ICPC Sinchon Advanced #1

숭실대학교 컴퓨터학부 나정휘

https://justiceHui.github.io/

목차

- 커리큘럼 소개
- 재귀 함수
- 수학적 귀납법
- 분할 정복

커리큘럼 소개

- 커리큘럼
  - 1회차 분할 정복
  - 2회차 세그먼트 트리
  - 3회차 모노톤 스택/큐
  - 4회차 DP (위상정렬, 트리, 구간, 기댓값 등)
  - 5회차 기초 정수론 (확장 유클리드, 페르마 소정리, 중국인의 나머지 정리 등)
  - 6회차 문제 풀이 1
  - 7회차 문자열 (해싱, 트라이, KMP)
  - 8회차 DP (그리디+DP, 게임, 자료구조 등)
  - 9회차 그래프 (DFS Tree, SCC, 2-SAT)
  - 10회차 트리 (오일러 투어 테크닉, LCA 등)
  - 11회차 문제 풀이 2

질문?

재귀함수

재귀함수

- 재귀함수: 자기 자신을 호출하는 함수
  - ex. f(n) = f(n-1) + f(n-2)
  - 점화식 계산, 완전 탐색, 반복문으로 반복하기 힘든 작업, ...

  - 어려움
  - 평소와 다른 방식으로 생각해야 함
    - 귀납적 사고
    - 익숙해지는데 2년 걸림

재귀함수 – 예시 1

- 배열 A의 모든 원소의 합을 구하는 작업
  - f(l, r) : A[l] + A[l+1] + … + A[r]을 구하는 함수
  - f(0, N-1)을 구해야 함

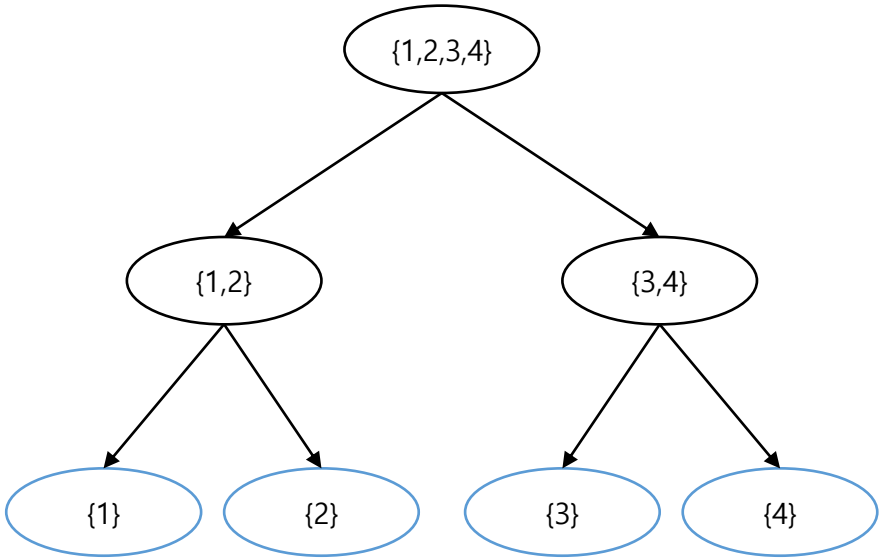
  - 일을 혼자 하는 건 어려우니까 직원 2명을 고용해서
  - 배열을 반으로 잘라서 나눠주면?
  - f(l, r) = f(l, (l+r)/2) + f((l+r)/2+1, r)

재귀함수 – 예시 1

- 배열 A의 모든 원소의 합을 구하는 작업
  - f(l, r) : A[l] + A[l+1] + … + A[r]을 구하는 함수

  - 일을 혼자 하는 건 어려우니까 직원 2명을 고용해서
  - 배열을 반으로 잘라서 나눠주면?
  - f(l, r) = f(l, (l+r)/2) + f((l+r)/2+1, r)

