**이진 탐색(Binary Search)이란?**

- 순차 탐색이 간단하고 작은 배열의 탐색에 효과적이라고 한다면

- 큰 배열의 탐색에 적합한 탐색 기법.

- 배열의 중앙에 있는 값을 조사 → 찾고자 하는 값이 왼쪽인지 오른쪽인지 조사 → 탐색의 범위를 반으로 줄임

- 위의 메커니즘을 계속해서 반복하여 원하는 값을 적은 횟수로 찾아낼 수 있음. (술게임에서 Up & Down을 생각하면 편하다)

- 시간 복잡도 O(logn)

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | int binary\_search(int value, int left, int right) {        int mid;        // left <= right의 이유?      // left < right로 할 경우, 원소가 하나 남았을 때는 value값과 비교하지 않고 종료할 수 있기 때문에 이를 방지.      while (left <= right) {          mid = (left + right) / 2;          if (value == arr[mid]) return mid;          else if (value > arr[mid]) left = mid + 1;          else right = mid - 1;      }      return -1;  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

**\*Upper Bound, Lower Bound란**

- BS(Binary Search)를 활용한 알고리즘.

- 원하는 값을 찾지 못했을 때 -1을 반환하는 BS와는 달리 Upper Bound는 원하는 값을 초과하는 첫 번째 위치, Lower Bound는 원하는 값 이상의 첫 번째 위치를 반환한다.

**Upper Bound(상계) : Key보다 큰 가장 첫 번째 값을 반환**

**Lower Bound(하계) : Key보다 크거나 같은 가장 첫 번째 값을 반환.**

말로 설명하는 것보다, 예제로 설명하는 것이 더 빠를 것 같아서 준비했다.

**arr[10] = 1 1 2 3 4 4 5 5 6 8**

위의 배열에서

Key = 1인 경우 Lower Bound : 1, Upper Bound : 2

Key = 3인 경우 Lower Bound : 4, Upper Bound : 4

**Upper Bound Binary Search 구현**

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | int binary\_search\_Upper\_Bound(int value, int left, int right) {        int mid;        // Upper Bound는 값을 찾지 못해도 value보다 큰 가장 첫 번째 값을 반환하면 된다.      // 그래서 기존 BS와는 다른점이 등호가 사라짐.      while (left < right) {          mid = (left + right) / 2;            // 이 부분이 핵심. 찾고자 하는 key값과 배열의 값이 같아도 인덱스를 한 칸 밀어주는게 핵심.          if (arr[mid] <= value) left = mid + 1;          else right = mid;      }      return right;  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

**Lower Bound Binary Search 구현**

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | int binary\_search\_Lower\_Bound(int value, int left, int right) {        int mid;        while (left < right) {          mid = (left + right) / 2;            if (arr[mid] < value) left = mid + 1;          else right = mid;      }      return right;  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |