2021 겨울 신촌 연합 알고리즘 캠프

정렬 & 이분탐색

초급 알고리즘

HI-ARC 김기선

정렬(Sorting)

원소들을 번호순이나 사전 순서와 같이 일정한 순서대로 열거하는 알고리즘

Ex. 4 2 3 1 -> 1 2 3 4
4 2 3 1 -> 4 3 2 1

얼마나 빠른 시간 내에 정렬하는지
메모리는 얼마나 써야 하는지가 중요한 요소

선택(Selection) 정렬

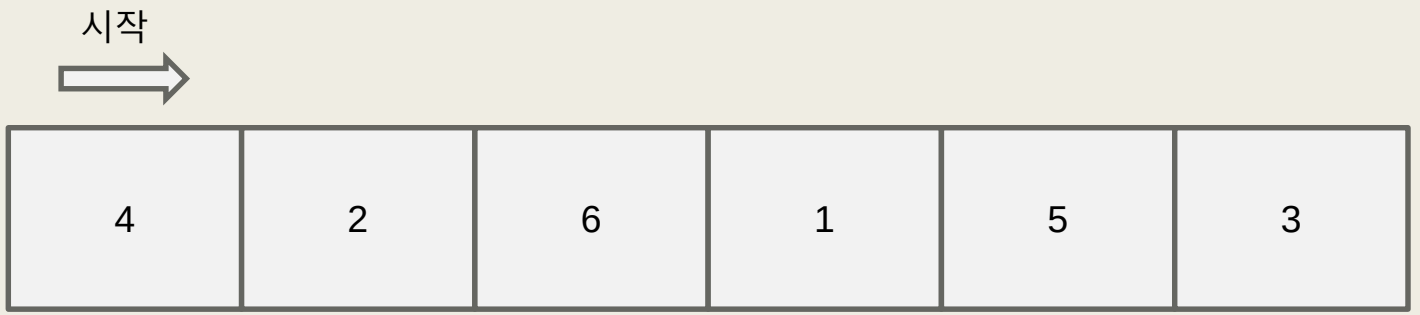
- 정렬되지 않은 리스트의 최솟값을 선택
- 그 최솟값을 리스트의 맨 앞의 값과 바꾼다.
- 최솟값이 저장된 index의 다음부터 위 과정을 반복한다.

시간복잡도 : 최선 O(n^2) 평균 O(n^2) 최악 O(n^2)

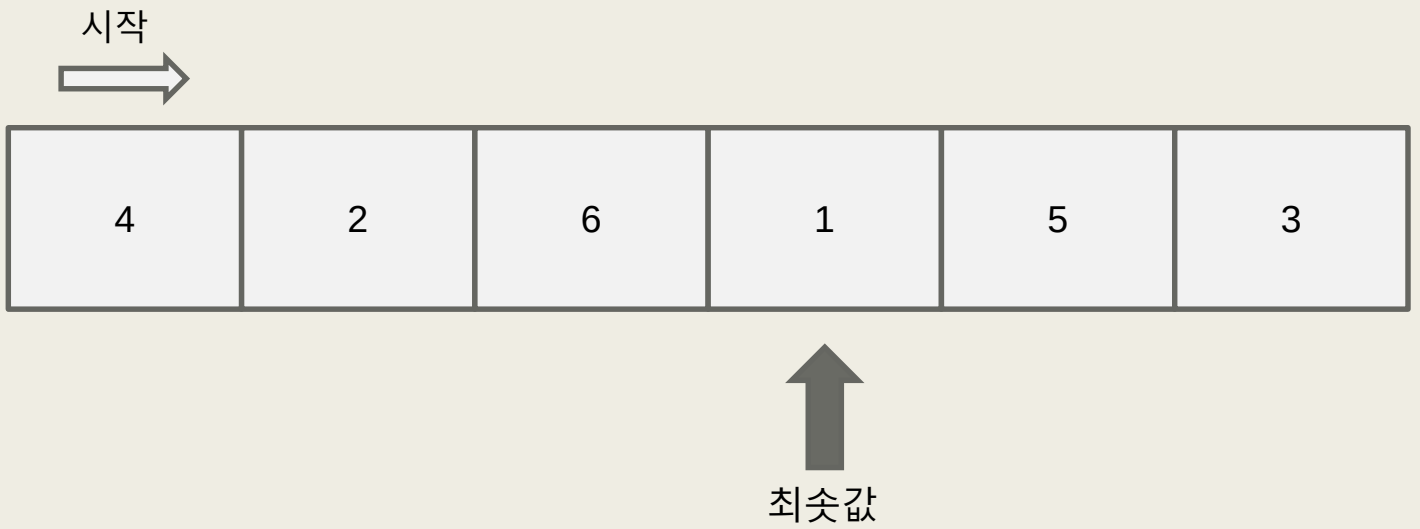
공간복잡도 : O(1)

선택 정렬

선택 정렬

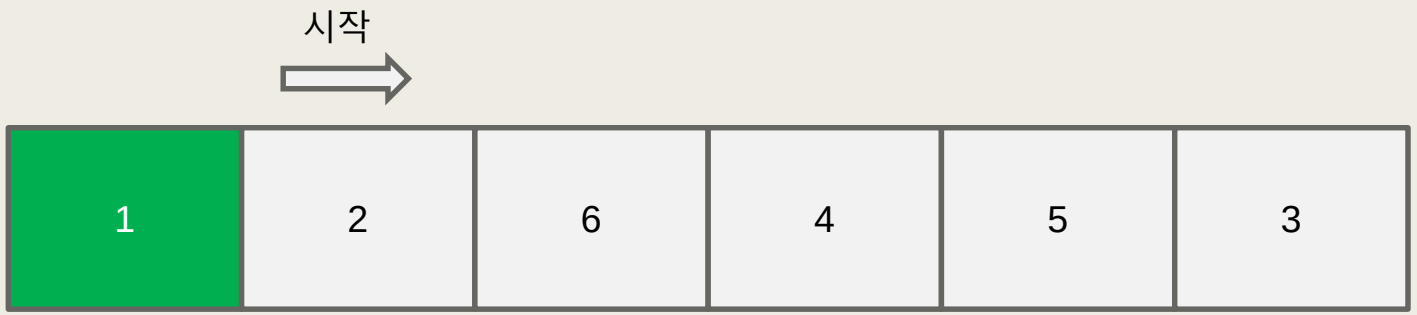


선택 정렬

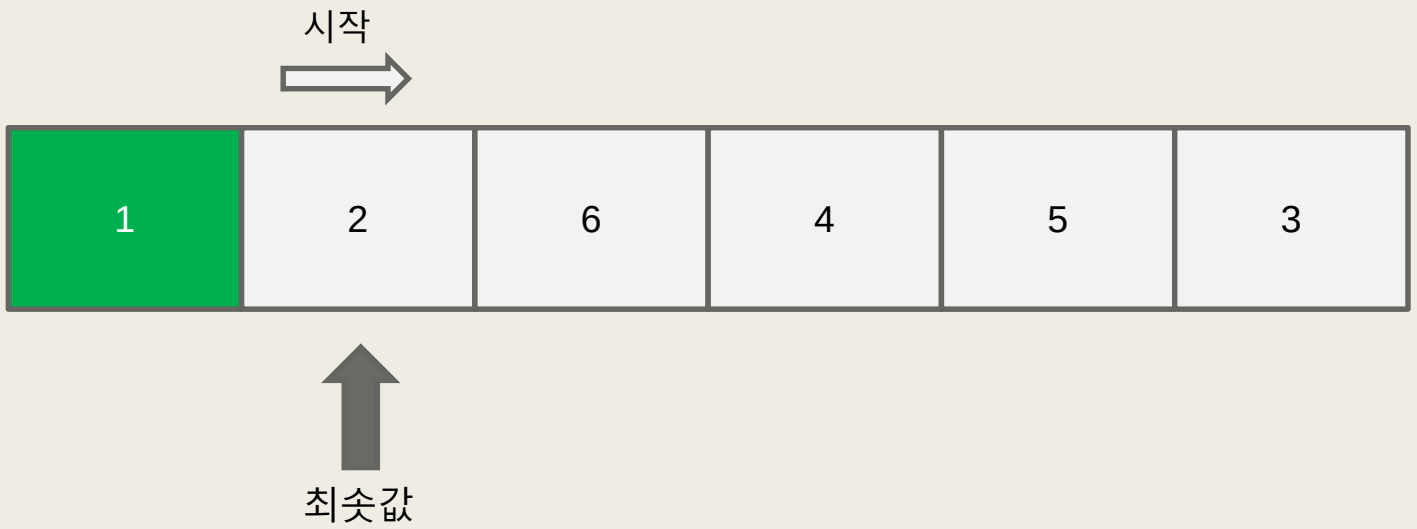


선택 정렬

선택 정렬



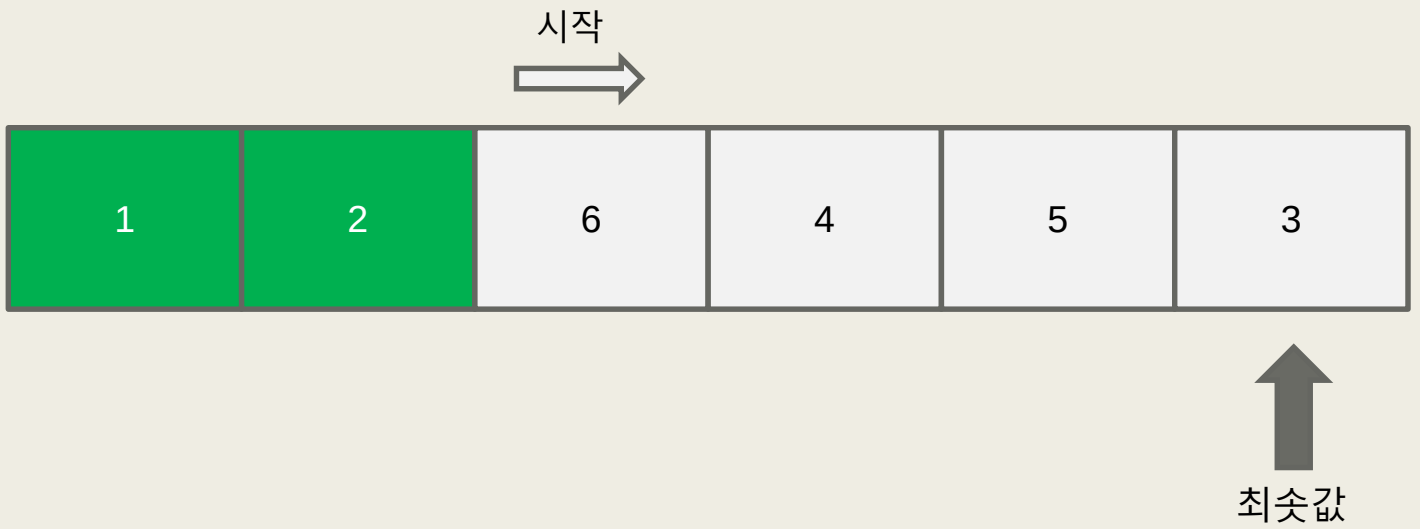
선택 정렬



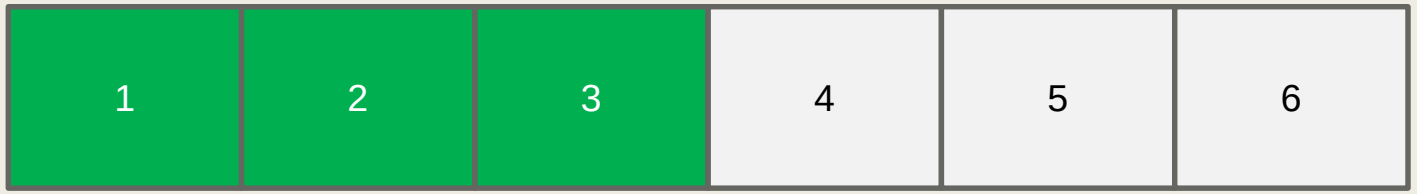
선택 정렬

선택 정렬

선택 정렬

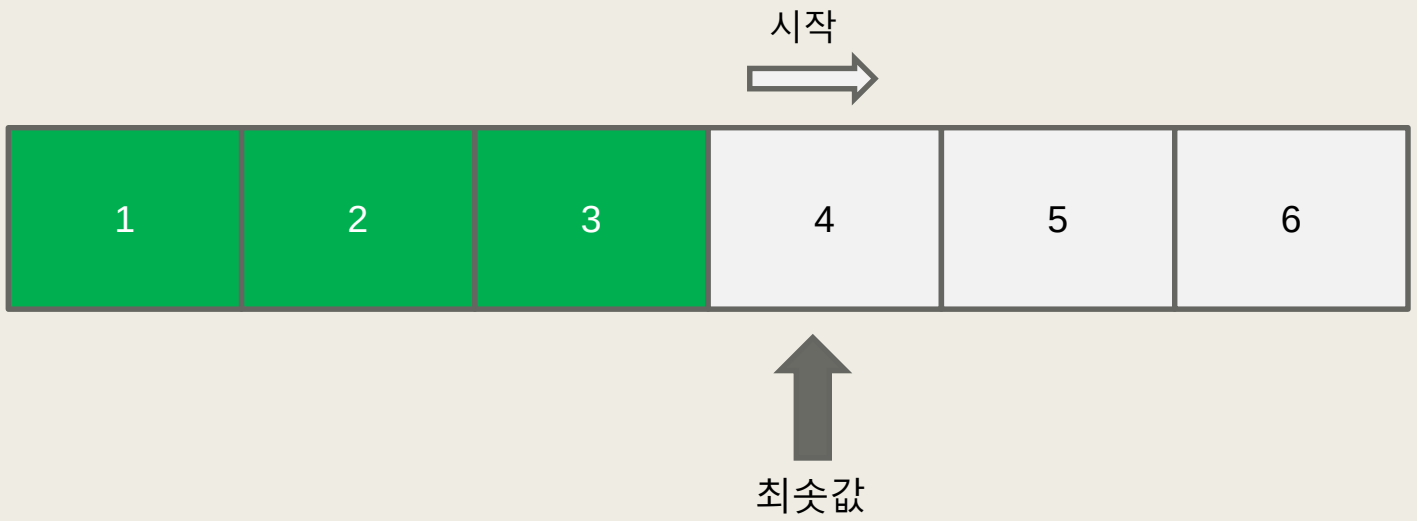


선택 정렬

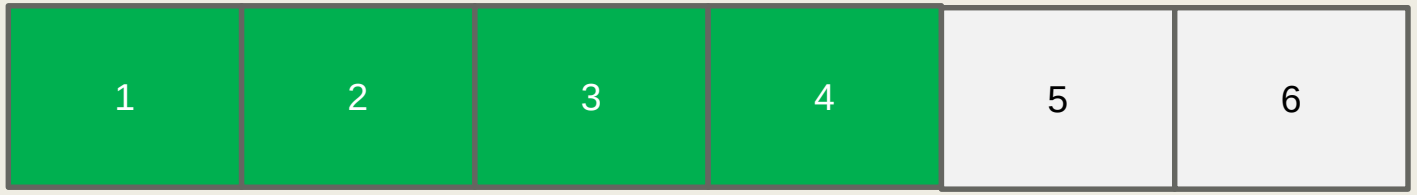


선택 정렬

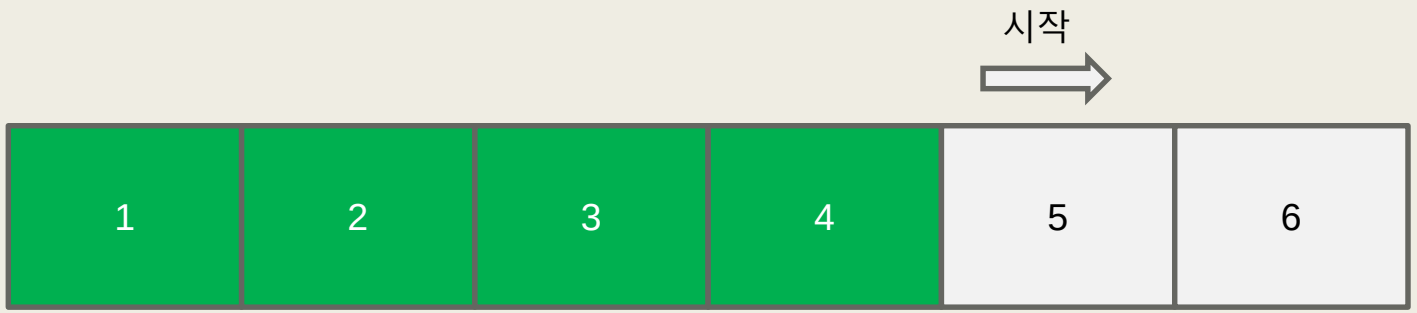
선택 정렬



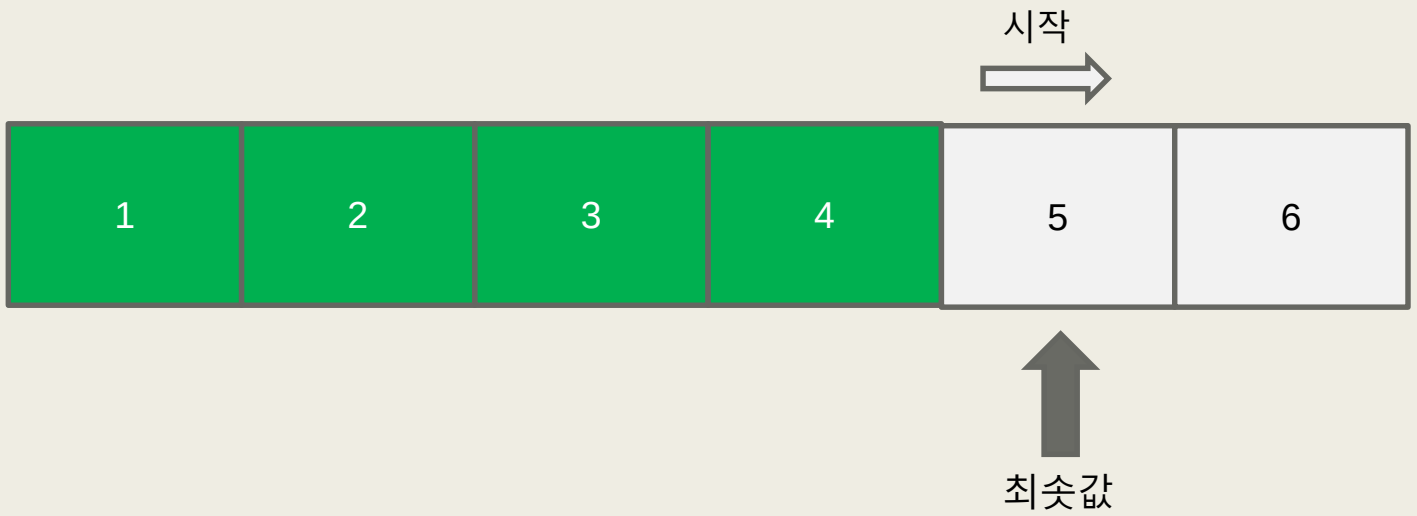
선택 정렬



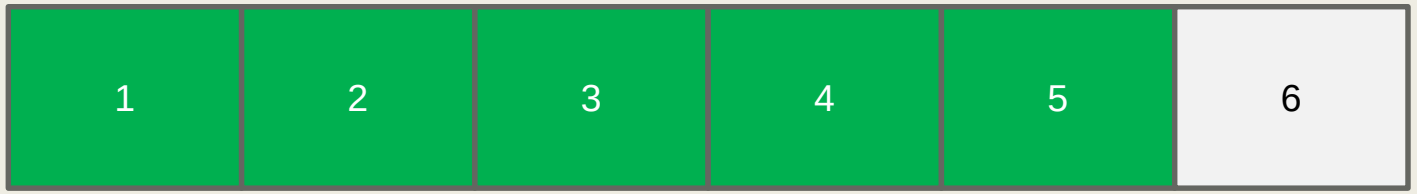
선택 정렬



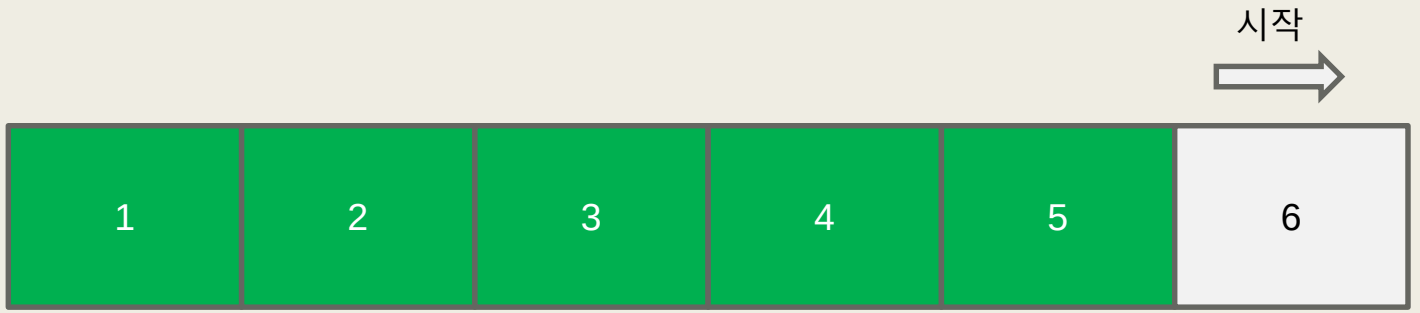
선택 정렬



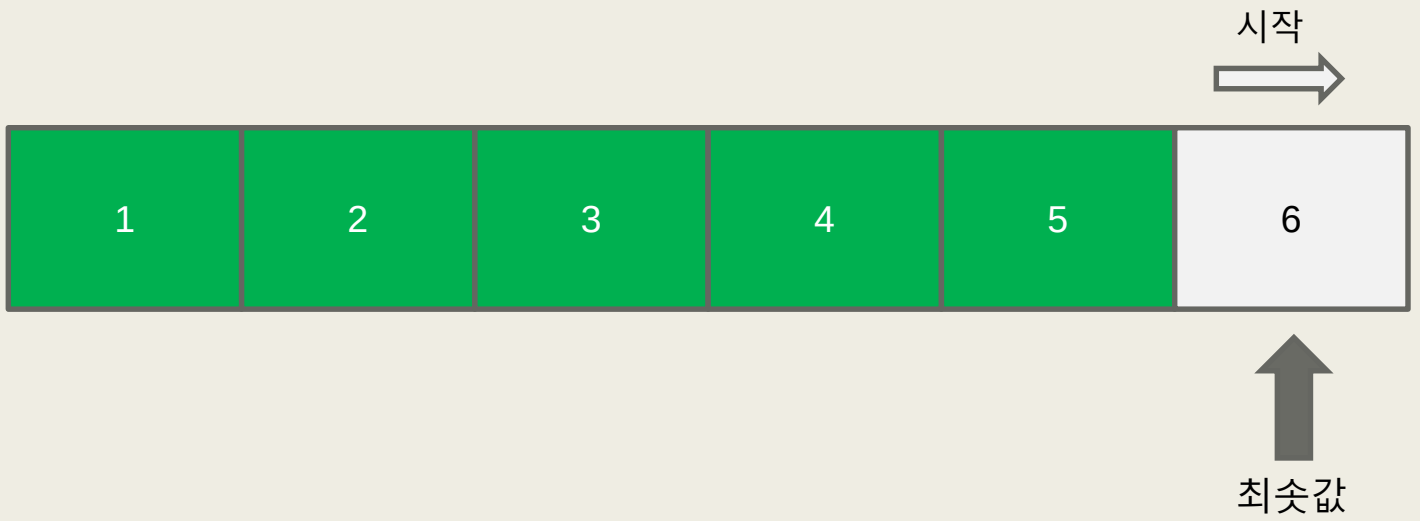
선택 정렬



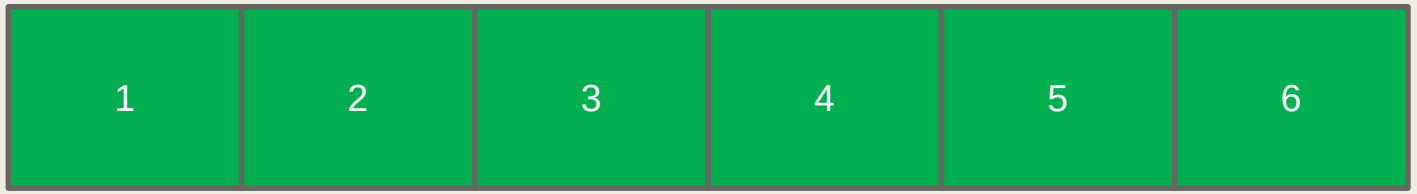
선택 정렬



선택 정렬



선택 정렬



선택 정렬


```
1 void selection_sort(){
2     int temp, min_index;
3     for (int i = 1; i <= n; i++) {
4         min_index = i;
5         for (int j = i + 1; j <= n; j++) {
6             if (list[j] < list[min_index]) {
7                 min_index = j;
8             }
9         }
10        temp = list[i];
11        list[i] = list[min_index];
12        list[min_index] = temp;
13    }
14 }
```

삽입(Insertion) 정렬

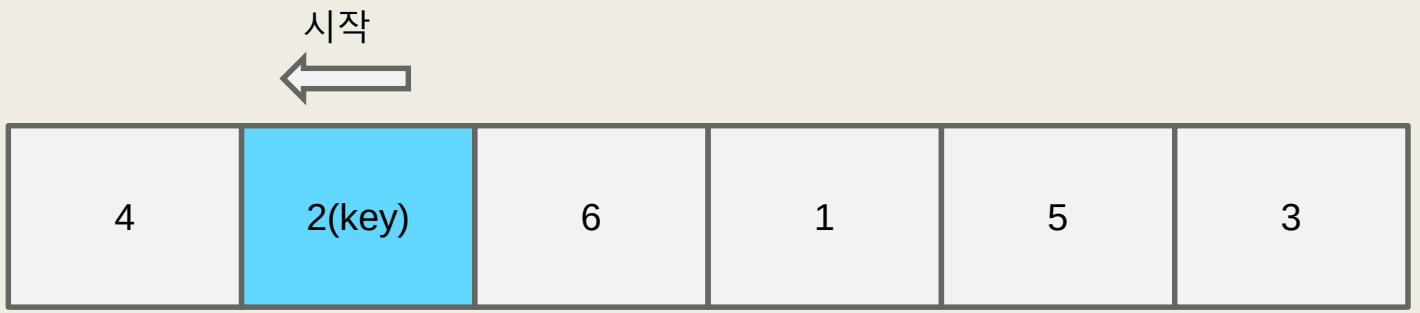
- 두번째 index부터 시작, key값을 index값으로 함
- key값과 정렬된 앞의 값들과 비교를 한 뒤에 적절한 위치에 삽입
- index+1 을 한 뒤 위 과정을 반복한다.

시간복잡도 : 최선 O(n) 평균 O(n^2) 최악 O(n^2)

공간복잡도 : O(1)

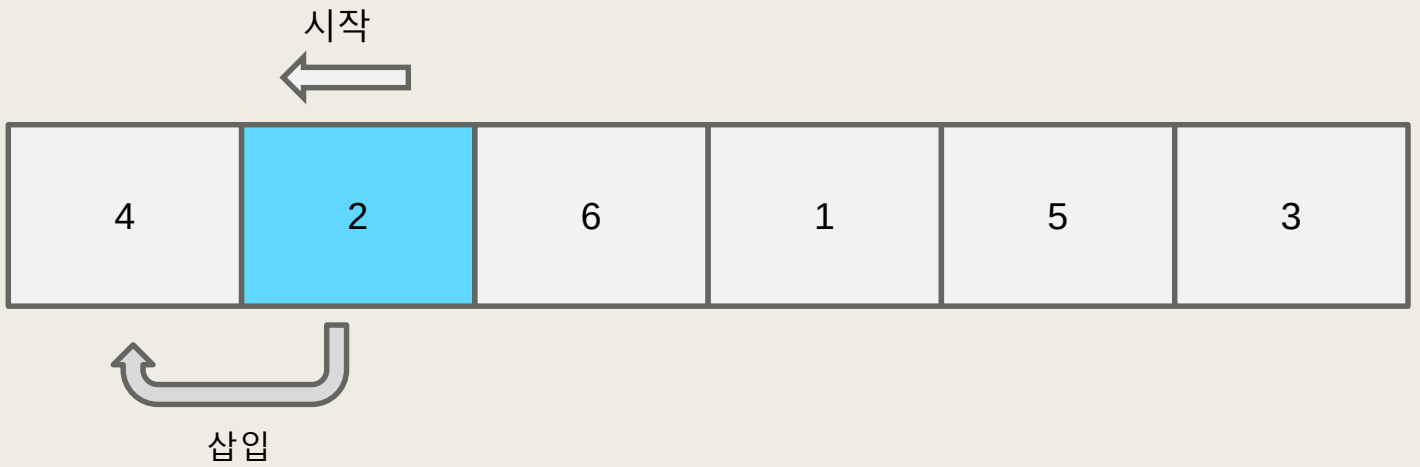
삽입 정렬

Key값

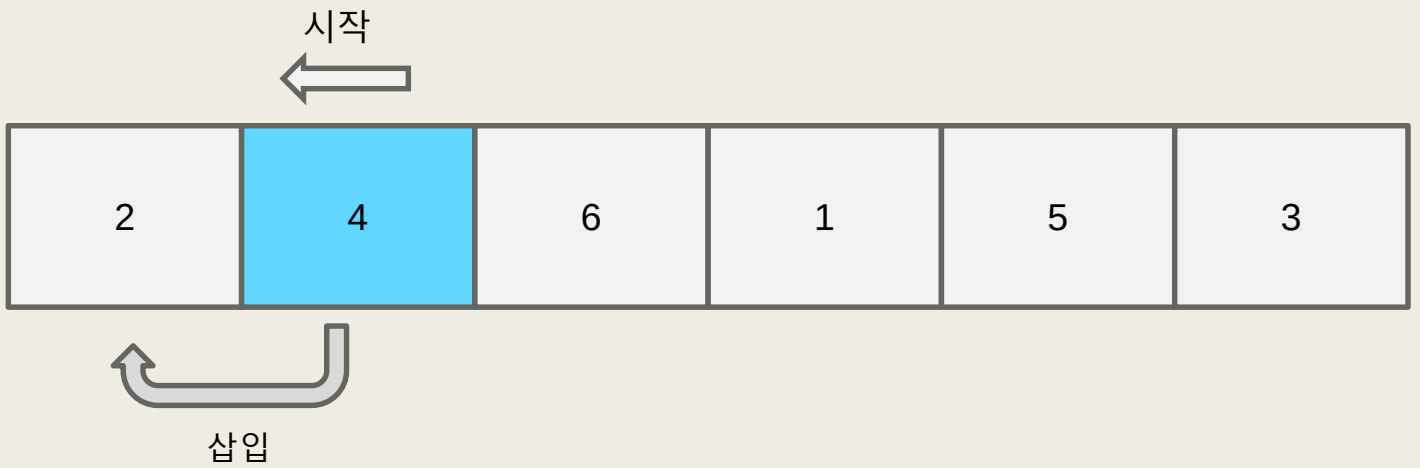


삽입 정렬

Key값

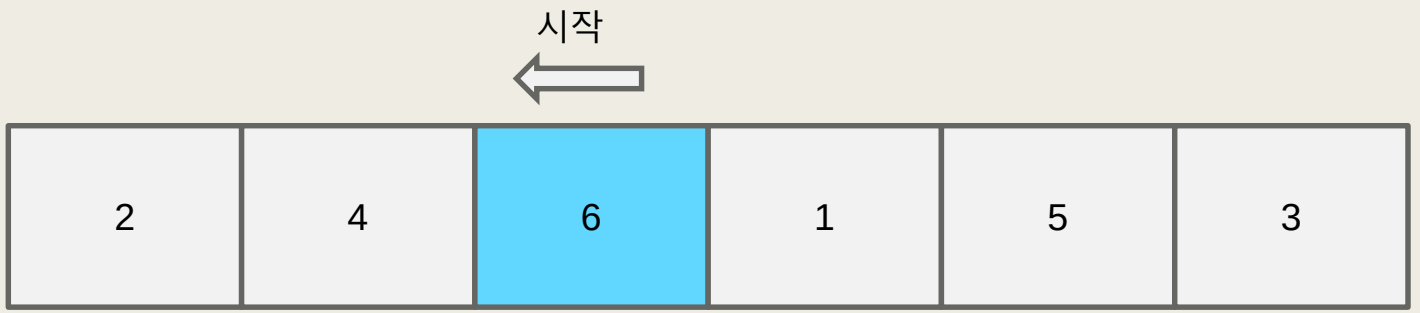


삽입 정렬



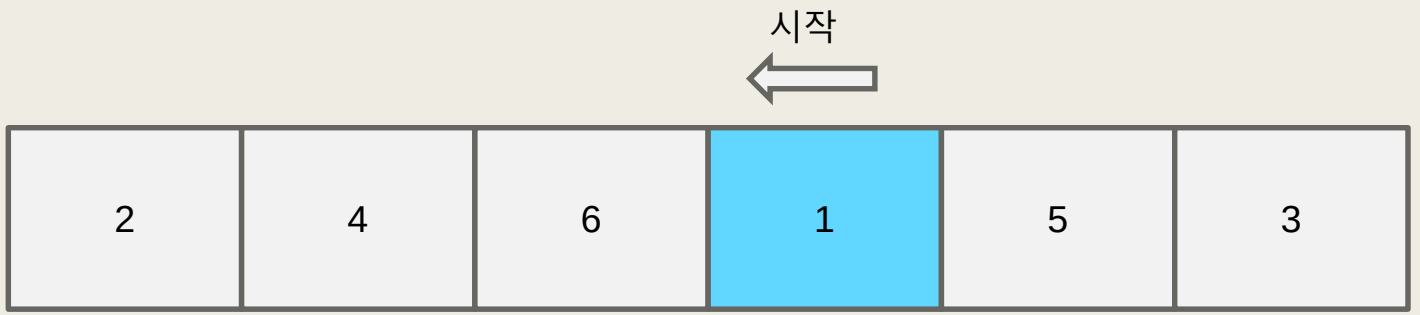
삽입 정렬

Key값

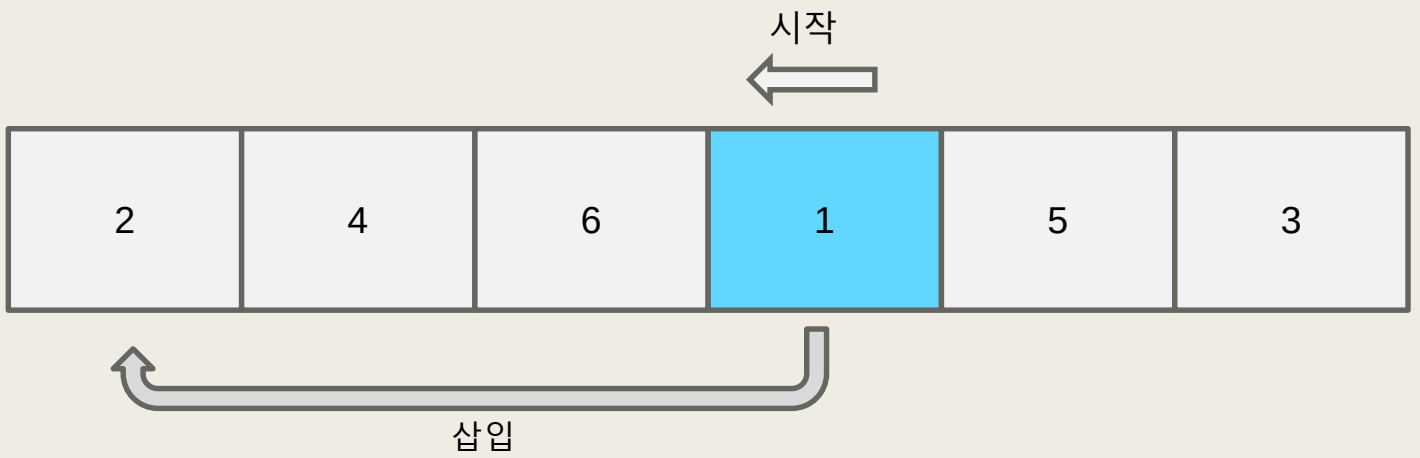


삽입 정렬

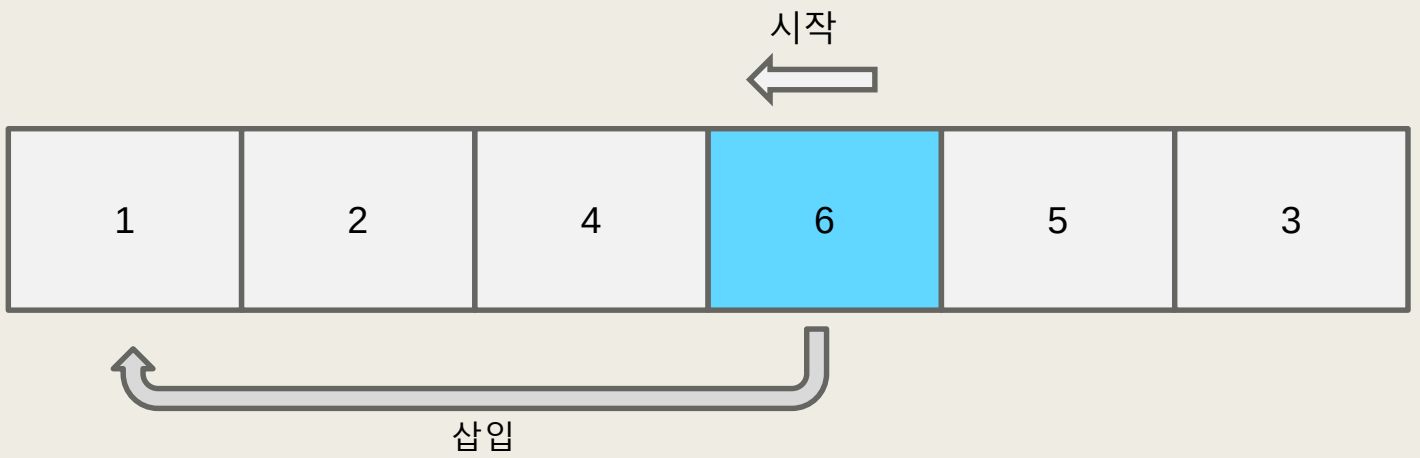
Key값



삽입 정렬

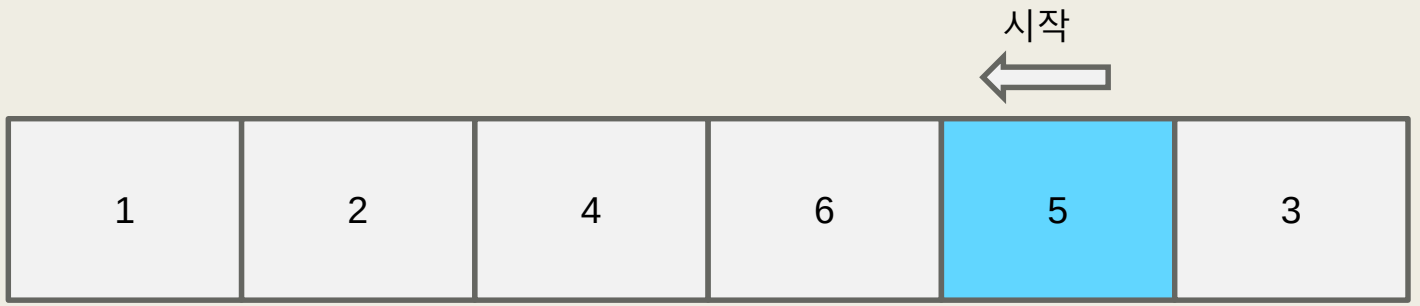


삽입 정렬

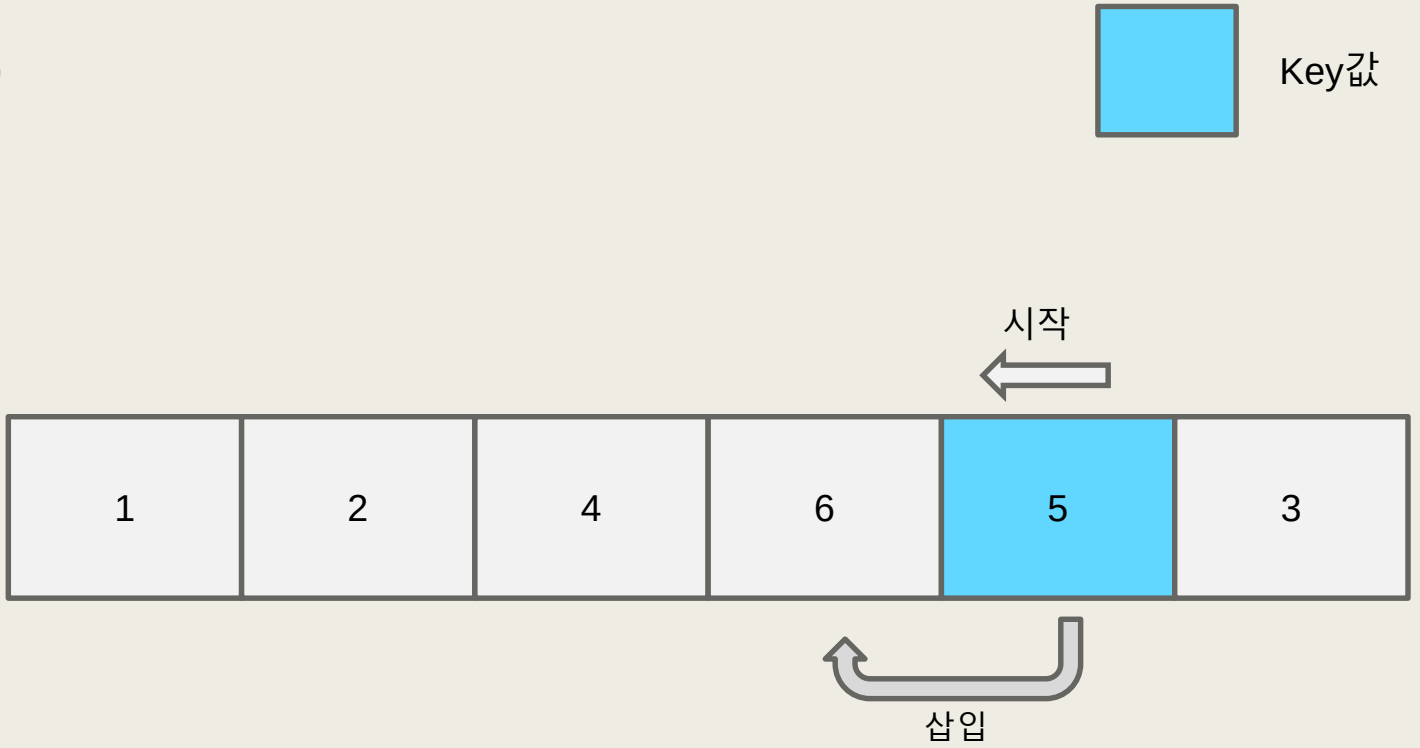


삽입 정렬

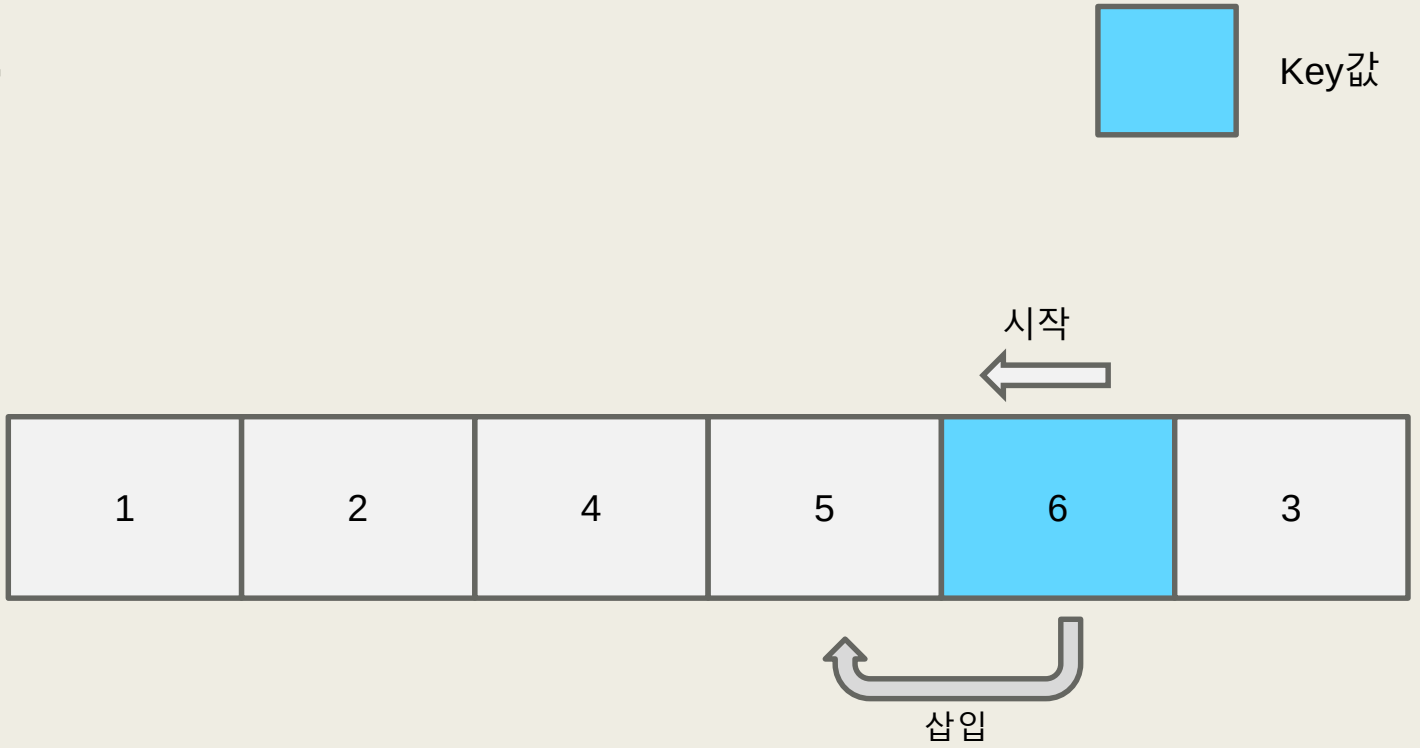
Key값



삽입 정렬



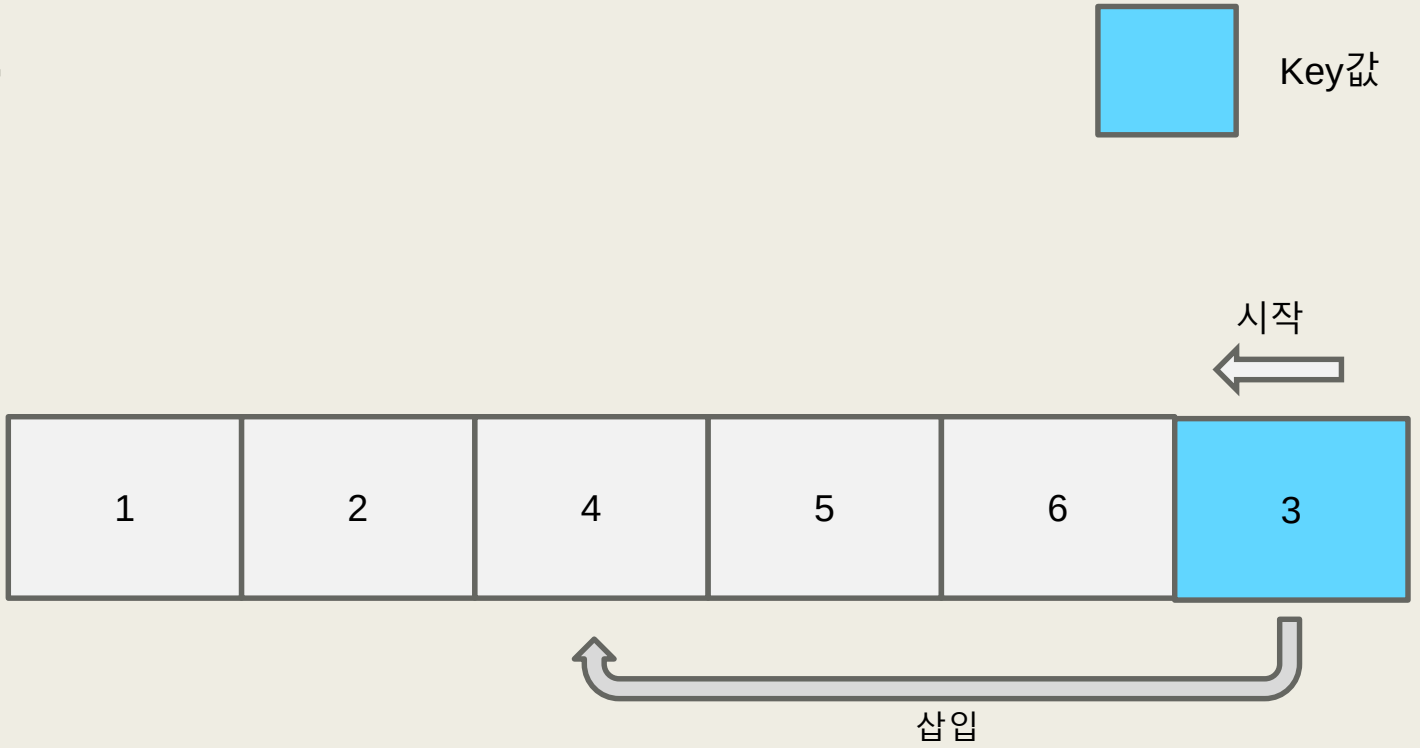
삽입 정렬



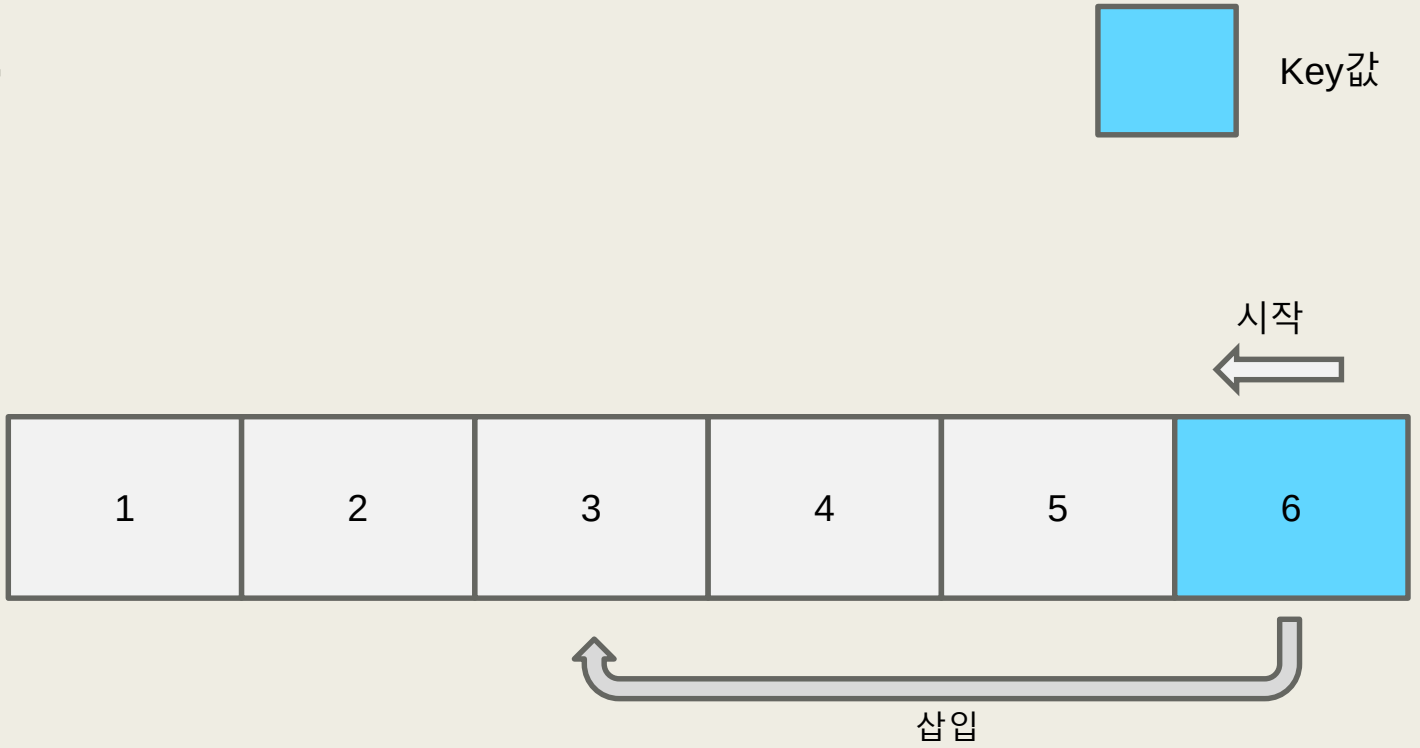
삽입 정렬

Key값

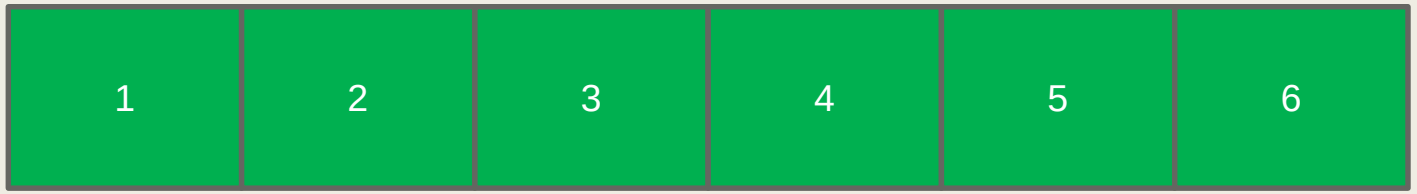
삽입 정렬



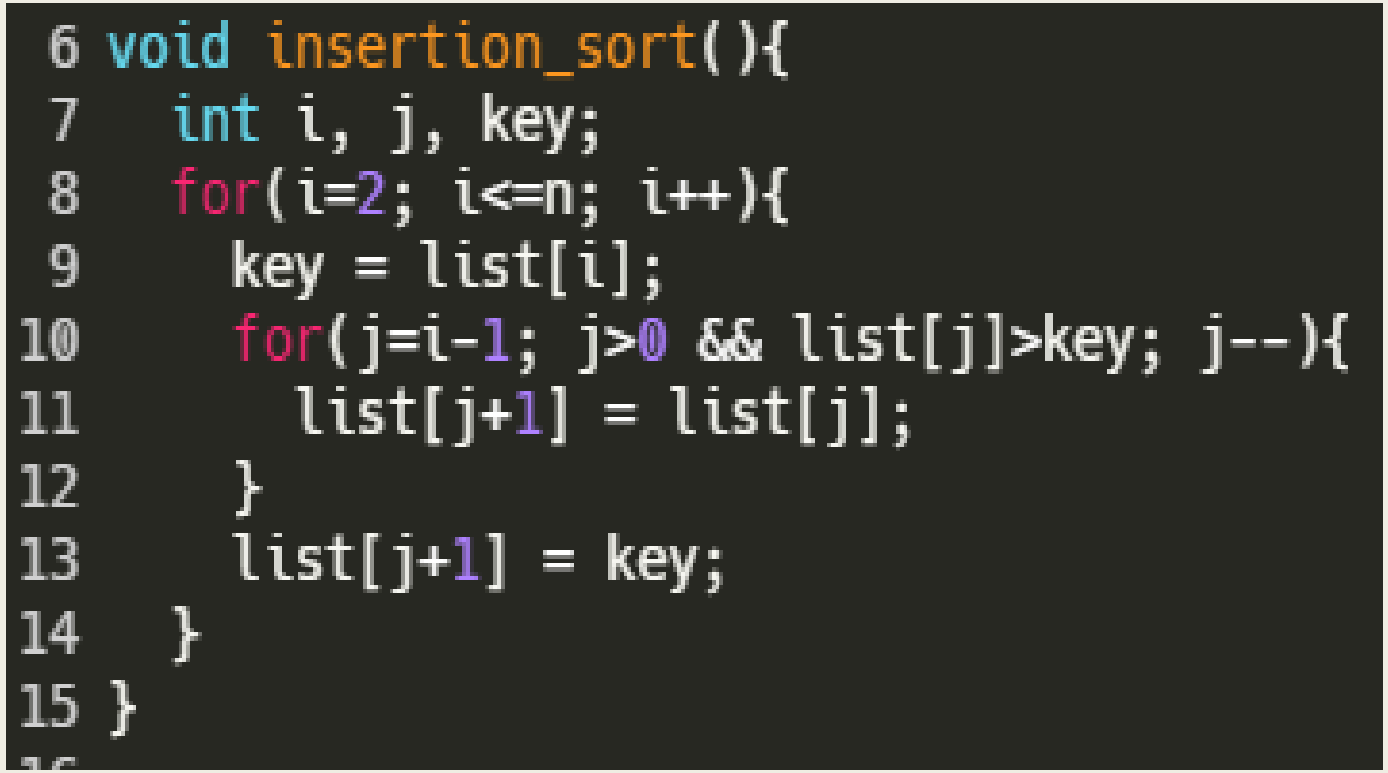
삽입 정렬



삽입 정렬



삽입 정렬


6 void insertion_sort(){
7 int i, j, key;
8 for(i=2; i<=n; i++){
9 key = list[i];
10 for(j=i-1; j>0 && list[j]>key; j--){
11 list[j+1] = list[j];
12 }
13 list[j+1] = key;
14 }
15 }

버블(Bubble) 정렬

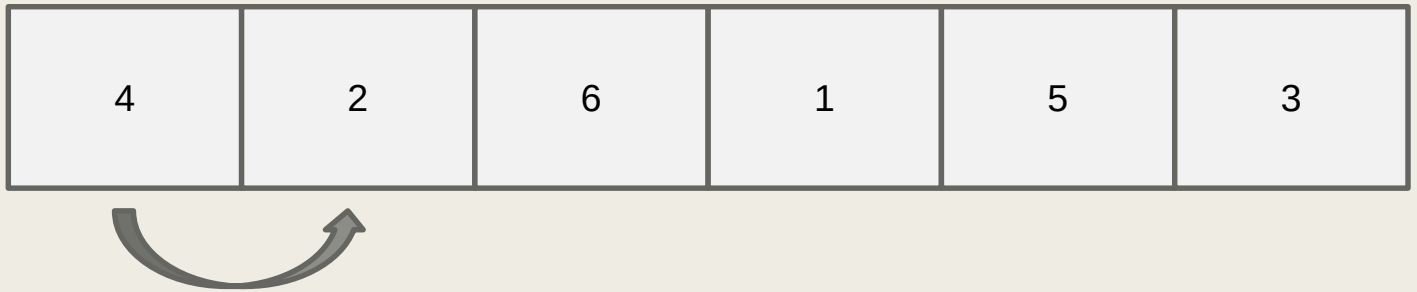
- 어떤 index와 index+1을 비교해서 정렬한다.
- index를 리스트의 갯수-1 까지 진행시킨다.
- 마지막 index를 제외한 뒤 위 과정을 반복한다.