  - 재귀 호출 과정을 살펴보면…

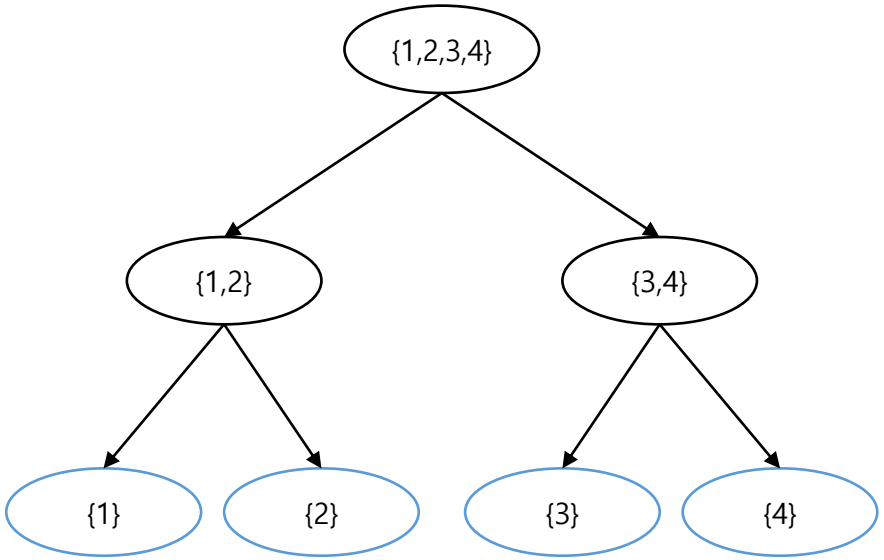


재귀함수 – 예시 1

- 배열 A의 모든 원소의 합을 구하는 작업
  - f(l, r) : A[l] + A[l+1] + … + A[r]을 구하는 함수

  - 일을 혼자 하는 건 어려우니까 직원 2명을 고용해서
  - 배열을 반으로 잘라서 나눠주면?
  - f(l, r) = f(l, (l+r)/2) + f((l+r)/2+1, r)

  - 재귀 호출 과정을 살펴보면…
    - 이렇게 하지 마세요
  - 함수 f가 올바른 결과를 반환한다고 “믿고”
  - f(l, r)에서는 f(l, (l+r)/2) + f((l+r)/2+1, r)을 반환하면 됨
    - 수학적 귀납법



재귀함수 – 예시 2

- 배열 A를 정렬하는 작업
  - f(l, r) = A[l..r]을 정렬된 상태로 만드는 함수
  - l = r이면 이미 정렬된 상태 (종료 조건)

  - f(l, m)과 f(m+1, r)을 호출
    - m = (l+r)/2

  - A[l..m]과 A[m+1..r]이 정렬되어 있다고 "믿고"
  - 정렬된 두 배열을 잘 합치면 됨

재귀함수

- 재귀호출
  - 자신과 동일한 일을 하는 부하 직원에게 작업을 요청하고 그 결과물을 돌려받는 과정
  - 부하 직원의 결과물이 항상 올바르다고 믿고 작업을 수행하면 됨
  - = 수학적 귀납법

  - 학교에서는 이런 거 안 알려줌 ㅎㅎ;;

