	12	16	17

3 투 포인터

left = 1, right = 5이 되었다.

arr[1] + arr[5] = 13 이므로 정답을 갱신하고 right을 줄이자.

left	right	a[left] + a[right]	결과
0	8	18	right 감소
0	7	17	right 감소
0	6	13	정답 갱신 right 감소
0	5	12	left 증가
1	5	13	정답 갱신 right 감소

left

right

↓

↓

index	0	1	2	3	4	5	6	7	8
a[i]	1	2	3	5	10	11	12	16	17

3 투 포인터

left = 1, right = 4이 되었다.
arr[1] + arr[4] < 13 이므로 left을 늘리자.

left	right	a[left] + a[right]	결과
0	8	18	right 감소
0	7	17	right 감소
0	6	13	정답 갱신 right 감소
0	5	12	left 증가
1	5	13	정답 갱신 right 감소
1	4	12	left 증가

	left				right				
	↓				↓				
index	0	1	2	3	4	5	6	7	8
a[i]	1	2	3	5	10	11	12	16	17

3 투 포인터

left = 2, right = 4이 되었다.
arr[2] + arr[4] = 13 이므로 right을 줄이자.

left	right	a[left] + a[right]	결과
0	8	18	right 감소
0	7	17	right 감소
0	6	13	정답 갱신 right 감소
0	5	12	left 증가
1	5	13	정답 갱신 right 감소
1	4	12	left 증가
2	4	13	정답 갱신 right 감소

			left		right				
			↓		↓				
index	0	1	2	3	4	5	6	7	8
a[i]	1	2	3	5	10	11	12	16	17

3 투 포인터

left = 2, right = 3이 되었다.
arr[2] + arr[3] < 13 이므로 right을 줄이자.

left	right	a[left] + a[right]	결과
0	8	18	right 감소
0	7	17	right 감소
0	6	13	정답 갱신 right 감소
0	5	12	left 증가
1	5	13	정답 갱신 right 감소
1	4	12	left 증가
2	4	13	정답 갱신 right 감소
2	3	8	left 증가

			left	right					
			↓	↓					
index	0	1	2	3	4	5	6	7	8
a[i]	1	2	3	5	10	11	12	16	17

3 투 포인터

9개의 서로 다른 양의 정수 a_1, a_2, \dots, a_n 으로 이루어진 정렬된 수열이 있다. $a_i + a_j = 13$ ($1 \leq i < j \leq n$)을 만족하는 (a_i, a_j) 쌍의 수를 구하는 프로그램을 작성하시오.

left = 3, right = 3이 되었다.

문제의 조건이 $i < j$ 이므로 탐색과정이 종료된다.

left	right	$a[\text{left}] + a[\text{right}]$	결과
0	8	18	right 감소
0	7	17	right 감소
0	6	13	정답 갱신 right 감소
0	5	12	left 증가
1	5	13	정답 갱신 right 감소
1	4	12	left 증가
2	4	13	정답 갱신 right 감소
2	3	8	left 증가
3	3		종료

left
right
↓

index	0	1	2	3	4	5	6	7	8
$a[i]$	1	2	3	5	10	11	12	16	17

3 투 포인터

- 시간복잡도 분석

left와 right가 양쪽에서 다가오는데 중간에 만나면 탐색이 멈추므로 left와 right가 움직인 거리는 총 배열의 길이와 같다. 따라서 $O(N)$