시간복잡도 : 최선 O(n^2) 평균 O(n^2) 최악 O(n^2)

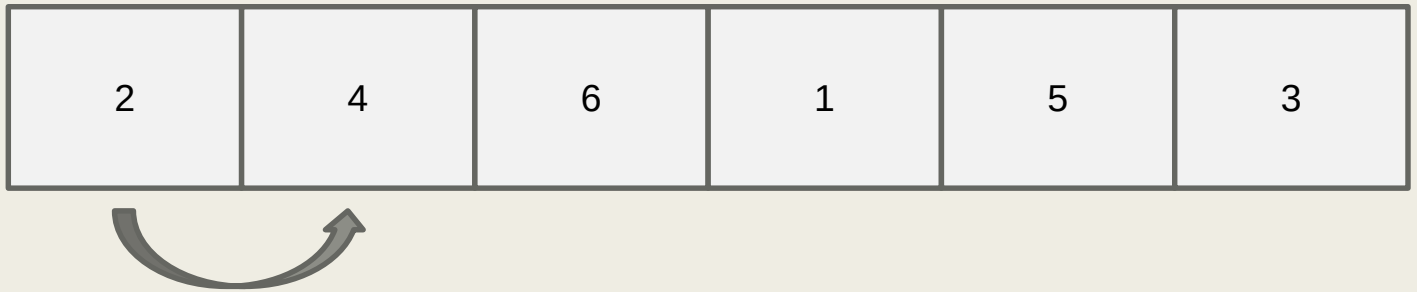
공간복잡도 : O(1)