재귀함수

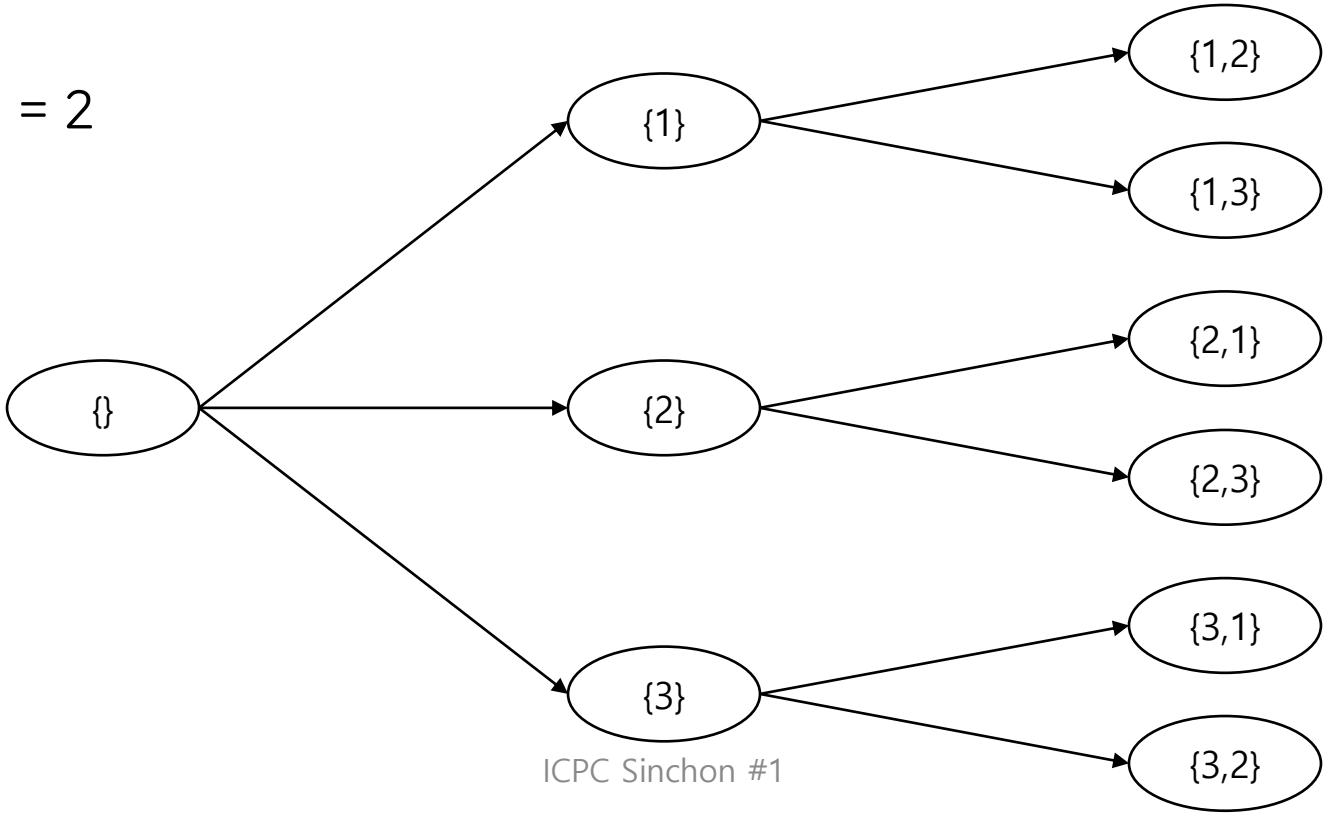
- 주의사항
  - 재귀 호출을 하지 않는 케이스(종료 조건)이 있어야 함
    - 없으면 프로그램이 종료되지 않음
  - 재귀 호출 과정에 사이클이 없어야 함
    - f(a) → f(b) → f(c) → … → f(a)
    - 프로그램이 종료되지 않음
  - 종료 조건에 가까워지는 방향으로 설계
    - 앞에서 본 예시는 l = r이 종료 조건, r – l이 작아지는 방향으로 재귀 호출

질문?