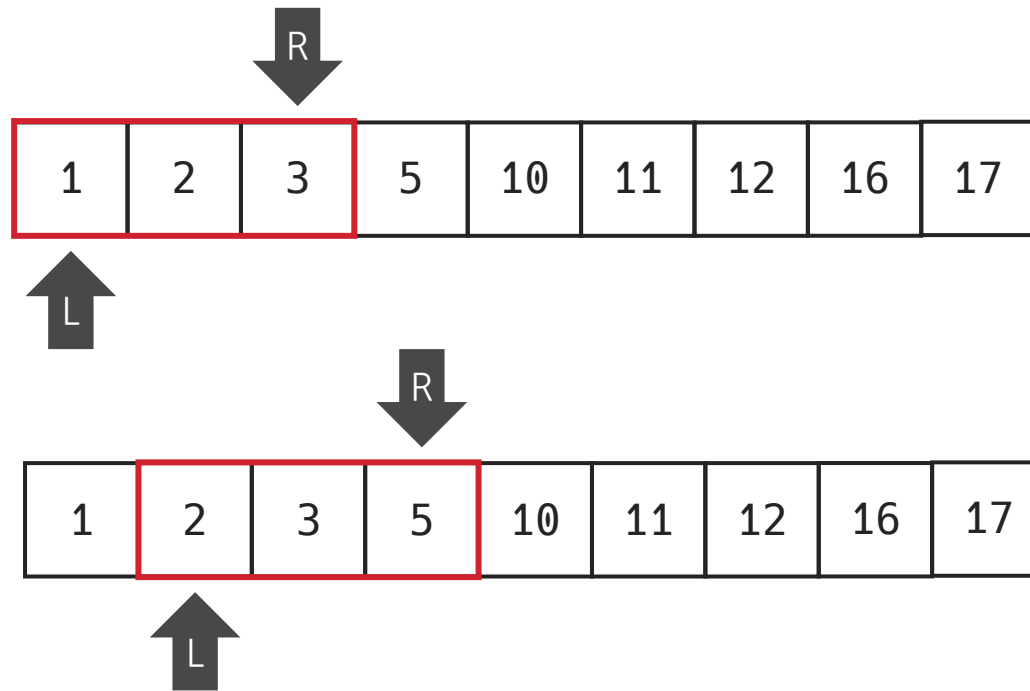
```
int left = 0, right = n - 1, answer = 0;
while(left < right) {
    if(a[left] + a[right] > 13) right--;
    else if(a[left] + a[right] < 13) left++;
    else { answer++; right--; }
}
```

- 사실 left와 right가 양쪽에서 다가오는 것 뿐만 아니라 left와 right가 한 쪽에서 같이 시작하는 경우도 있다.
어떻게 구분하냐? 경험...^^

4 슬라이딩 윈도우

- 슬라이딩 윈도우란?

고정된 길이의 두 개의 포인터(커서)를 같이 움직이면서 $O(N)$ 에 배열을 탐색하는テクニック



윈도우 크기가 3인 경우

4 슬라이딩 윈도우

7개의 a_0, a_1, \dots, a_6 으로 이루어진 수열이 있다. 이 수열에서 연속된 3개의 값의 합 중 최댓값을 구하는 프로그램을 작성하시오.

3	5	1	7	2	4	1
---	---	---	---	---	---	---

완전탐색으로 풀 경우 코드는 아래와 같다.

시간복잡도는 N 개의 수열에서 연속된 M 개의 값 중 최댓값을 찾는 경우라면 $O(NM)$ 이다.

```
int answer = 0;
for(int i = 0; i <= n - m; i++) {
    int sum = 0;
    for(int j = 0; j < m; j++) sum += a[i + j];
    answer = max(answer, sum);
}
```