버블 정렬

버블 정렬

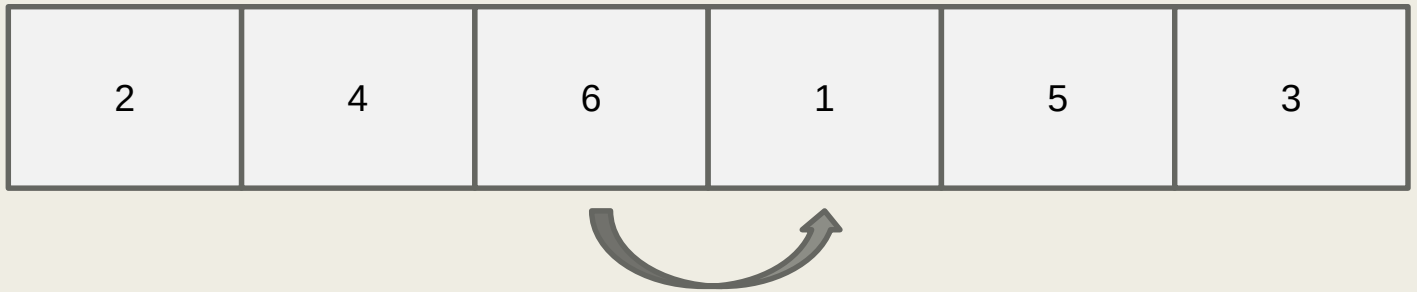


버블 정렬

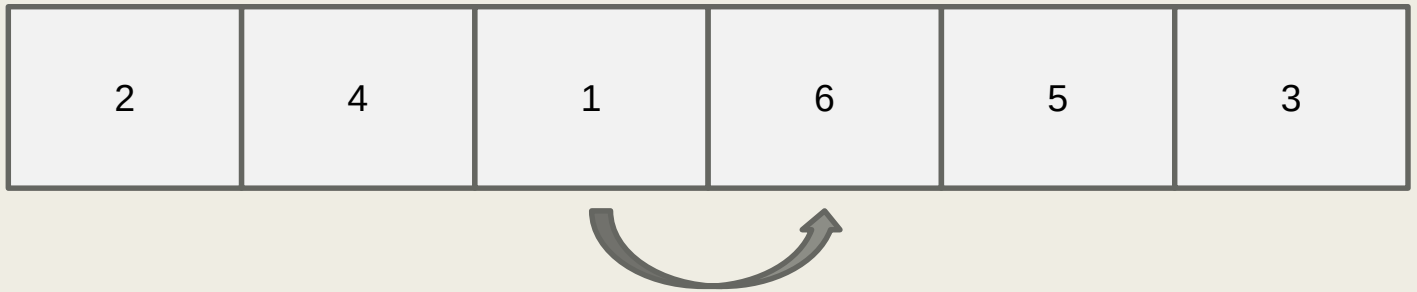


버블 정렬

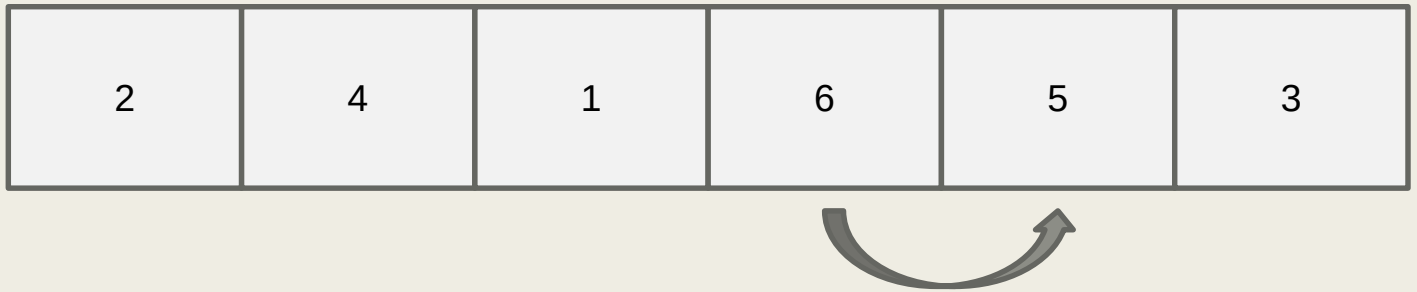
버블 정렬



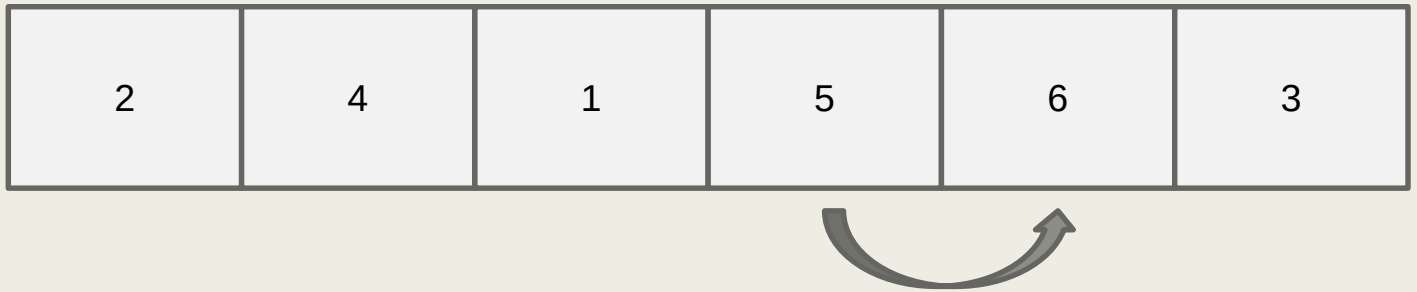
버블 정렬



버블 정렬



버블 정렬



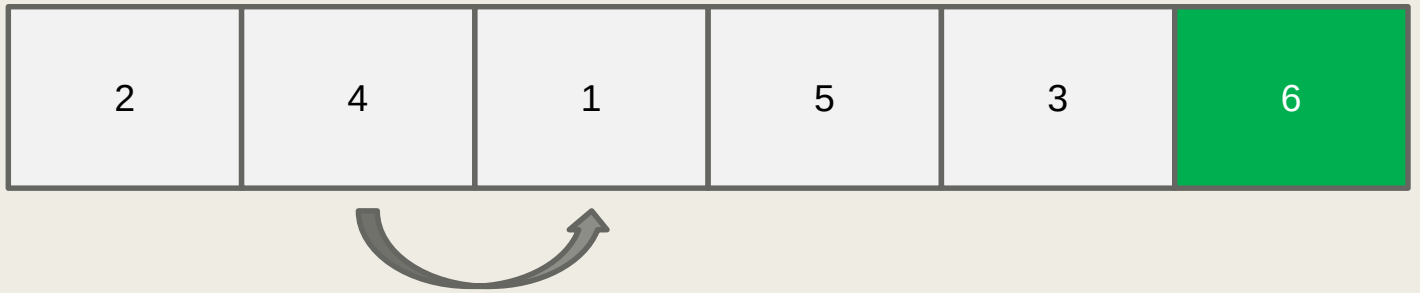
버블 정렬

버블 정렬

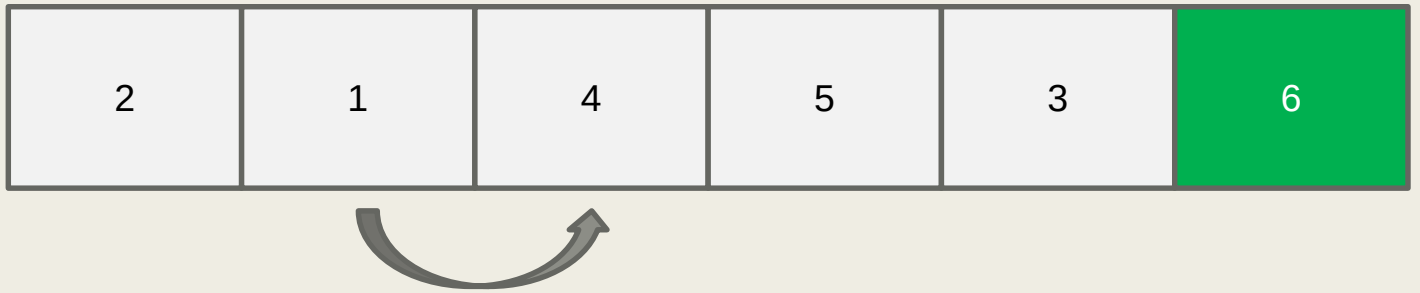
버블 정렬

버블 정렬

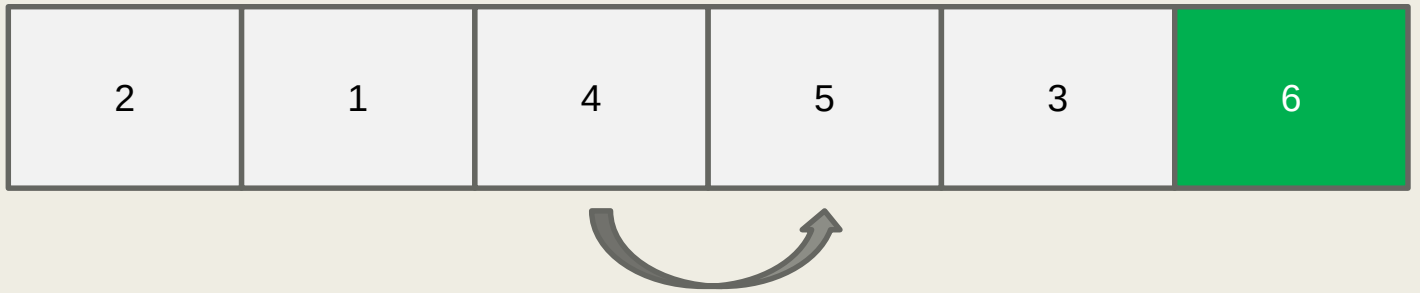
버블 정렬



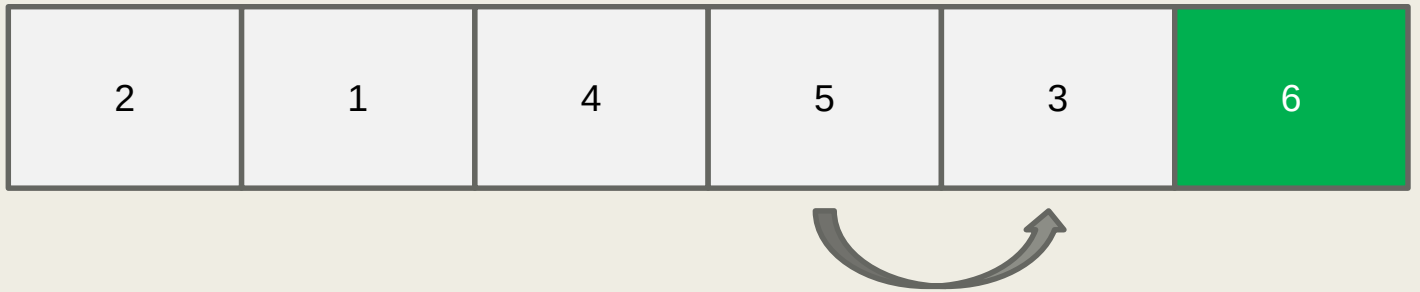
버블 정렬



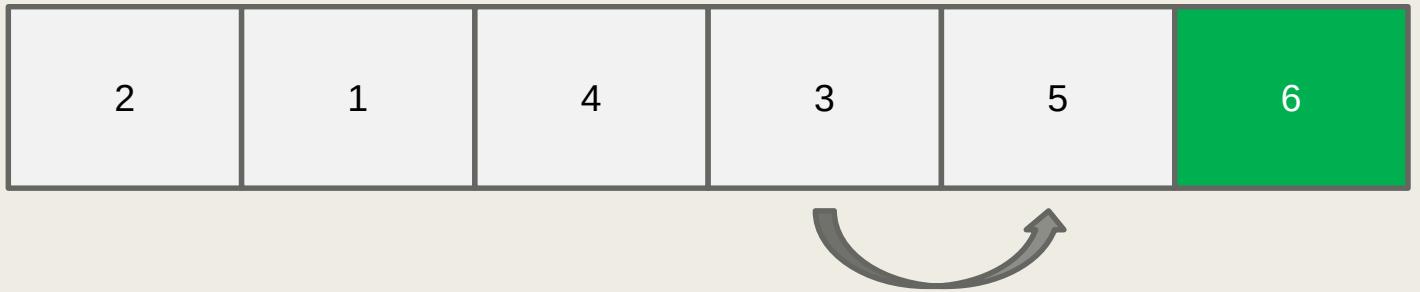
버블 정렬



버블 정렬

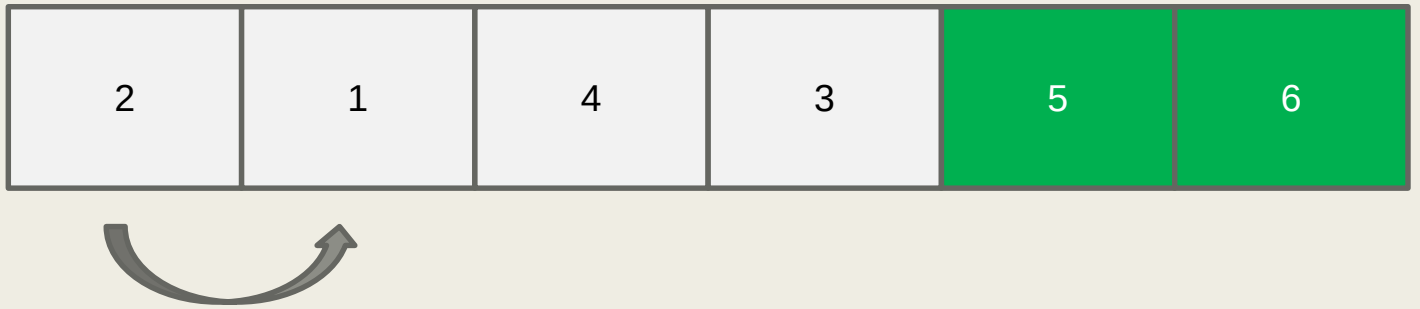


버블 정렬

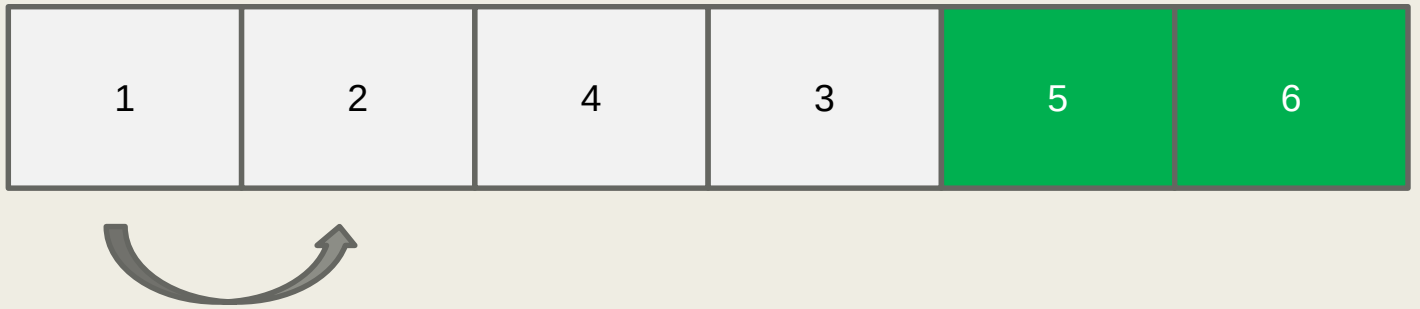


버블 정렬

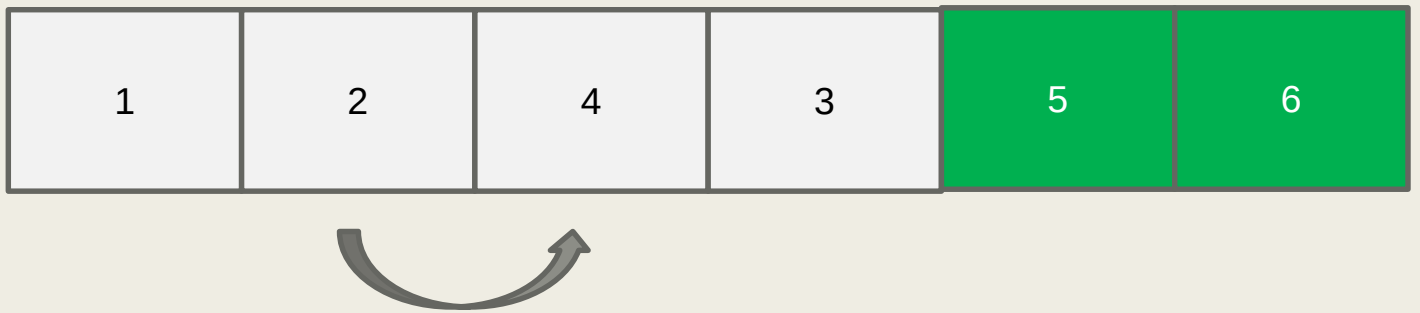
버블 정렬



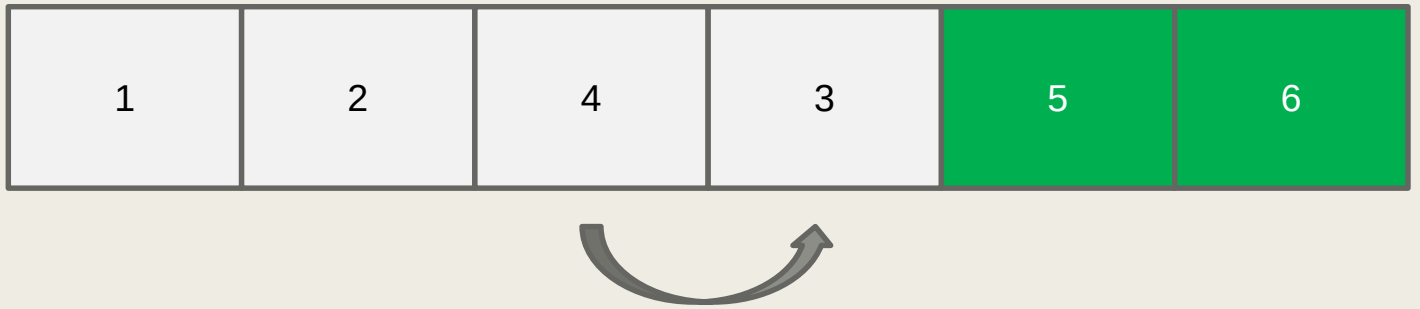
버블 정렬



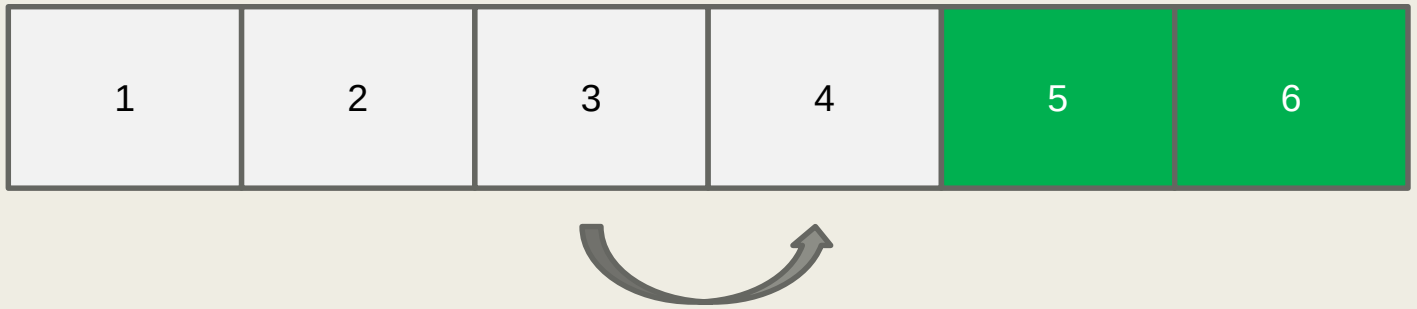
버블 정렬



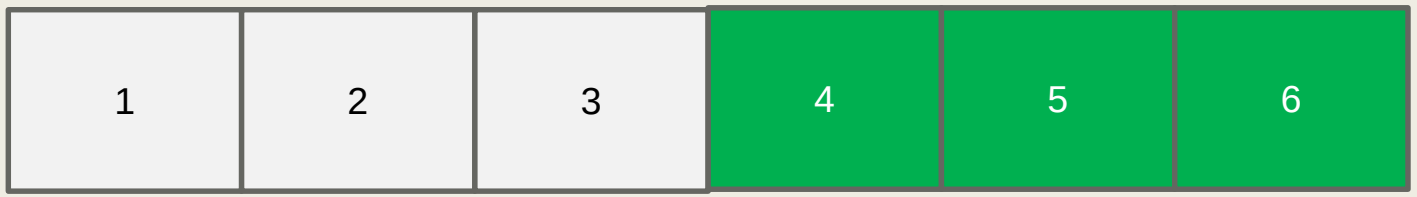
버블 정렬



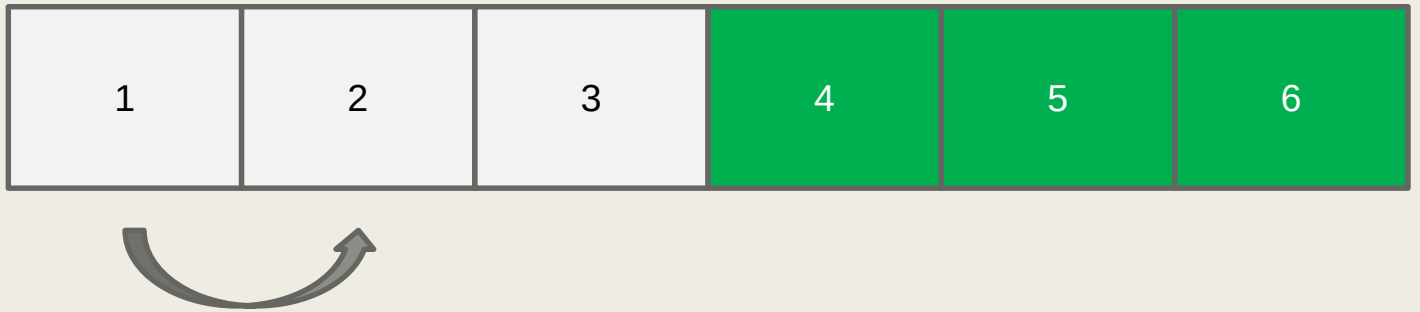
버블 정렬



버블 정렬

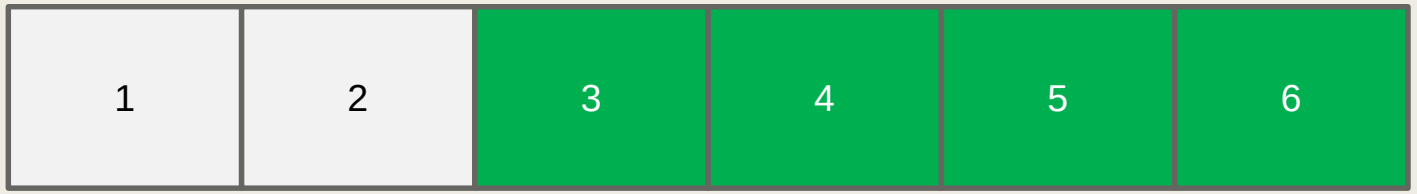


버블 정렬

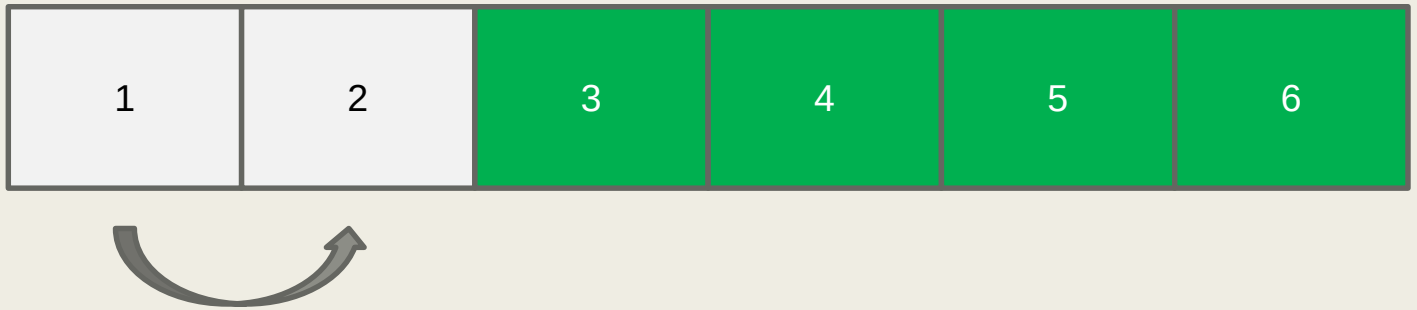


버블 정렬

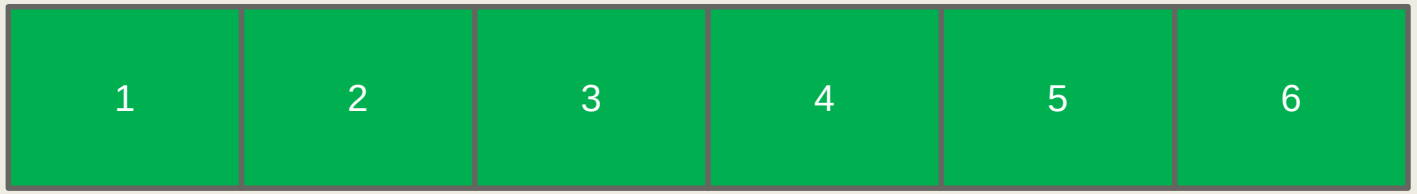
버블 정렬



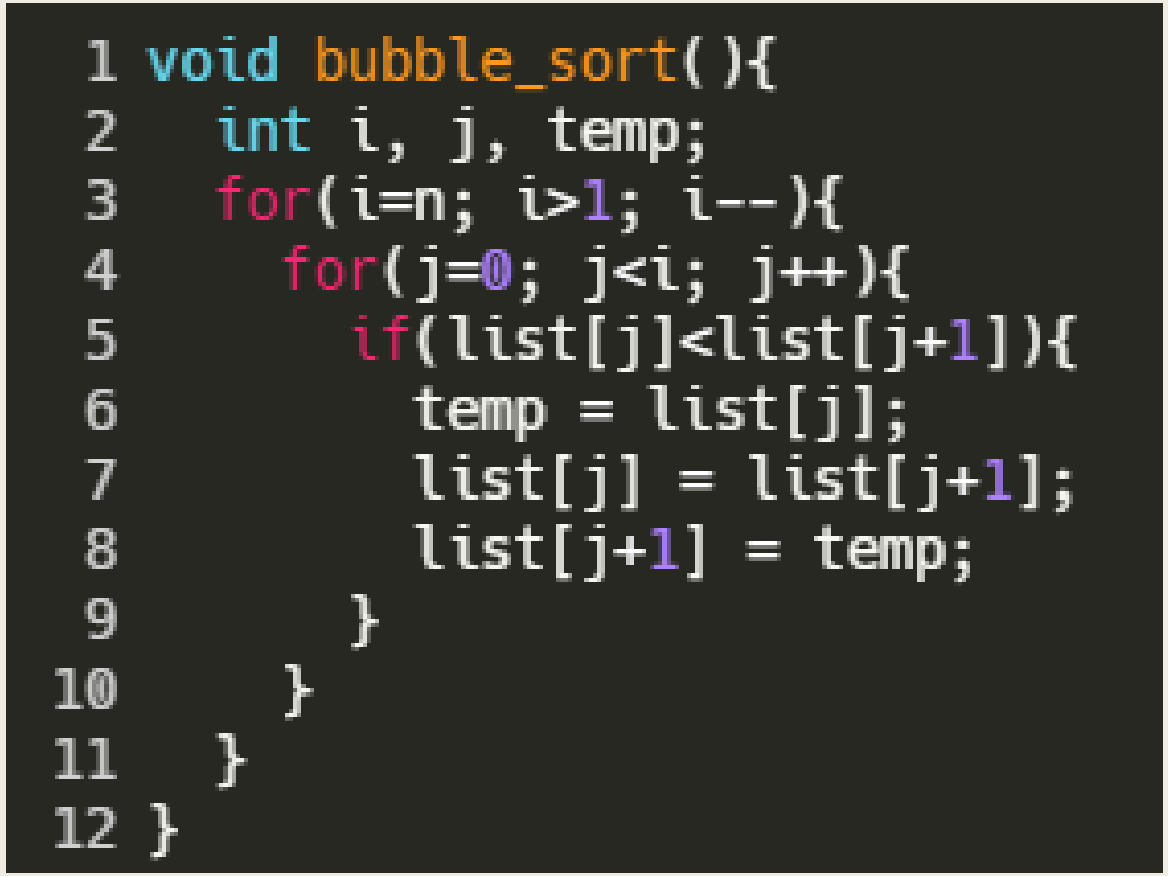
버블 정렬



버블 정렬



버블 정렬



```
 1 void bubble_sort(){
 2   int i, j, temp;
 3   for(i=n; i>1; i--){
 4     for(j=0; j<i; j++){
 5       if(list[j]<list[j+1]){
 6         temp = list[j];
 7         list[j] = list[j+1];
 8         list[j+1] = temp;
 9       }
10     }
11   }
12 }
```

퀵(Quick) 정렬

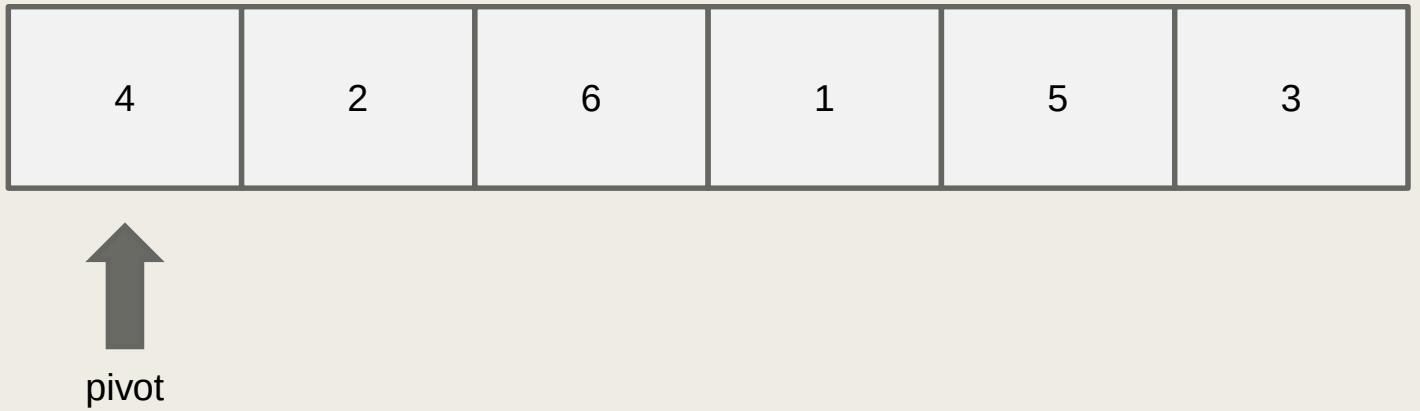
- 임의의 index를 pivot으로 잡는다.
- 피벗 좌측에는 피벗보다 작은 수, 우측에는 큰 수가 오게끔 배치한다.
- 피벗을 제외한 피벗의 좌측과 우측 두 개의 리스트에 대해서 위 과정을 재귀적으로 반복한다.

시간복잡도 : 최선 O(nlogn) 평균 O(nlogn) 최악 O(n^2)

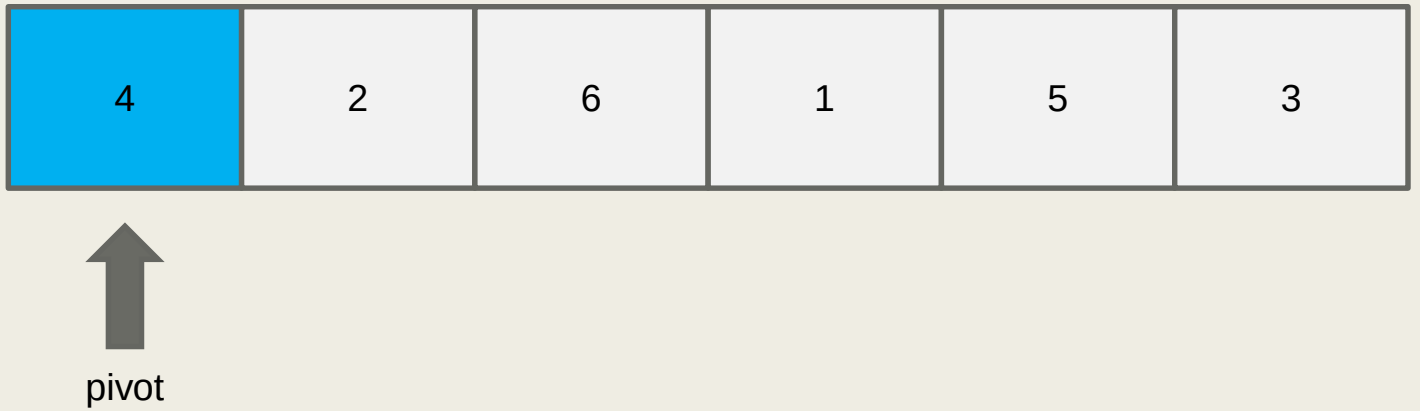
공간복잡도 : O(1)