재귀함수 – 예제 3

- BOJ 15649 N과 M (1)
  - 1부터 N까지 자연수 중에서 중복 없이 M개를 고른 수열을 모두 출력
  - 수열은 사전 순으로 증가하는 순서로 출력

  - ex. N = 3, M = 2



재귀함수 – 예제 3

- f(지금까지 선택한 수의 개수, 지금까지 만든 배열)
  - f(choose, arr)
  - choose = M이면 배열 출력하고 종료
  - arr에 없는 수를 배열의 맨 뒤에 추가하고 재귀 호출
    - arr[choose] = num;
    - f(choose+1, arr)

```c
#include <stdio.h>

void f(int n, int m, int choose, int arr[]){
    if(choose == m){ // 종료 조건
        for(int i=0; i<m; i++){
            printf("%d ", arr[i]);
        }
        printf("\n");
        return;
    }

    for(int i=1; i<=n; i++){ // n개의 수를 한 번씩 시도
        int exist = 0; // 이미 만든 배열에 i가 있으면 exist = 1
        for(int j=0; j<choose; j++){
            if(arr[j] == i) exist = 1;
        }
        if(exist == 0){
            arr[choose] = i; // i 선택
            f(n, m, choose+1, arr); // 선택한 수의 개수 1 증가
            arr[choose] = 0; // i 넣은 거 원상 복구
        }
    }
}

int main(){
    int N, M, A[8];
    scanf("%d %d", &N, &M);
    f(N, M, 0, A);
}
```

질문?

재귀함수 – 예제 3

- exist 변수의 값을 매번 계산하는 것을 비효율적
  - use[num] = num을 사용했으면 1, 안 했으면 0
    - arr[choose] = num; use[num] = 1;
    - f(choose+1, arr, use);
    - arr[choose] = 0; use[num] = 0;

```c
#include <stdio.h>

void f(int n, int m, int choose, int arr[], int use[]){
    if(choose == m){ // 종료 조건
        for(int i=0; i<m; i++){
            printf("%d ", arr[i]);
        }
        printf("\n");
        return;
    }

    for(int i=1; i<=n; i++){ // n개의 수를 한 번씩 시도
        if(use[i] == 0){ // 아직 i를 사용하지 않았다면
            arr[choose] = i; // i 선택
            use[i] = 1;
            f(n, m, choose+1, arr, use); // 선택한 수의 개수 1 증가
            arr[choose] = 0; // i 넣은 거 원상 복구
            use[i] = 0;
        }
    }
}

int main(){
    int N, M, A[8], U[9] = {0}; // U는 0으로 초기화
    scanf("%d %d", &N, &M);
    f(N, M, 0, A, U);
}
```

재귀함수 – 예제 3

- 항상 5개의 인자를 모두 넘겨줘야 할까?
  - n, m은 변하지 않음
  - arr, use도 포인터를 넘겨주기 때문에 변하지 않음
  - 항상 동일한 값은 전역 변수로 빼도 됨

```
#include <stdio.h>

int N, M, A[8], U[9]; // 전역 변수는 0으로 초기화

void f(int choose){
    if(choose == M){
        for(int i=0; i<M; i++) printf("%d ", A[i]);
        printf("\n");
        return;
    }

    for(int i=1; i<=N; i++){
        if(!U[i]){
            A[choose] = i; U[i] = 1;
            f(choose+1);
            A[choose] = 0; U[i] = 0;
        }
    }
}

int main(){
    scanf("%d %d", &N, &M);
    f(0);
}
```

재귀함수 – 예제 4

- BOJ 15650 N과 M (2)
  - 1부터 N까지 자연수 중에서 중복 없이 M개를 오름차순으로 고른 수열을 모두 출력
  - 수열은 사전 순으로 증가하는 순서로 출력

  - 현재 배열에 맨 뒤에 있는 수보다 큰 수를 선택해야 함

```
int N, M, A[8];

void f(int choose){
    if(choose == M){
        for(int i=0; i<M; i++) printf("%d ", A[i]);
        printf("\n");
        return;
    }

    for(int i=1; i<=N; i++){
        if(choose == 0 || A[choose-1] < i){
            A[choose] = i;
            f(choose+1);
        }
    }
}
```

질문?