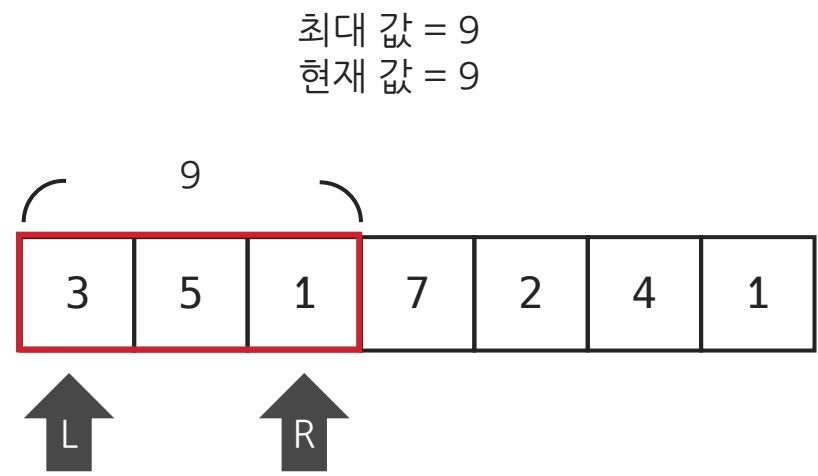
4 슬라이딩 윈도우

슬라이딩 윈도우 하는 법

- 1. 윈도우의 사이즈를 정한다.
- 2. 첫번째 윈도우에서 값을 계산한다.
- 3. 첫번째 계산값을 기반으로 윈도우를 옮겨가며 값을 갱신한다.

예제에 적용

- 1. 윈도우의 사이즈를 정한다.
 - ↳ 연속된 3개의 값을 봐야하니 윈도우 크기는 3이다.
- 2. 첫번째 윈도우에서 값을 계산한다.
 - ↳ $a[0] + a[1] + a[2] = 9$ 이다.



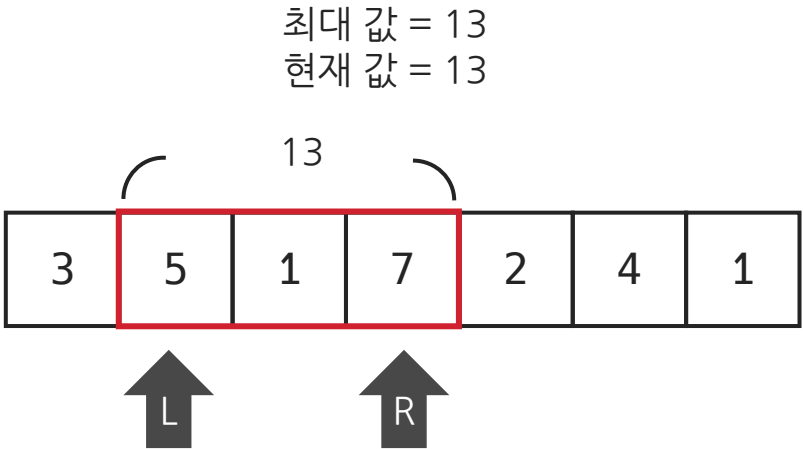
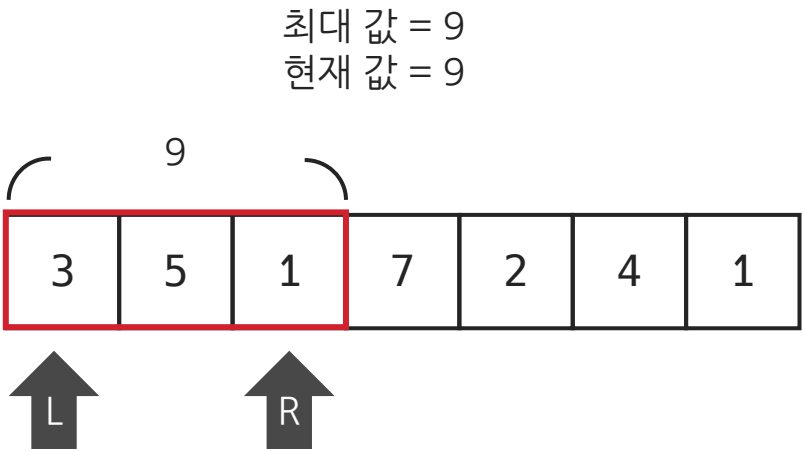
4 슬라이딩 윈도우