퀵 정렬

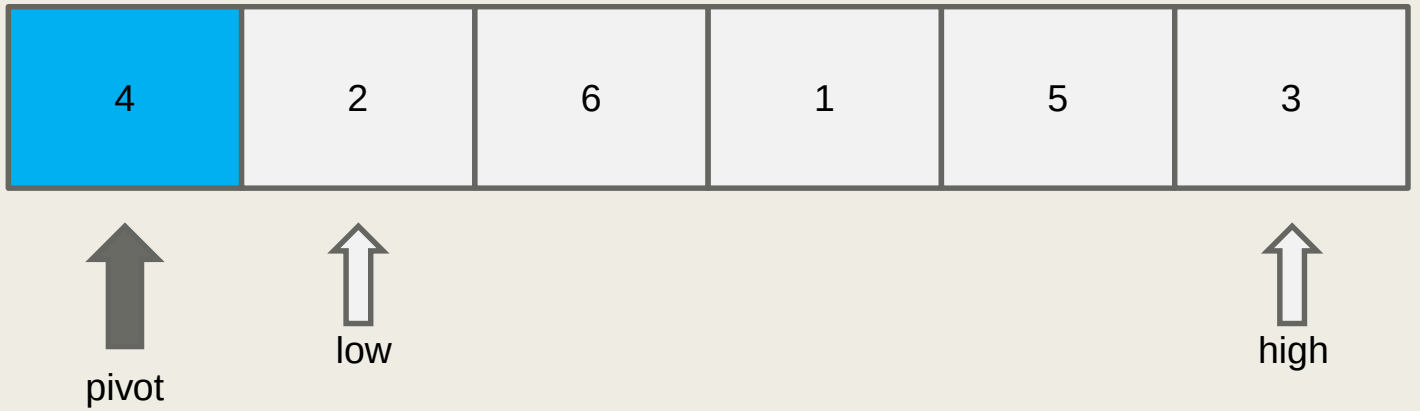
퀵 정렬



퀵 정렬

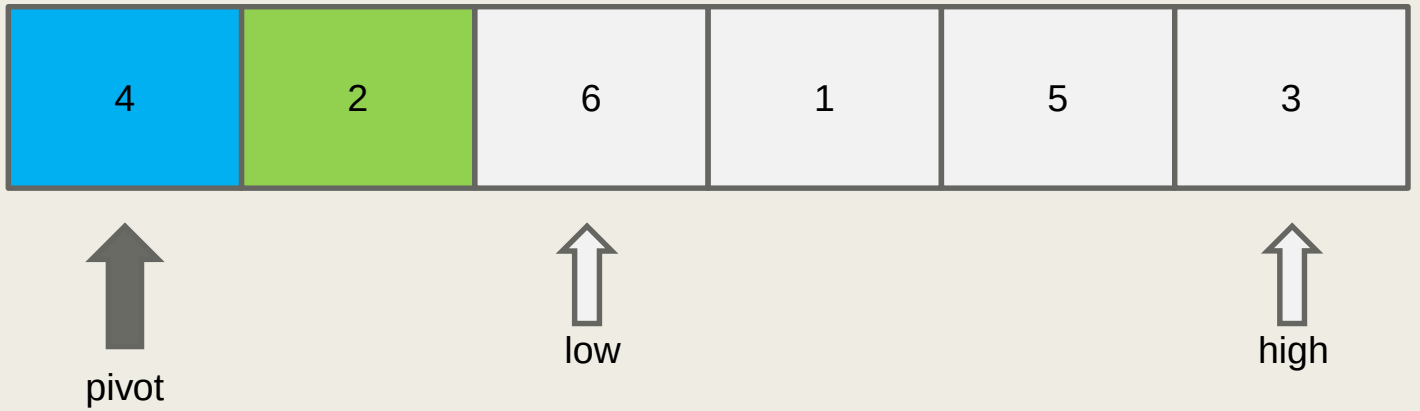


퀵 정렬



low는 pivot보다 큰 값까지 이동, high는 pivot보다 작은 값까지 이동

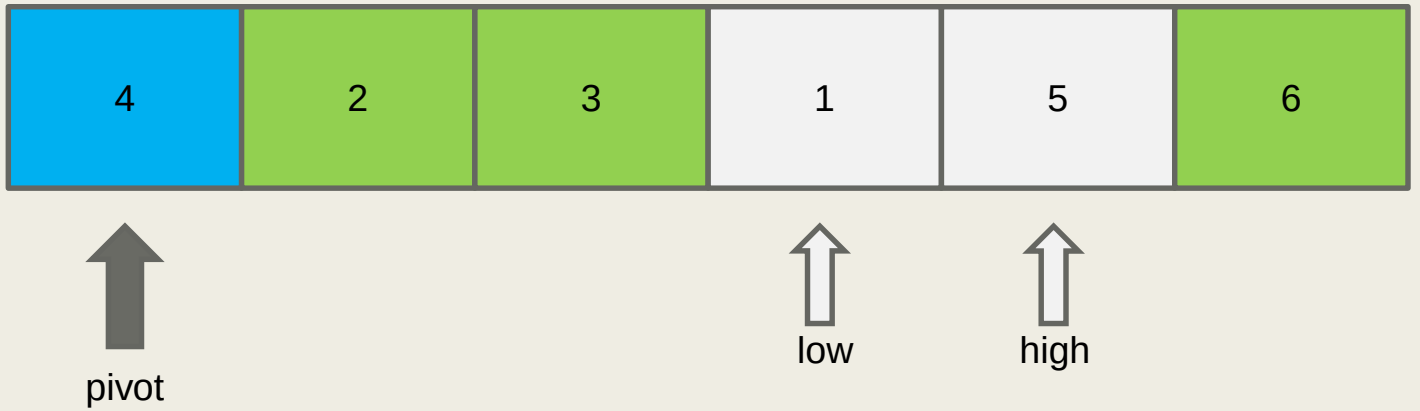
퀵 정렬



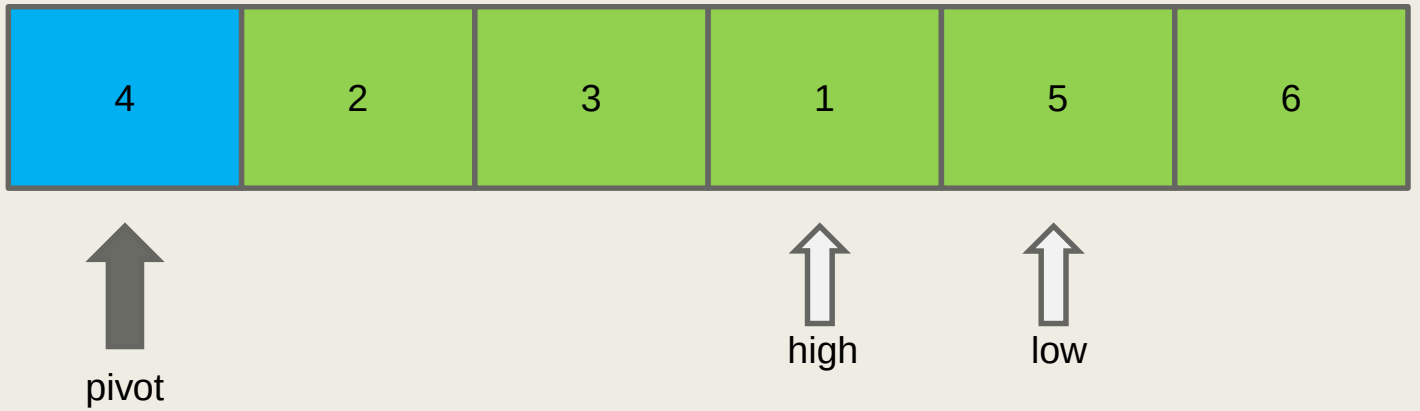
low의 값과 high의 값을 교환

퀵 정렬

퀵 정렬



퀵 정렬



high < low 가 되면 high값과 pivot 값을 바꾸고, 해당 리스트 종료

퀵 정렬

퀵 정렬

pivot의 좌측 리스트

pivot의 우측 리스트

리스트의 크기가 1이나 0이 될 때까지 반복

퀵 정렬

```
1 void quicksort(int low, int high) {
2     if (low >= high)return;
3     int pivot = low;
4     int i = low + 1, j = high, temp;
5     while (i <= j) {
6         while (i <= high && list[i] <= list[pivot]) {
7             i++;
8         }
9         while (j > low && list[j] >= list[pivot]) {
10            j--;
11        }
12        if (i > j) {
13            temp = list[j];
14            list[j] = list[pivot];
15            list[pivot] = temp;
16        }
17        else {
18            temp = list[i];
19            list[i] = list[j];
20            list[j] = temp;
21        }
22    }
23
24    quicksort(low, j - 1);
25    quicksort(j + 1, high)
26 }
```

합병(Merge) 정렬

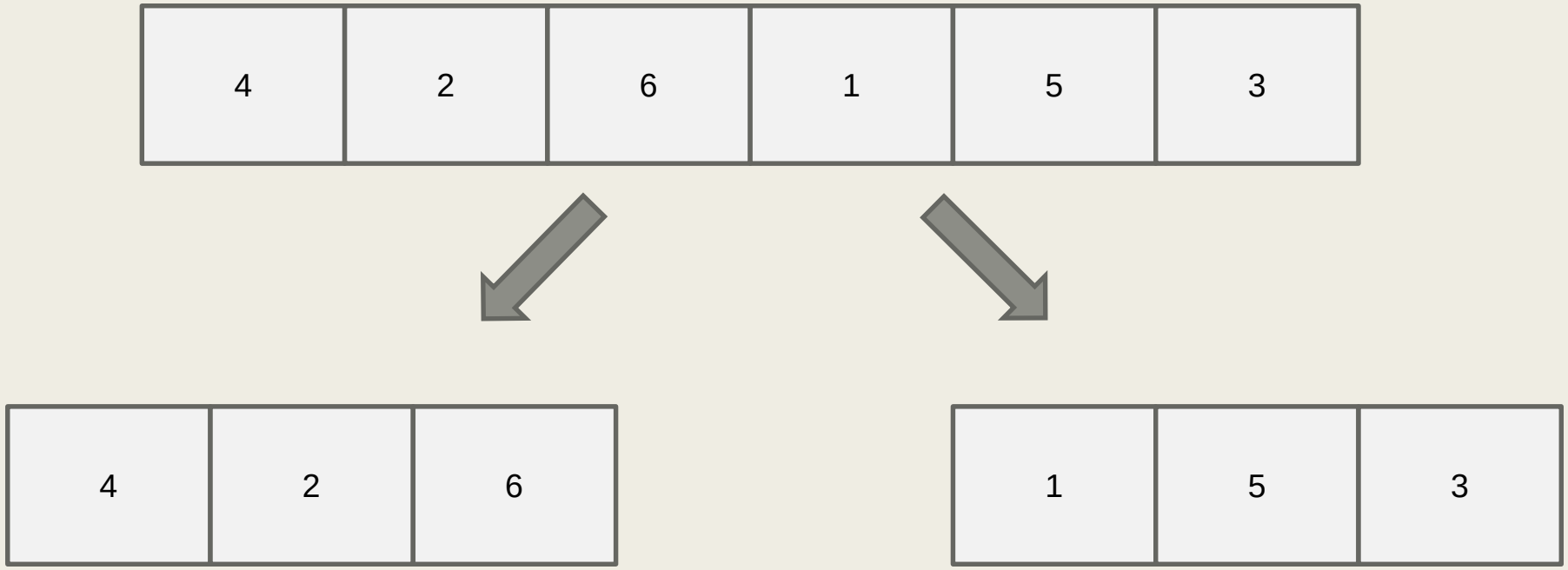
- 정렬되지 않은 리스트의 크기가 1이 될 때까지 절반으로 잘라 나눈다.
- 인접한 두 개의 리스트를 정렬하면서 합친다. (각각 리스트는 정렬되어 있다.)
- 리스트가 모두 합쳐질 때까지 2-3 과정을 반복.

시간복잡도 : 최선 O(nlogn) 평균 O(nlogn) 최악 O(nlogn)

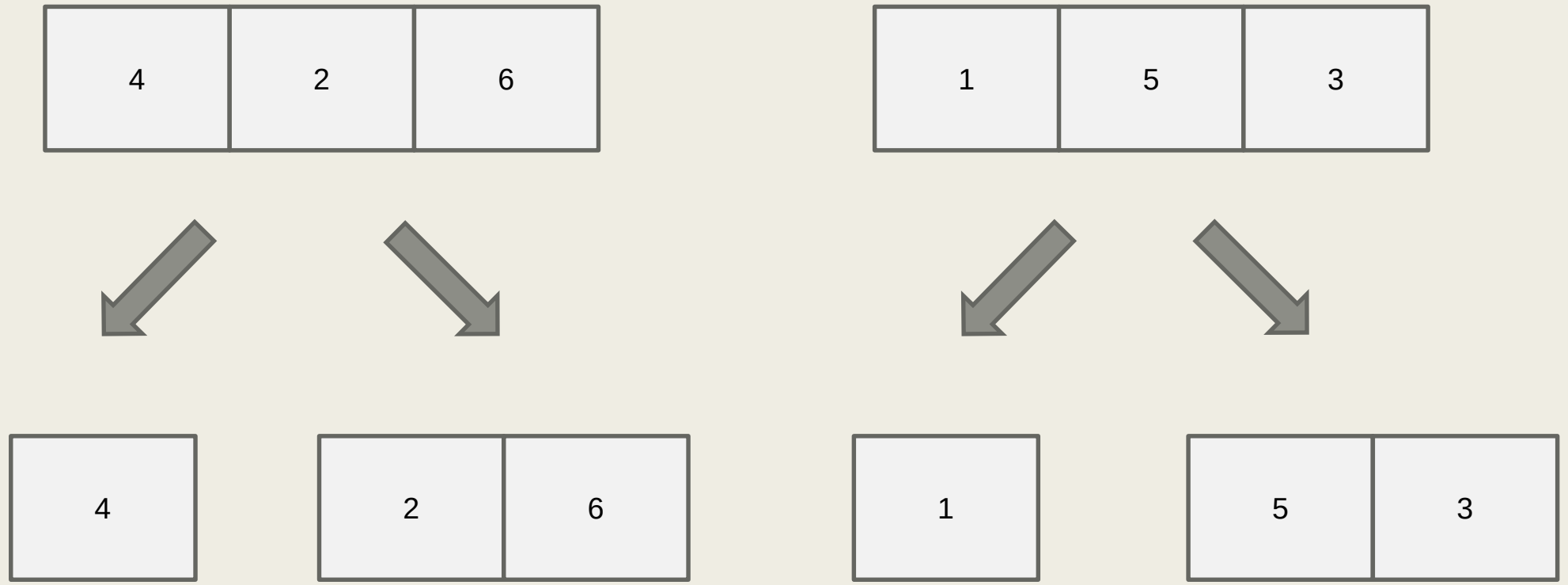
공간복잡도 : O(n)