재귀함수 – 예제 5

- BOJ 11729 하노이 탑 이동 순서
  - 3개의 기둥이 있고 첫 번째 기둥에 서로 다른 크기의 원판 N개 있음
  - 어떤 원판 위에는 자신보다 작은 원판만 올라올 수 있음
  - 원판을 최소한으로 이동해서 첫 번째 기둥에 있는 모든 원판을 세 번째 기둥으로 옮겨야 함

재귀함수 – 예제 5

- 최적 전략에 대해 생각해보자
  - 첫 번째 기둥에 있는 N개의 원판을 세 번째 기둥으로 옮겨야 함
  - 가장 큰 원판부터 차례대로 세 번째 기둥으로 옮김

  - 가장 큰 원판을 어떻게 꺼내지?
  - 위에 있는 N-1개를 잠시 두 번째 기둥으로 옮기면 됨

  - N-1개를 첫 번째 기둥에서 두 번째 기둥으로 옮기고
  - N번째 원판을 첫 번째 기둥에서 세 번째 기둥으로 옮기고
  - N-1개를 두 번째 기둥에서 세 번째 기둥으로 옮기면 됨

재귀함수 – 예제 5

- f(n, a, b, c) : n개의 원판을 a에서 c로 옮기는 과정을 구하는 함수
  - n-1개를 a에서 b로 옮김 : f(n-1, a, c, b);
  - n번째 원판을 a에서 c로 옮김 : f(1, a, b, c);
  - n-1개를 b에서 c로 옮김 : f(n-1, b, a, c);

  - 종료 조건 : n = 1, a c 출력하고 종료

  - 이동 횟수
    - T(1) = 1
    - T(n) = 2*T(n-1) + 1
    - T(n) = 2^n-1

```c
#include <stdio.h>

void f(int n, int a, int b, int c){
    if(n == 1){
        printf("%d %d\n", a, c);
        return;
    }
    f(n-1, a, c, b);
    f(1, a, b, c);
    f(n-1, b, a, c);
}

int main(){
    int N;
    scanf("%d", &N);
    printf("%d\n", (1<<N) - 1);
    f(N, 1, 2, 3);
}
```

질문?

수학적 귀납법

수학적 귀납법

- 자연수에 대한 명제 P(n)이 모든 자연수에 대해 성립함을 증명
  - 어떤 정수 n_0에 대해, $n \geq n_0$을 만족하는 모든 n에 대해 P(n)이 성립

- (base case) P(n_0)이 성립함을 증명
- (inductive step) P(k)가 성립하면 P(k+1)이 성립함을 증명
  - k는 n_0보다 크거나 같은 정수

- $n \geq n_0$인 모든 정수 n에 대해 P(n)이 성립
  - P(n_0) ⇒ P(n_0+1) ⇒ P(n_0+2) ⇒ P(n_0+3) ⇒ …

수학적 귀납법

- $1 + 3 + 5 + \cdots + (2n-1) = n^2$이 성립함을 증명하자.
  - $P(n) = 1 + 3 + 5 + \cdots + (2n-1) = n^2$이 성립하는가?
  - $n_0 = 1$

- $P(1) : 1 = 1^2$이므로 자명
- $P(k) : 1 + 3 + 5 + \cdots + (2k-1) = k^2$ 성립한다고 가정하자.
  - 양변에 $2k+1$을 더하면
  - $1 + 3 + 5 + \cdots + (2k-1) + (2k+1) = k^2 + 2k + 1 = (k+1)^2$
  - $1 + 3 + 5 + \cdots + (2k-1) + (2(k+1)-1) = (k+1)^2$
  - $1 + 3 + 5 + \cdots + (2(k+1)-1) = (k+1)^2 : P(k+1)$

강한 수학적 귀납법

- (base case) $P(n_0)$이 성립
- (inductive step) $n \leq k$인 모든 n에 대해 P(n)이면 P(k+1)

- $n \geq n_0$인 모든 정수 n에 대해 P(n)이 성립
  - $P(n_0) \Rightarrow P(n_0+1)$
  - $P(n_0), P(n_0+1) \Rightarrow P(n_0+2)$
  - $P(n_0), P(n_0+1), P(n_0+2) \Rightarrow P(n_0+3)$

