

# TermProject

자료구조 및 알고리즘1

## <소스코드 (TermProject2.java)>

```
import java.util.Random;
import java.util.Scanner;

public class TermProject2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Random rand = new Random( seed: 100);

        int n = sc.nextInt(); // n의 값 입력
        int[] arr = new int[n+1];
        int min, max, sum;

        for (int i = 1; i <= n; i++) {
            arr[i] = rand.nextInt( bound: 1000000) + 1; // data의 범위 1 ~ 1000000
            System.out.print(arr[i]+" "); // data 값 출력
        }
        System.out.println();

        int k = sc.nextInt(); // k 값 입력
        for (int i = 0; i < k; i++) {
            int start = rand.nextInt(n) + 1;
            int end = rand.nextInt(n) + 1;

            if (start > end) {
                int tmp = start;
                start = end;
                end = tmp;
            }

            max = min = arr[start];
            sum = 0;
            for (int i = start; i <= end; i++) {
                if(arr[i] < min) min = arr[i];
                if(arr[i] > max) max = arr[i];
                sum += arr[i];
            }
            System.out.println(i+1 + "구간 = (" + start + ", " + end + ")");
            System.out.println("Min: " + min + " Max: " + max + " Sum: " + sum + "\n");
        }
    }
}
```

- N을 입력하고, N개의 데이터를 랜덤으로 생성 (랜덤값 범위 : (int) 1 ~ 1,000,000 )
- (랜덤으로 생성한 N개의 데이터를 출력)
- K를 입력하고, K개의 구간을 생성 =>  $(a_1, b_1), (a_2, b_2), \dots, (a_K, b_K)$
- $(i = 1, 2, 3, \dots, K)$ ,  $(a_i, b_i)$ 은 각각 랜덤으로 생성하며  $a_i \leq b_i$
- K개의 구간에 대해서 각각 min, max, sum 값을 구하고, 출력


## <소스코드 측정부분 (TimeTest.java)>

```
long total_time = 0;
for (int i = 0; i < k; i++) {
    int start = rand.nextInt(n) + 1;
    int end = rand.nextInt(n) + 1;

    if (start > end) {
        int tmp = start;
        start = end;
        end = tmp;
    }

    max = min = arr[start];
    sum = 0;

    long t = System.currentTimeMillis();
    for (int i = start; i <= end; i++) {
        if(arr[i] < min) min = arr[i];
        if(arr[i] > max) max = arr[i];
        sum += arr[i];
    }
    long elapsed = System.currentTimeMillis() - t;
    total_time += elapsed;
    System.out.println(j+1 + "구간 = (" + start + ", " + end + ")");
    System.out.println("Min: " + min + " Max: " + max + " Sum: " + sum + "\n");
}
System.out.println("Elapsed : " + total_time + "ms");
```

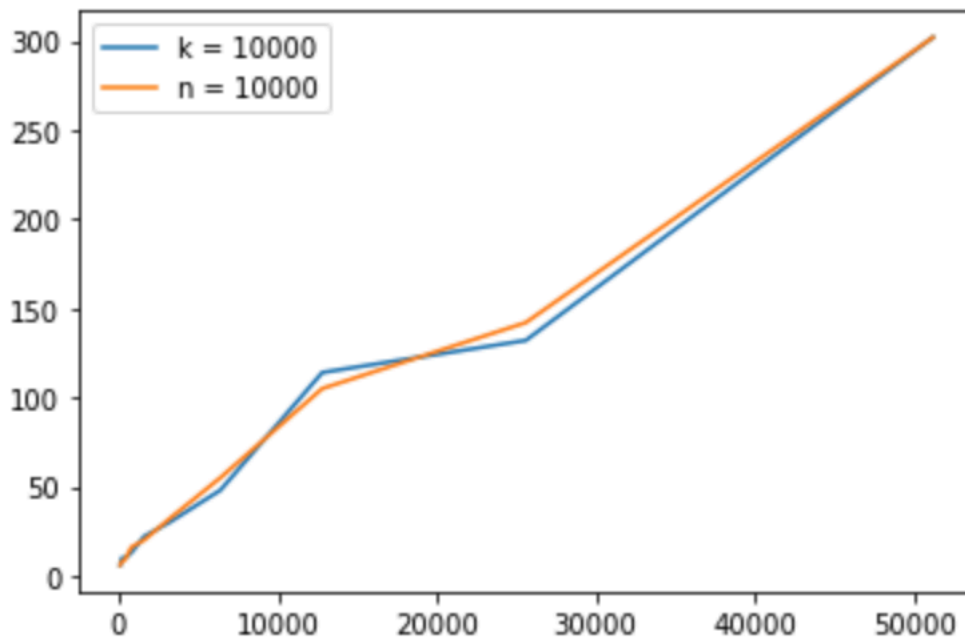


- N개 데이터 생성, 랜덤 구간 K개 생성 시간은 포함하지 않았습니다.
- K개 구간에서 min, max, sum 값을 구하는 과정을 더했습니다.

## < 그래프 및 성능 평가 (그래프.ipynb)>

K=10,000 고정 후 N 값 변화 (파란색)

N=10,000 고정 후 K 값 변화 (주황색)



**K** 시간(ms)

100	6
200	8
400	9
800	16
1600	20
3200	32
6400	55
12800	105
25600	142
51200	302

**N** 시간(ms)

100	6
200	10
400	10
800	13
1600	22
3200	30
6400	48
12800	114
25600	132
51200	302

K를 고정 시킨 경우와 N을 고정 시킨 경우 비슷한 성능이 나온다.

단순 for 문으로 min, max, sum을 구할때

각 구간  $(a_i, b_i)$ 가  $a_i = b_i$  인 경우 BestCase  $\rightarrow O(K)$

- For문 1번 \* K번 = K

각 구간  $(a_i, b_i)$ 가  $(1, N)$  인 경우 WorstCase  $\rightarrow O(KN)$

- For문 N번 \* K번 = KN

최악의 경우를 생각하면  $N*K$  번의 연산을 실행한다고 볼 수 있다.

데이터의 수(N), 구간의 수(K)가 적다면 크게 문제되지 않는다.

하지만 데이터의 수가  $10^5$ , 구간의 수가  $10^5$  이면 최악의 경우에는  $10^{10}$  만큼 어마어마한 연산의 횟수를 가진다.

세그먼트 트리 (Segment Tree)를 이용하면 성능개선 가능.