합병 정렬

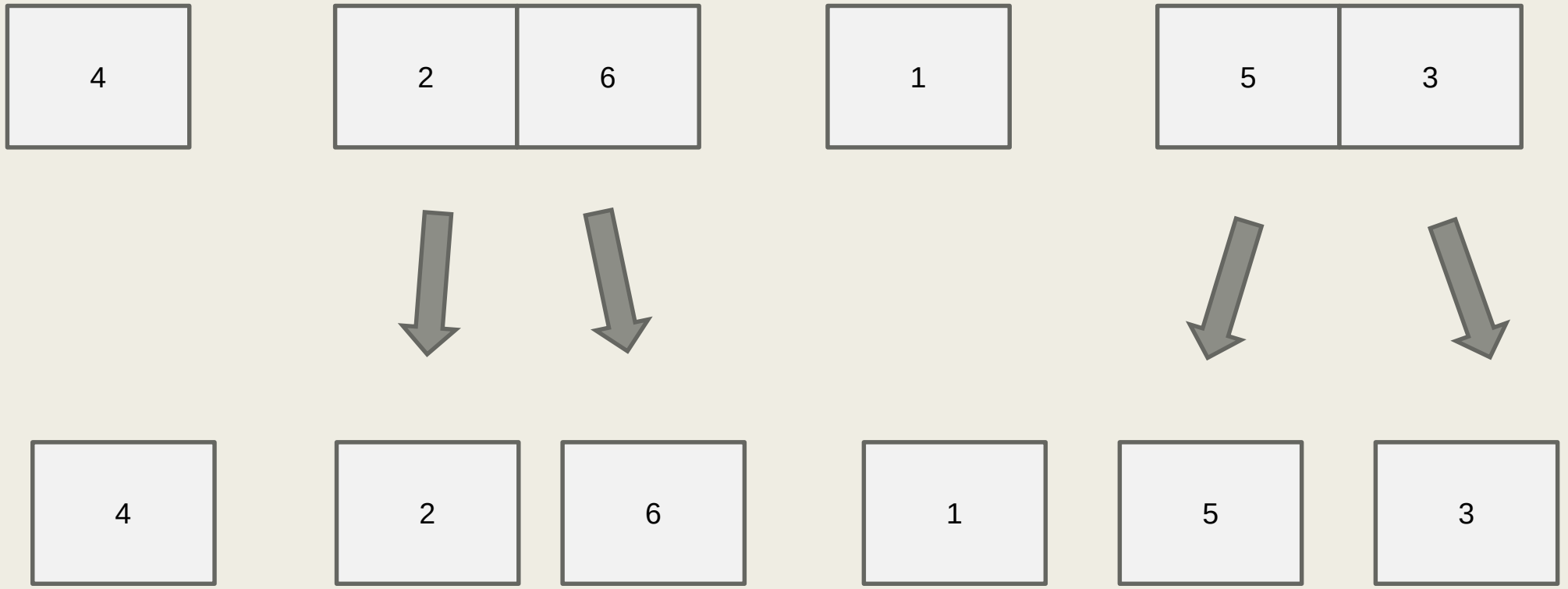
합병 정렬



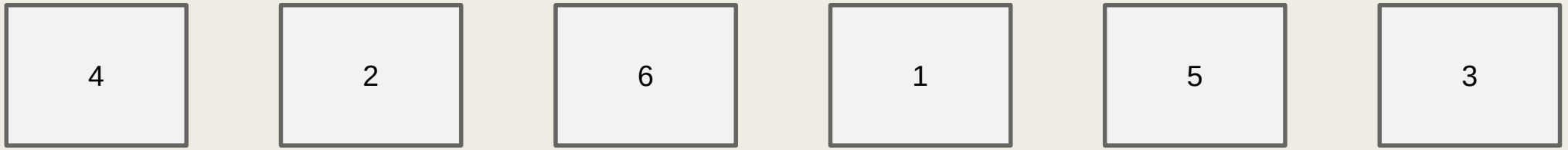
합병 정렬



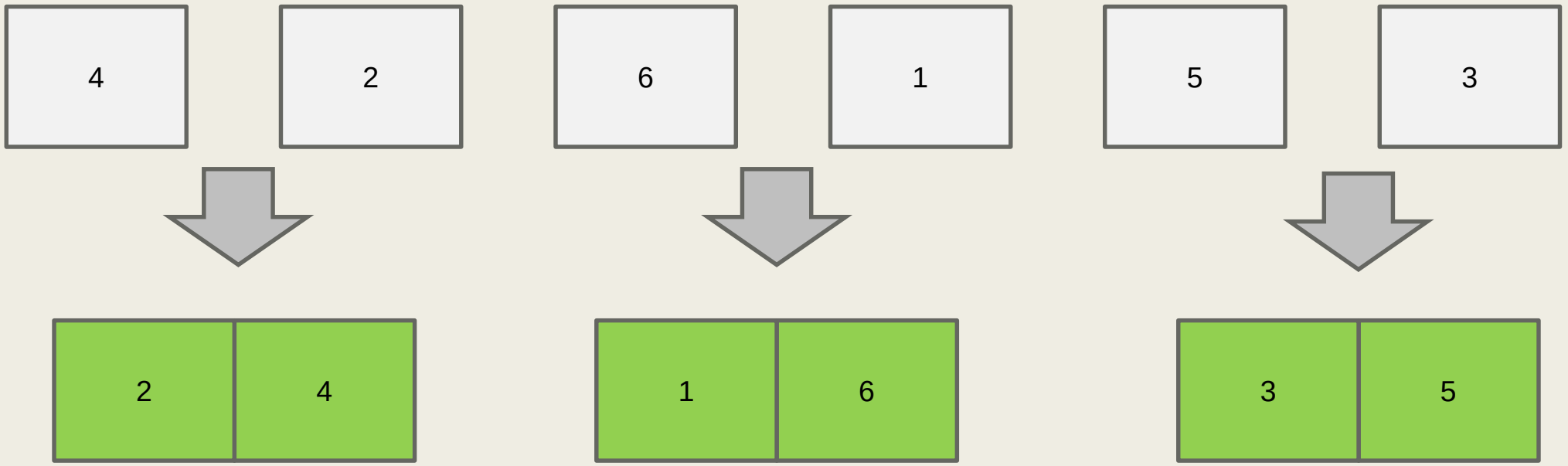
합병 정렬



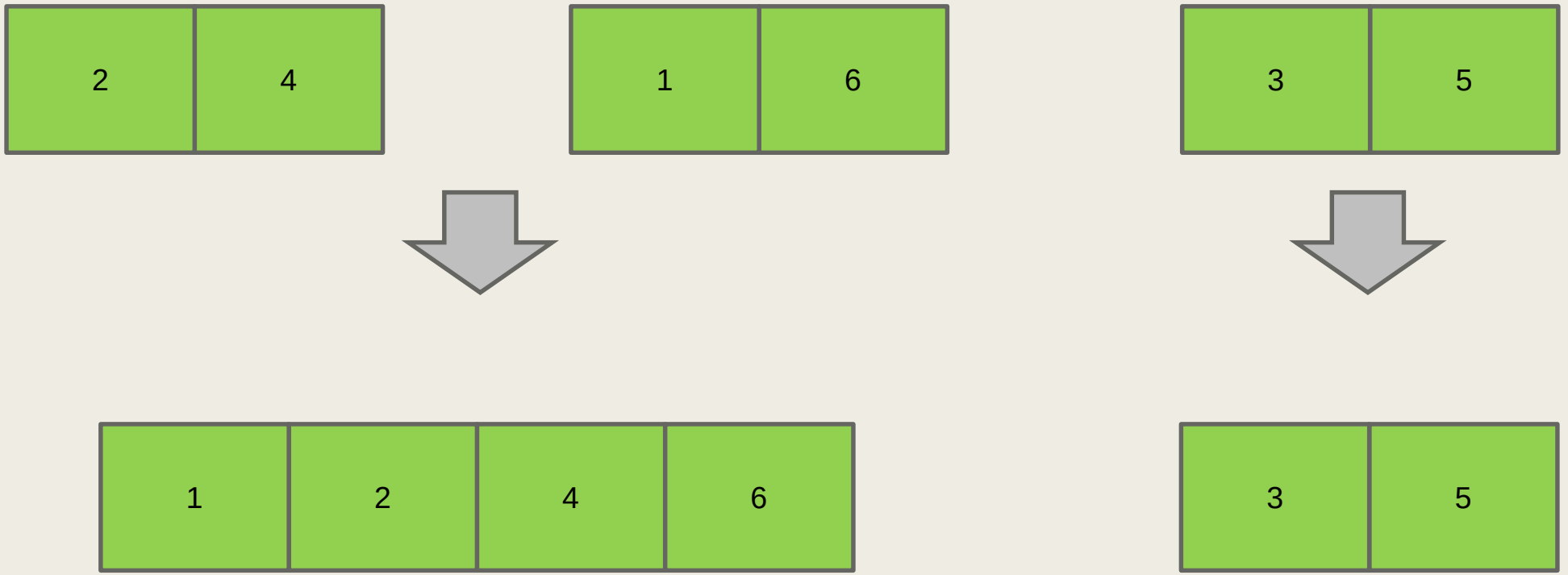
합병 정렬



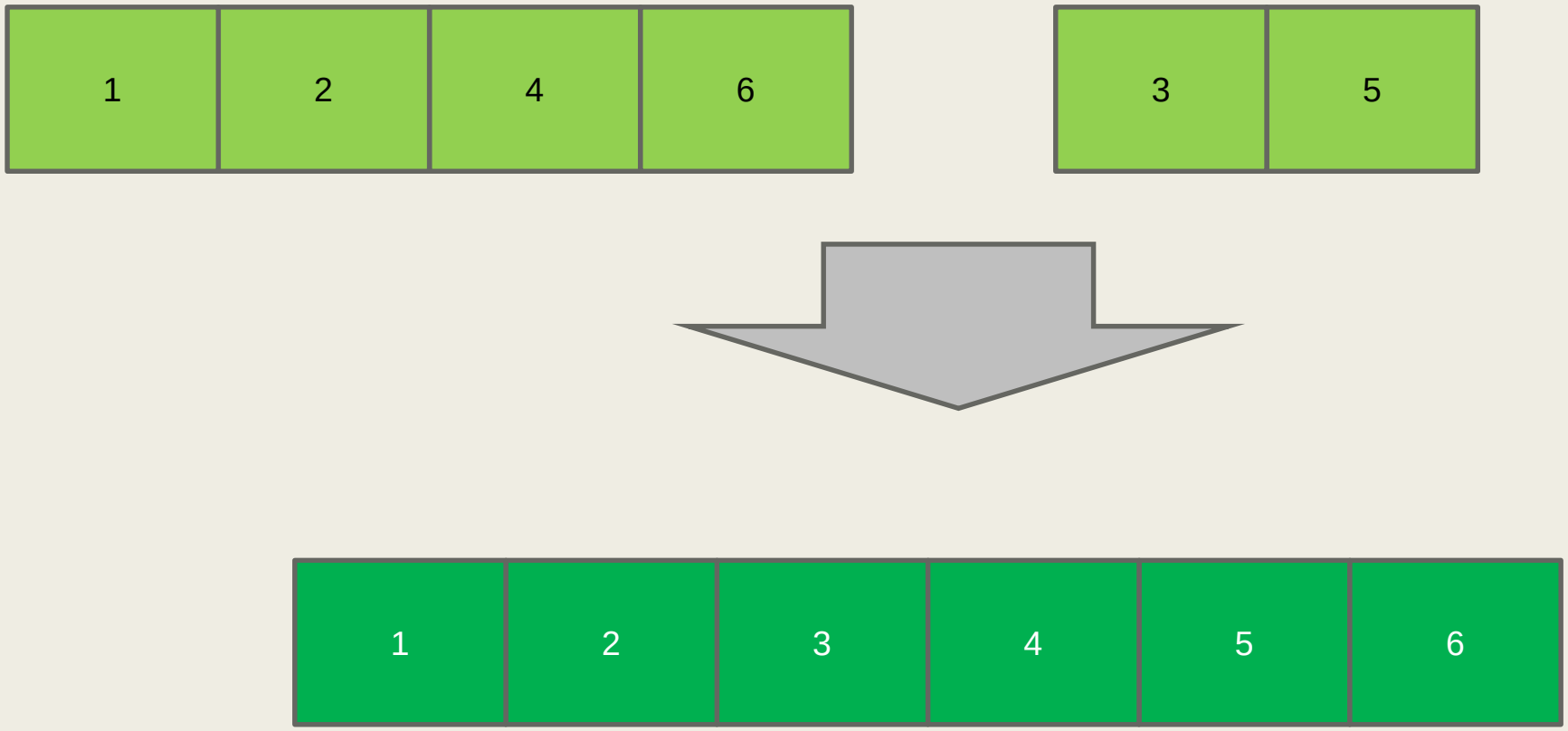
합병 정렬



합병 정렬



합병 정렬



합병 정렬

```
1 void merge(int low, int mid, int high) {
2     int l_idx = low;
3     int r_idx = mid;
4     int t_idx = low;
5     while (l_idx < mid && r_idx < high) {
6         if (arr[l_idx] < arr[r_idx]) tmp[t_idx++] = arr[l_idx++];
7         else tmp[t_idx++] = arr[r_idx++];
8     }
9
10    while (l_idx < mid) tmp[t_idx++] = arr[l_idx++];
11    while (r_idx < high) tmp[t_idx++] = arr[r_idx++];
12
13    for (int i = low; i < high; i++) arr[i] = tmp[i];
14 }
15
16 void merge_sort(int low, int high) {
17    if (low == high - 1) return;
18    int mid = (low + high) / 2;
19    merge_sort(low, mid);
20    merge_sort(mid, high);
21    merge(low, mid, high);
22 }
```

STL

sort 함수　　algorithm 헤더

```
1 template <class RandomAccessIterator>
2   void sort (RandomAccessIterator first, RandomAccessIterator last);
```

기본형(오름차순)

```
1 template <class RandomAccessIterator, class Compare>
2   void sort (RandomAccessIterator first, RandomAccessIterator last, Compare
  comp);
```

직접 비교함수 작성

STL

sort 함수

```
1 struct student {
2     int eyesight, score;
3 };
4
5 bool compare(struct student a, struct student b) {
6     if (a.eyesight != b.eyesight)return a.eyesight < b.eyesight;
7     return a.score > b.score;
8 }
```

이분(Binary) 탐색

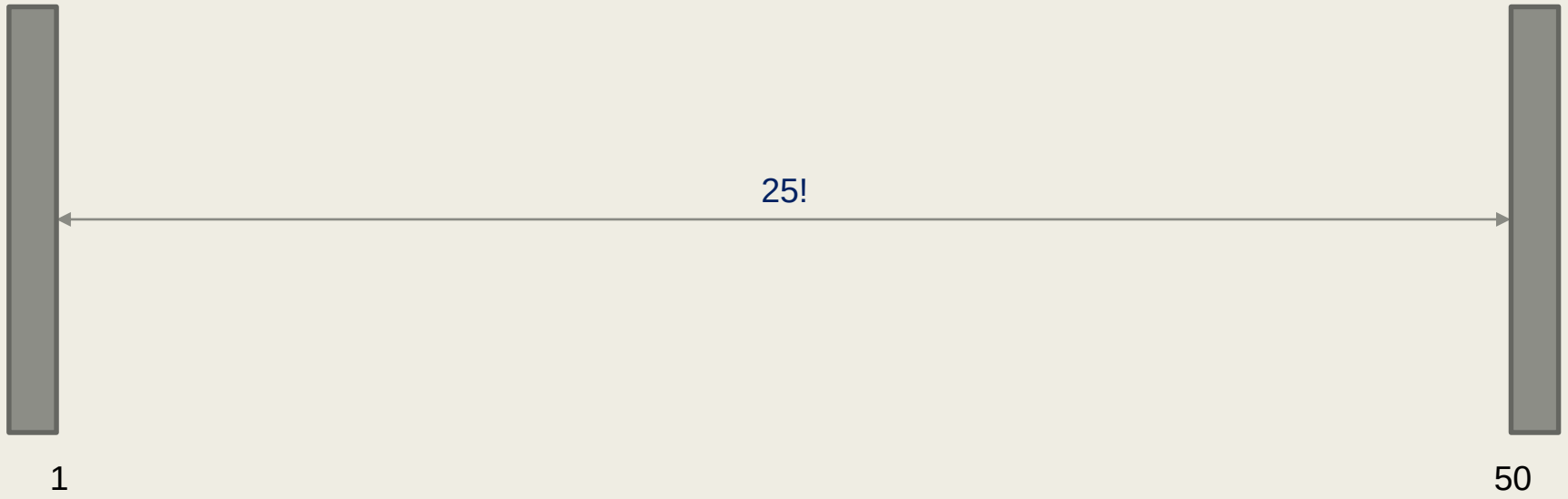
정렬된 리스트에서 특정한 값의 위치를 찾아내는 알고리즘

Ex) 소주병뚜껑 게임

병뚜껑에 1~50 사이의 숫자 하나가 써 있다.
한 사람 씩 숫자를 말하고, 술래가 정답이 그 숫자가 정답의 up/down인지 말해준다.
특정 횟수 이내에 못 맞추면 술래 승리 맞추면 술래 패배