강한 수학적 귀납법

- 모든 트리에는 차수가 1 이하인 정점이 있음을 증명
  - P(n) := 정점이 n개인 트리에서 성립하는가?
  - $n_0 = 1$
- P(1) : 정점이 1개인 트리는 차수가 0인 정점이 있음
- 정점이 1..k인 트리에 모두 차수가 1 이하인 정점이 있다고 하자.
  - 정점이 k+1인 트리는 정점이 k개 이하인 1개 이상의 트리가
  - 새로 추가된 정점과 연결되어 있는 형태
  - 정점이 k개 이하인 트리에 차수가 1 이하인 정점이 있으므로
  - 정점이 k+1개인 트리에도 차수가 1 이하인 정점이 있음 : P(k+1)

질문?

분할 정복

분할 정복

- 분할 정복
  - 큰 문제를 여러 개의 작은 문제로 쪼갠 다음 (분할)
  - 작은 문제들의 답을 이용해 큰 문제의 답을 구함 (정복)

  - 동적 계획법?
    - 뒤에 있는 내용을 보면서 어떤 점이 다른 지 직접 확인해 보자.

분할 정복

- 분할 정복
  - 문제의 크기가 충분히 작은 경우 직접 해결
  - 문제의 크기가 큰 경우 K≥1개의 부분 문제로 쪼개서 각각 해결

```
void DivideAndConquer(InputType in, OutputType out){
    // 문제의 크기가 충분히 작은 경우 직접 해결
    if(in.size() <= Small){
        DirectSolve(in, out);
        return;
    }

    // 문제를 K개의 부분 문제로 분할함
    InputType in_small[K] = Divide(in, K);
    OutputType out_small[K];
    for(int i=0; i<K; i++){
        DivideAndConquer(in_small[i], out_small[i]);
    }
    out = Combine(out_small[0], out_small[1], ... , out_small[k-1]);
}
```

분할 정복

- 분할 정복의 시간 복잡도
  - $T(N) = D(N) + \text{sum } T(i) + C(N)$ if $N > Small$
    $S(N)$ if $N \leq Small$
    - N : 입력의 크기
    - T(N) : 입력의 크기가 N인 문제를 풀 때 걸리는 시간
    - D(N) : 크기가 N인 문제를 부분 문제로 분할할 때 걸리는 시간
    - C(N) : 크기가 N인 문제에서 부분 문제의 답을 합칠 때 걸리는 시간
    - S(N) : 크기가 N인 문제를 직접 해결할 때 걸리는 시간
  - 부분 문제들의 크기를 균등하게 분할하면
    - $T(N) = D(N) + aT(N/b) + C(N)$

질문?

분할 정복의 예시

분할 정복의 예시 – 1

- 정수 거듭제곱
  - 음이 아닌 정수 a, b, c에 대해 $a^b \pmod c$를 구하는 문제

  - b = 0이면 직접 해결 ($a^b=1$)
  - b ≥ 1이면 더 작은 문제로 나눠서 해결
    - b가 짝수면 $a^{b/2} * a^{b/2}$
    - b가 홀수면 $a^{(b-1)/2} * a^{(b-1)/2} * a$

  - 중복된 호출을 제외하면, 크기가 b인 문제를 크기가 b/2인 부분 문제 하나로 쪼갬
  - T(N) = O(1) + T(N/2) + O(1) = T(N/2) + O(1)
  - T(N) = O(log N)

분할 정복의 예시 – 2

- 합병 정렬
  - 배열의 구간 [l, r]을 정렬하는 알고리즘

  - l = r이면 직접 해결 (원소가 하나인 배열은 이미 정렬되어 있음)
  - l < r이면 더 작은 문제로 나눠서 해결
    - m = floor((l+r)/2)라고 할 때
    - [l, m]과 [m+1, r]을 각각 정렬한 다음 (분할)
    - 정렬된 두 배열을 합침 (정복)