3. 첫번째 계산값을 기반으로 윈도우를 옮겨가며 값을 갱신한다

- 1. 현재 값에서 a[left]을 빼주고 left를 증가시킨다.
- 2. right를 증가시킨 후 현재 값에 a[right]을 더해준다.

```
answer = max(answer, current);
current -= a[left];
left += 1;
right += 1;
current += a[right];
```

left	right	a[left] + a[right]
0	2	9
1	3	13



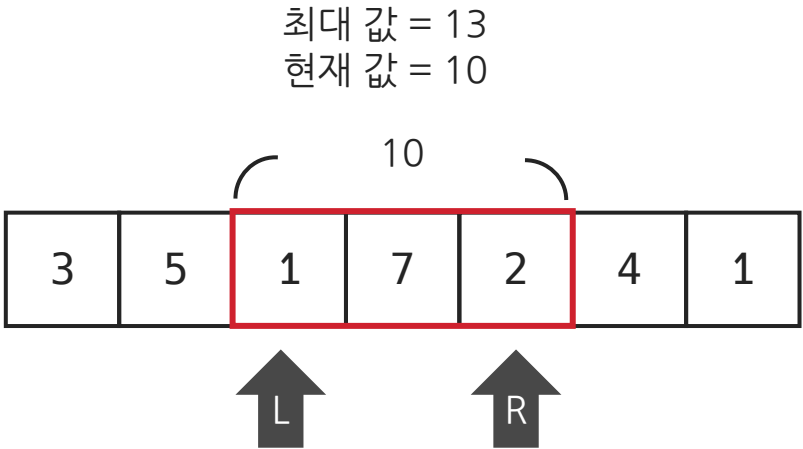
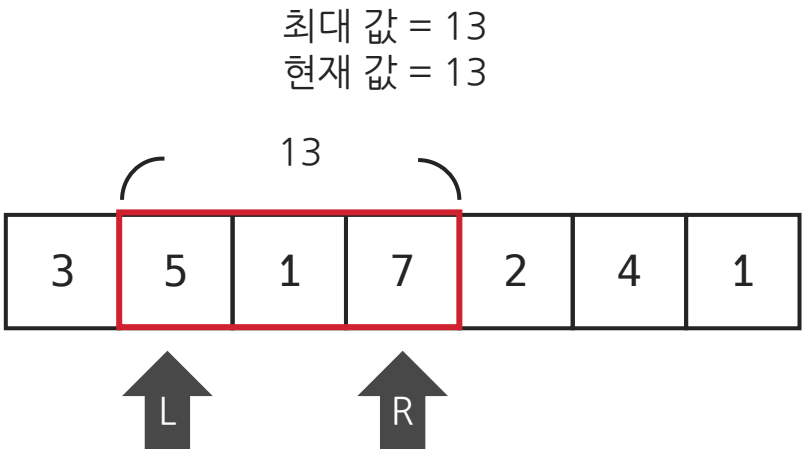
4 슬라이딩 윈도우

3. 첫번째 계산값을 기반으로 윈도우를 옮겨가며 값을 갱신한다

- 1. 현재 값에서 a[left]을 빼주고 left를 증가시킨다.
- 2. right을 증가시킨 후 현재 값에 a[right]을 더해준다.

```
answer = max(answer, current);
current -= a[left];
left += 1;
right += 1;
current += a[right];
```

left	right	a[left] + a[right]
0	2	9
1	3	13
2	4	10



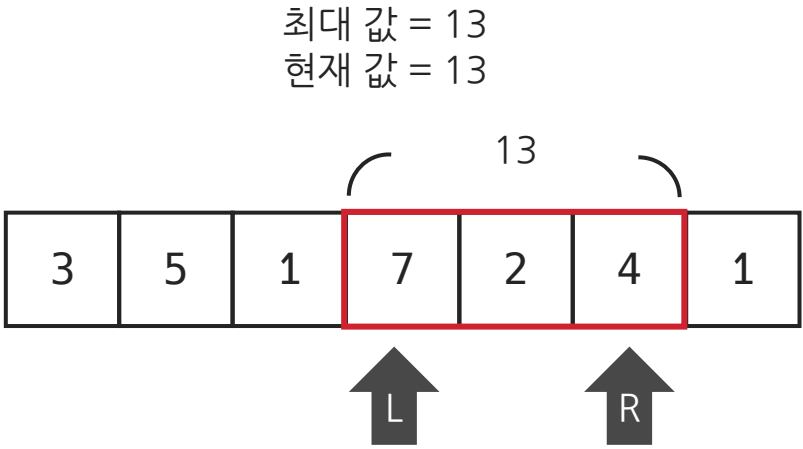
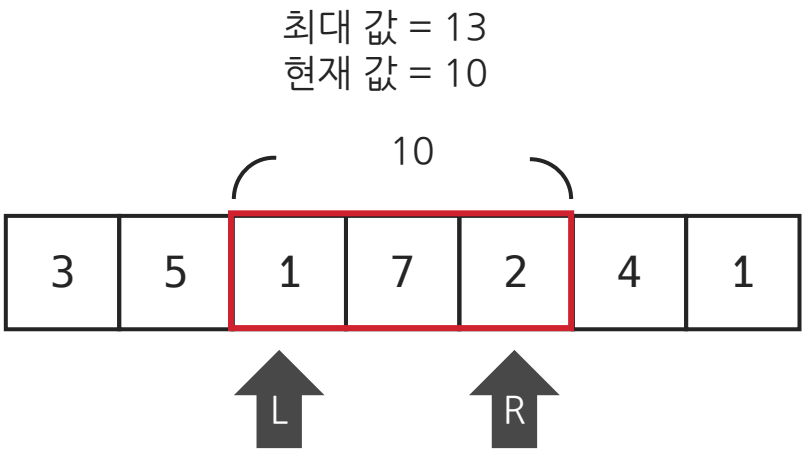
4 슬라이딩 윈도우

3. 첫번째 계산값을 기반으로 윈도우를 옮겨가며 값을 갱신한다

- 1. 현재 값에서 a[left]을 빼주고 left를 증가시킨다.
- 2. right을 증가시킨 후 현재 값에 a[right]을 더해준다.

```
answer = max(answer, current);
current -= a[left];
left += 1;
right += 1;
current += a[right];
```

left	right	a[left] + a[right]
0	2	9
1	3	13
2	4	10
3	5	13



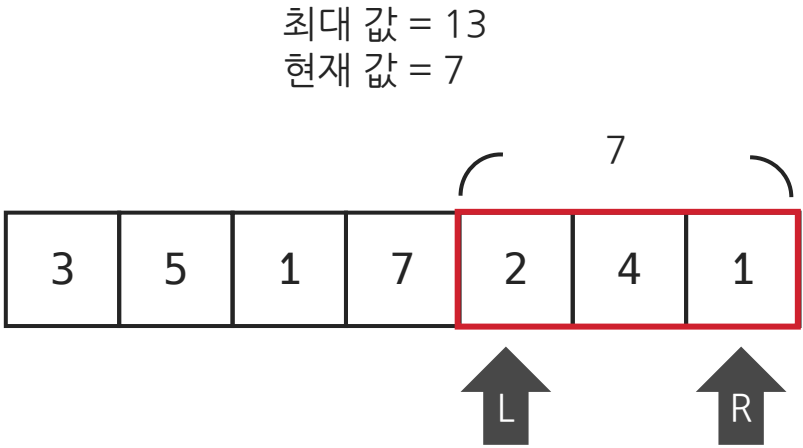
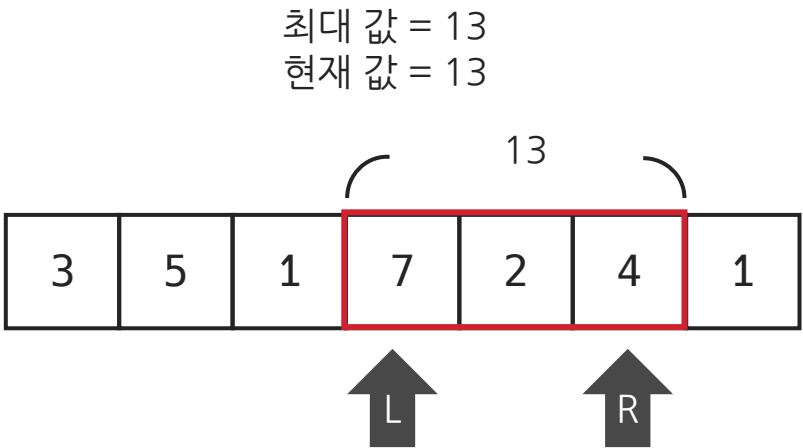
4 슬라이딩 윈도우

3. 첫번째 계산값을 기반으로 윈도우를 옮겨가며 값을 갱신한다

- 1. 현재 값에서 a[left]을 빼주고 left를 증가시킨다.
- 2. right을 증가시킨 후 현재 값에 a[right]을 더해준다.

```
answer = max(answer, current);
current -= a[left];
left += 1;
right += 1;
current += a[right];
```

left	right	a[left] + a[right]
0	2	9
1	3	13
2	4	10
3	5	13
4	7	7



4 슬라이딩 윈도우

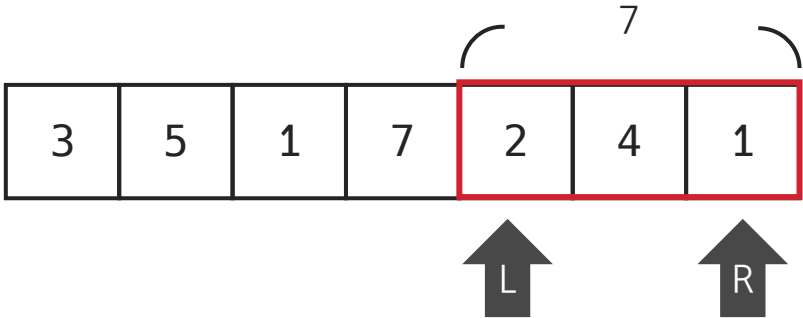
3. 첫번째 계산값을 기반으로 윈도우를 옮겨가며 값을 갱신한다

- 1. 현재 값에서 a[left]을 빼주고 left를 증가시킨다.
- 2. right을 증가시킨 후 현재 값에 a[right]을 더해준다.

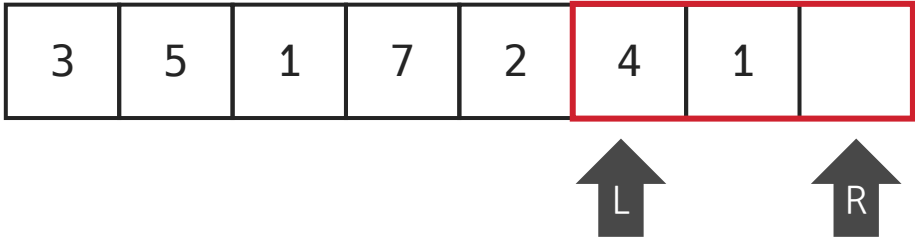
```
answer = max(answer, current);
current -= a[left];
left += 1;
right += 1;
current += a[right];
```

left	right	a[left] + a[right]
0	2	9
1	3	13
2	4	10
3	5	13
4	7	7
5	8	종료

최대 값 = 13
현재 값 = 7



종료



4 슬라이딩 윈도우

- 시간복잡도 분석

원소의 개수가 n 개인 배열에서 윈도우의 크기가 m 일 때, $right$ 가 $m - 1$ 번째 인덱스부터 n 번째 인덱스까지 움직인다.

따라서 $O(N - M)$ 이다. 보통 $N > M$ 이므로 $O(N)$ 이라고 부르기도 한다.

```
int left = 0, right = m - 1, answer = 0, current = 0;
while(right < n) {
    answer = max(answer, current);
    current -= a[left];
    left += 1;
    right += 1;
    current += a[right];
}
```