이분 탐색

정답 : 23

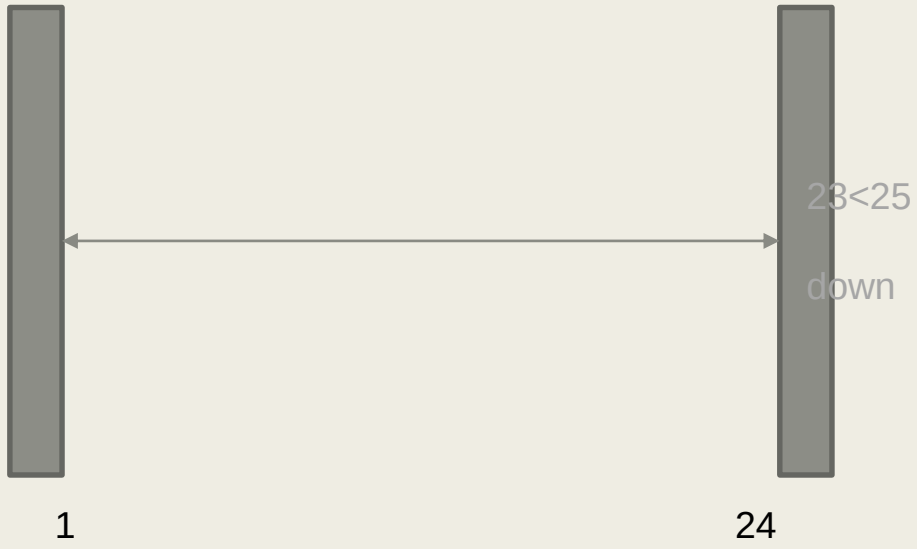


이분 탐색

정답 : 23

이분 탐색

정답 : 23

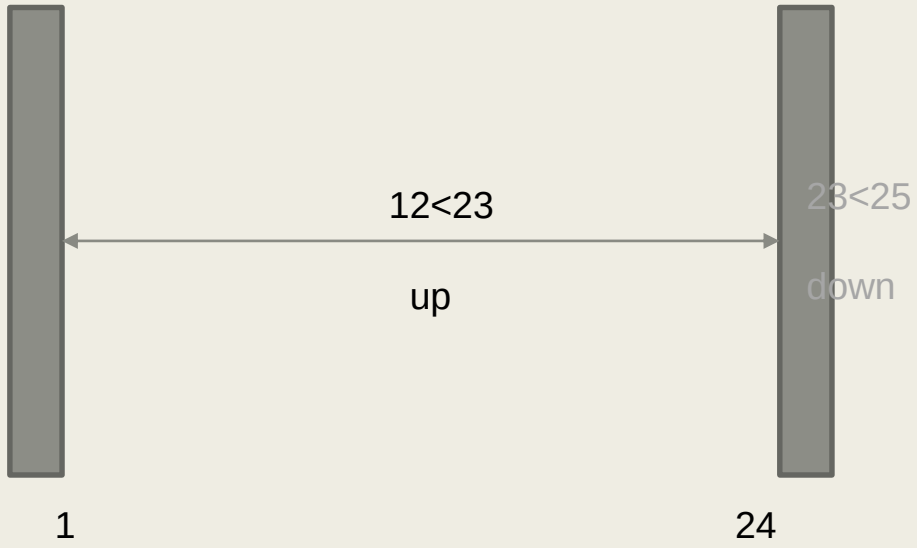


이분 탐색

정답 : 23

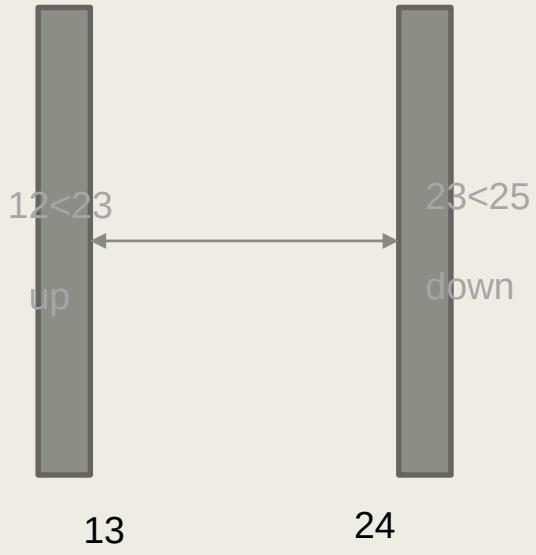
이분 탐색

정답 : 23



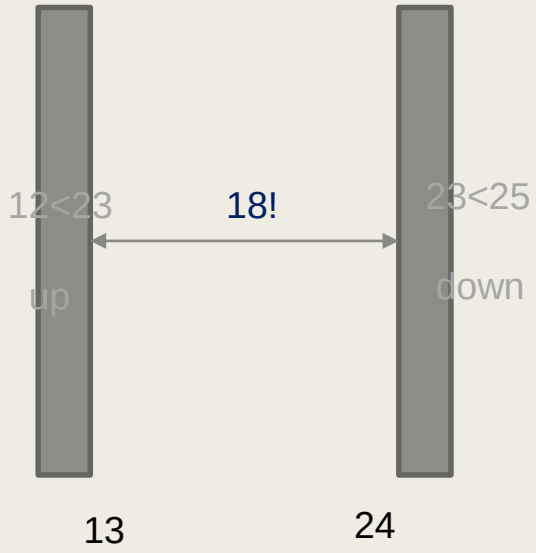
이분 탐색

정답 : 23



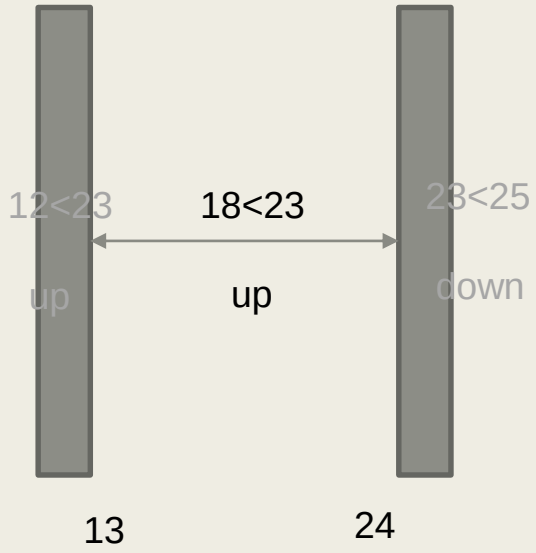
이분 탐색

정답 : 23



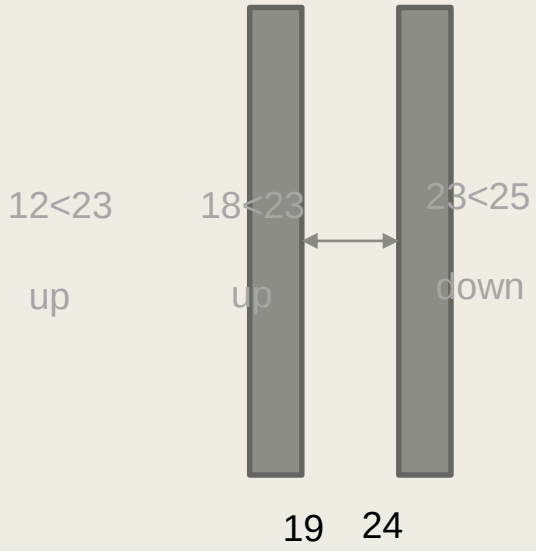
이분 탐색

정답 : 23



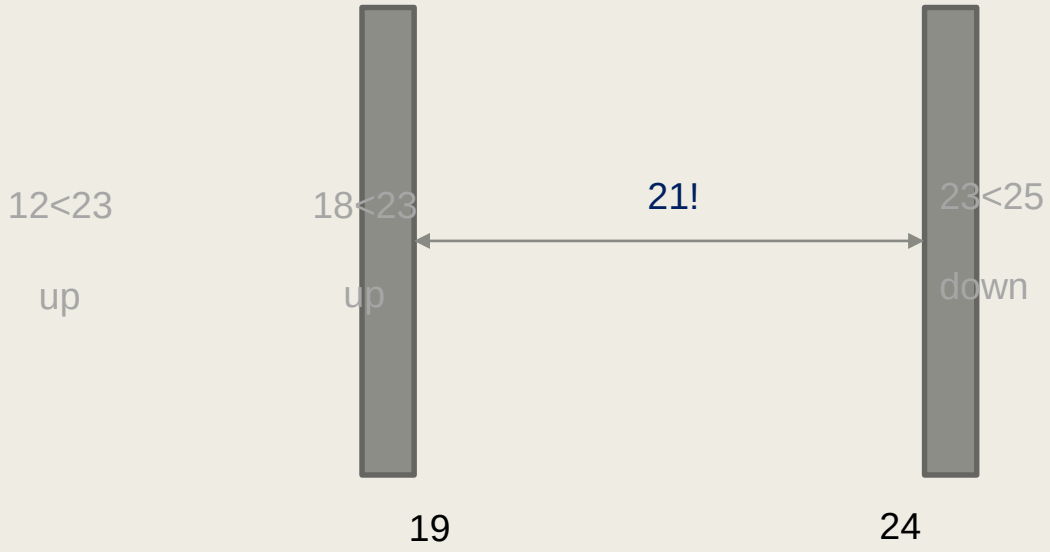
이분 탐색

정답 : 23



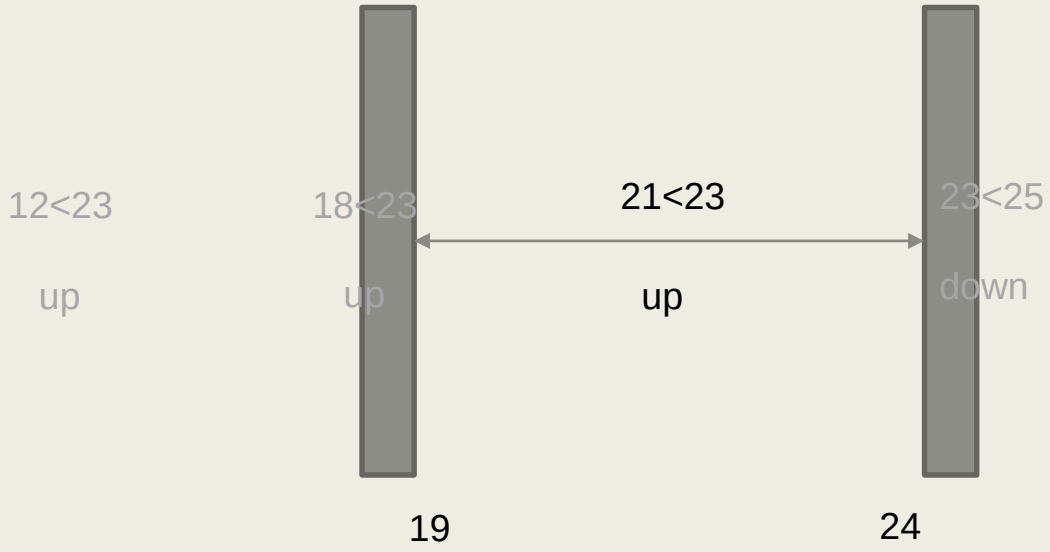
이분 탐색

정답 : 23



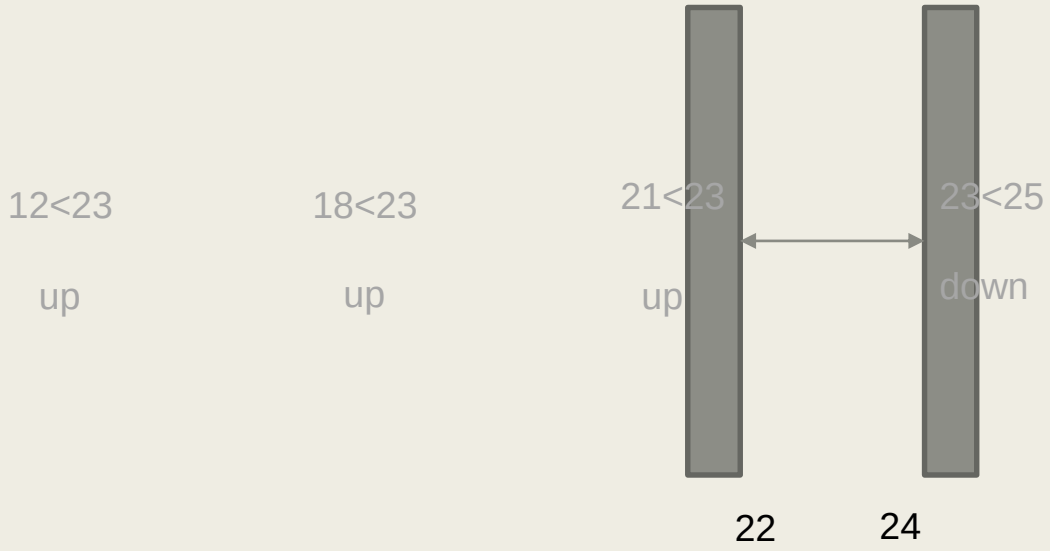
이분 탐색

정답 : 23



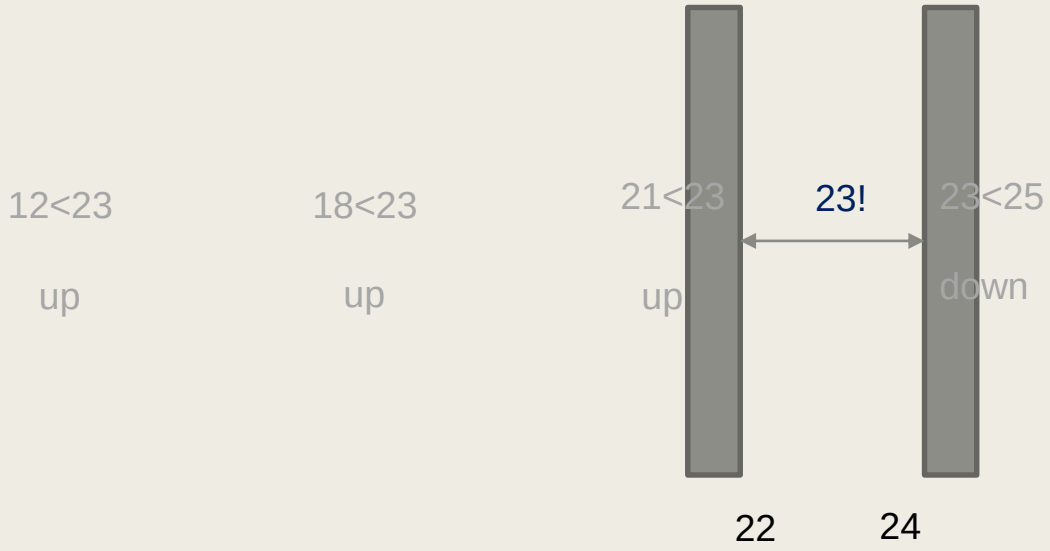
이분 탐색

정답 : 23



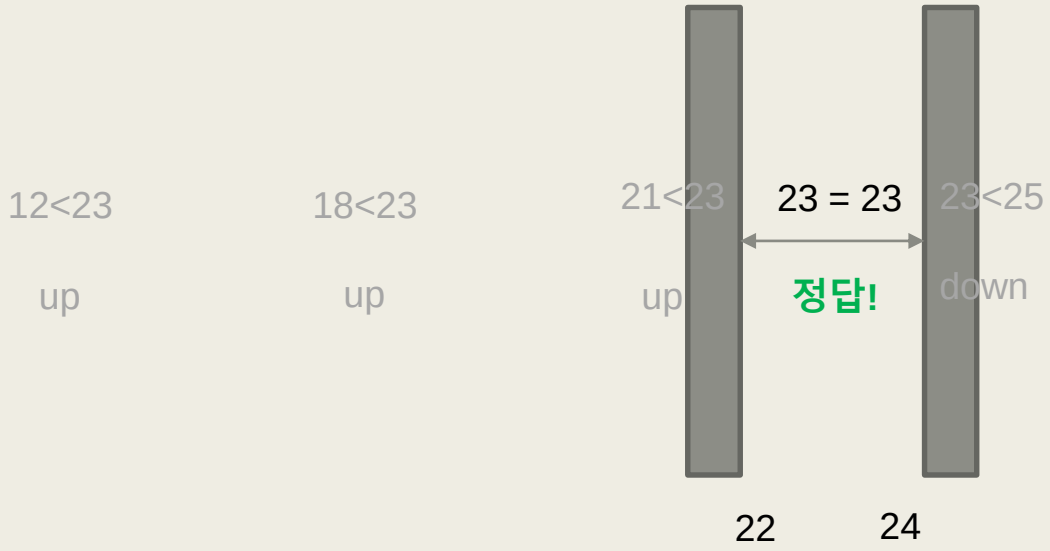
이분 탐색

정답 : 23



이분 탐색

정답 : 23



이분 탐색

```
 1 int binary_search(int answer) {
 2     int high = n;
 3     int low = 1;
 4     int mid;
 5     while (high >= low) {
 6         mid = (high + low) / 2;
 7         if (num[mid] == answer)return mid;
 8         if (num[mid] > answer)high = mid - 1;
 9         else low = mid + 1;
10     }
11     return -1;
12 }
13
```

백준 2805 나무 자르기

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
1 초	256 MB	55973	15976	10038	25.937%

문제

상근이는 나무 M미터가 필요하다. 근처에 나무를 구입할 곳이 모두 망해버렸기 때문에, 정부에 벌목 허가를 요청했다. 정부는 상근이네 집 근처의 나무 한 줄에 대한 벌목 허가를 내주었고, 상근이는 새로 구입한 목재절단기를 이용해서 나무를 구할것이다.

목재절단기는 다음과 같이 동작한다. 먼저, 상근이는 절단기에 높이 H를 지정해야 한다. 높이를 지정하면 톱날이 땅으로부터 H미터 위로 올라간다. 그 다음, 한 줄에 연속해있는 나무를 모두 절단해버린다. 따라서, 높이가 H보다 큰 나무는 H 위의 부분이 잘릴 것이고, 낮은 나무는 잘리지 않을 것이다. 예를 들어, 한 줄에 연속해있는 나무의 높이가 20, 15, 10, 17이라고 하자. 상근이가 높이를 15로 지정했다면, 나무를 자른 뒤의 높이는 15, 15, 10, 15가 될 것이고, 상근이는 길이가 5인 나무와 2인 나무를 들고 집에 갈 것이다. (총 7미터를 집에 들고 간다) 절단기에 설정할 수 있는 높이는 양의 정수 또는 0이다.

상근이는 환경에 매우 관심이 많기 때문에, 나무를 필요한 만큼만 집으로 가져가려고 한다. 이때, 적어도 M미터의 나무를 집에 가져가기 위해서 절단기에 설정할 수 있는 높이의 최댓값을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 나무의 수 N과 상근이가 집으로 가져가려고 하는 나무의 길이 M이 주어진다. ($1 \le N \le 1,000,000$, $1 \le M \le 2,000,000,000$)

둘째 줄에는 나무의 높이가 주어진다. 나무의 높이의 합은 항상 M보다 크거나 같기 때문에, 상근이는 집에 필요한 나무를 항상 가져갈 수 있다. 높이는 1,000,000,000보다 작거나 같은 양의 정수 또는 0이다.

출력

적어도 M미터의 나무를 집에 가져가기 위해서 절단기에 설정할 수 있는 높이의 최댓값을 출력한다.

백준 2805 나무 자르기

백준 2805 나무 자르기

```cpp
#include<iostream>
#include<algorithm>
using namespace std;
int num[1000005];
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    int n, m;
    cin >> n >> m;
    int high=0, low = 0, mid;
    for (int i = 0; i < n; i++) {
        cin >> num[i];
        high = max(high, num[i]);
    }
    long long sum;
    int ans=-1;
    while (high >= low) {
        mid = (high + low) / 2;
        sum = 0;
        for (int i = 0; i < n; i++) {
            if (num[i] - mid > 0)sum += num[i] - mid;
        }
        if (sum < m) high = mid - 1;
        else {
            ans = max(ans,mid);
            low = mid + 1;
        }
    }
    cout << ans;
}
```

백준 3079 입국심사

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
1 초	128 MB	8128	1670	1116	23.867%

문제

상근이와 친구들은 오스트레일리아로 여행을 떠났다. 상근이와 친구들은 총 M명이고, 지금 공항에서 한 줄로 서서 입국심사를 기다리고 있다. 입국심사대는 총 N개가 있다. 각 입국심사관이 심사를 하는데 걸리는 시간은 사람마다 모두 다르다. k번 심사대에 앉아있는 심사관이 한 명을 심사를 하는데 드는 시간은 T_k이다.

가장 처음에 모든 심사대는 비어있고, 심사를 할 준비를 모두 끝냈다. 상근이와 친구들은 비행기 하나를 전세내고 놀러갔기 때문에, 지금 심사를 기다리고 있는 사람은 모두 상근이와 친구들이다. 한 심사대에서는 한 번에 한 사람만 심사를 할 수 있다. 가장 앞에 서 있는 사람은 비어있는 심사대가 보이면 거기로 가서 심사를 받을 수 있다. 하지만 항상 이동을 해야 하는 것은 아니다. 더 빠른 심사대의 심사가 끝나길 기다린 다음에 그 곳으로 가서 심사를 받아도 된다.

상근이와 친구들은 모두 컴퓨터 공학과 학생이기 때문에, 어떻게 심사를 받으면 모든 사람이 심사를 받는데 걸리는 시간이 최소가 될지 궁금해졌다.

예를 들어, 두 심사대가 있고, 심사를 하는데 걸리는 시간이 각각 7초와 10초라고 하자. 줄에 서 있는 사람이 6명이라면, 가장 첫 두 사람은 즉시 심사를 받으러 가게 된다. 7초가 되었을 때, 첫 번째 심사대는 비어있게 되고, 세 번째 사람이 그곳으로 이동해서 심사를 받으면 된다. 10초가 되는 순간, 네 번째 사람이 이곳으로 이동해서 심사를 받으면 되고, 14초가 되었을 때는 다섯 번째 사람이 첫 번째 심사대로 이동해서 심사를 받으면 된다. 20초가 되었을 때, 두 번째 심사대가 비어있게 된다. 하지만, 여섯 번째 사람이 그 곳으로 이동하지 않고, 1초를 더 기다린 다음에 첫 번째 심사대로 이동해서 심사를 받으면, 모든 사람이 심사를 받는데 걸리는 시간이 28초가 된다. 만약, 마지막 사람이 1초를 더 기다리지않고, 첫 번째 심사대로 이동하지 않았다면, 모든 사람이 심사를 받는데 걸리는 시간이 30초가 되게 된다.

상근이와 친구들이 심사를 받는데 걸리는 시간의 최솟값을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N과 M이 주어진다. ($1 \le N \le 100,000$, $1 \le M \le 1,000,000,000$)

다음 N개 줄에는 각 심사대에서 심사를 하는데 걸리는 시간인 T_k가 주어진다. ($1 \le T_k \le 10^9$)

출력

첫째 줄에 상근이와 친구들이 심사를 마치는데 걸리는 시간의 최솟값을 출력한다.

백준 3079 입국심사

백준 3079 입국심사

```cpp
#include<iostream>
#include<algorithm>
using namespace std;
#define MAX 100005

long long N, M, low, high, mid, sum, answer;
int line[MAX];

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cin >> N >> M;
    for (int i = 0; i < N; i++) {
        cin >> line[i];
    }
    high = 1e18;
    low = 1;
    while (high >= low) {
        mid = (high + low) / 2;
        long long sum = 0;
        for (int i = 0; i < N; i++) {
            sum += mid / line[i];
            if (sum >= M)break;
        }
        if (sum >= M) {
            answer = mid;
            high = mid - 1;
        }
        else {
            low = mid + 1;
        }
    }
    cout << answer << "\n";
}
```

STL

upper_bound & lower_bound

```
1 template <class ForwardIterator, class T>
2   ForwardIterator upper_bound (ForwardIterator first, ForwardIterator last,
  const T& val)
```

upper_bound : 찾고자 하는 값을 최초로 초과하는 값 index의 주소를 반환

```
1 template <class ForwardIterator, class T>
2   ForwardIterator lower_bound (ForwardIterator first, ForwardIterator last,
  const T& val)
```

lower_bound : 찾고자 하는 값보다 크거나 같은 최초의 값 index의 주소를 반환

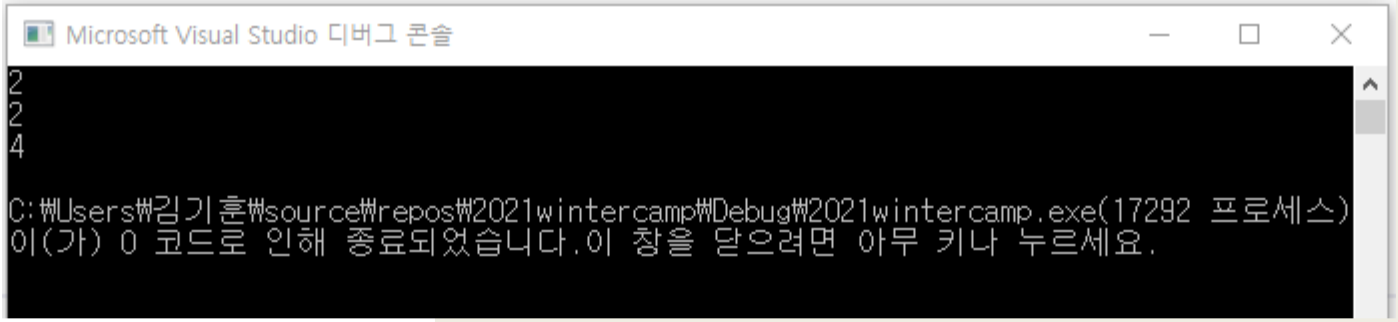
STL

upper_bound & lower_bound

```cpp
#include<iostream>
#include<algorithm>
using namespace std;

int arr[6] = { 1,2,5,7,8,10 };

int main() {
    int index = lower_bound(arr, arr + 6, 3) - arr;
    cout << index << '\n';
    index = lower_bound(arr, arr + 6, 5) - arr;
    cout << index << '\n';
    index = upper_bound(arr, arr + 6, 7) - arr;
    cout << index << '\n';
}
```



필수문제

- A - 수 정렬하기
- B - 수 정렬하기 4
- C - 수 찾기
- D - 입국심사
- E - 과자 나눠주기

연습문제

- A - 보물
- B - 단어 정렬
- C - 수 정렬하기 2
- D - KCPC
- E - 30
- F - 수 정렬하기 3
- G - 좌표 정렬하기
- H - 랜선 자르기
- I - 공유기 설치
- J - 기타 레슨
- K - 두 용액
- L - 세 용액
- M - 예산
- N - 나무 자르기
- O - 가장 긴 증가하는 부분 수열 3
- P - 이상한 술집
- Q - Ax+Bsin(x)=C ②

피드백 및 질의응답

감사합니다!