  - 크기가 r – l인 문제를 크기가 약 (r-l)/2인 부분 문제 2개로 쪼갬
    - T(N) = O(1) + 2T(N/2) + O(N) = 2T(N/2) + O(N)
    - T(N) = O(N log N)

분할 정복의 예시 – 2

- 합병 정렬

```cpp
#include <bits/stdc++.h>
using namespace std;

int N, A[1010101];

void Merge(int s, int m, int e){
    static int tmp[1010101];

    int i = s, j = m + 1, idx = s;
    while(i <= m && j <= e){
        if(A[i] < A[j]) tmp[idx++] = A[i++];
        else tmp[idx++] = A[j++];
    }
    while(i <= m) tmp[idx++] = A[i++];
    while(j <= e) tmp[idx++] = A[j++];
    for(int k=s; k<=e; k++) A[k] = tmp[k];
}

void MergeSort(int s, int e){
    if(s == e) return;
    int m = (s + e) / 2;
    MergeSort(s, m);
    MergeSort(m+1, e);
    Merge(s, m, e);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i];
    MergeSort(1, N);
    for(int i=1; i<=N; i++) cout << A[i] << "\n";
}
```

질문?

분할 정복의 예시 – 3

- Inversion Counting
  - i < j & A[i] > A[j]를 만족하는 순서쌍 (i, j)의 개수를 세는 문제 (BOJ 1517 버블 소트)

  - 각각의 A[i]에 대해, 자신보다 뒤에 있으면서 자신보다 작은 A[j]의 개수를 세야 함
    - 단순하게 세면 $O(N^2)$
    - 원소들의 실제 위치가 아닌 전후 관계만 중요하다는 것에 주목하자.

분할 정복의 예시 – 3

- Inversion Counting
  - 배열을 반으로 잘라보자.
    - [l, m]과 [m+1, r] 안에서 발생하는 inversion은 재귀적으로 계산하고 (분할)
    - [l, m]과 [m+1, r] 간의 inversion의 개수만 세면 된다. (정복)
    - [l, m]의 원소 x보다 작은 [m+1, r]의 원소 y의 개수를 세면 된다.
      - 합병 정렬과 똑같은 방법으로 할 수 있음
  - T(N) = O(1) + 2T(N/2) + O(N) = 2T(N/2) + O(N)
  - T(N) = O(N log N)

```
long long R = 0;

void Merge(int s, int m, int e){
    static int tmp[505050];

    int i = s, j = m + 1, idx = s, cnt = 0;
    while(i <= m && j <= e){
        if(A[i] <= A[j]) tmp[idx++] = A[i++], R += cnt;
        else tmp[idx++] = A[j++], cnt += 1;
    }
    while(i <= m) tmp[idx++] = A[i++], R += cnt;
    while(j <= e) tmp[idx++] = A[j++];
    for(int k=s; k<=e; k++) A[k] = tmp[k];
}
```

질문?

분할 정복의 예시 – 4

- BOJ 1725 히스토그램
  - 히스토그램에서 가장 큰 직사각형을 구하는 문제
    - 배열에서 (j-i+1) * min(A[i], A[i+1], … , A[j])의 최댓값을 구하는 문제
  - 단순하게 계산하면 $O(N^3)$
  - 조금 더 똑똑하게 계산하면 $O(N^2)$
  - 배열을 단순히 반으로 나누는 것은 조금 애매함…

분할 정복의 예시 – 4

- BOJ 1725 히스토그램
  - 배열을 반으로 잘라보자.
    - A[m]을 포함하는 모든 구간을 처리하면
    - 고려하지 않은 구간은 [l, m-1] 또는 [m+1, r]에 완전히 포함됨

    - [l, m-1]과 [m+1, r]에 포함된 구간은 재귀적으로 잘 처리하고
    - [l, r]에서 A[m]을 포함하는 모든 구간을 확인하는 방법만 찾으면 됨

분할 정복의 예시 – 4

- BOJ 1725 히스토그램
  - A[m]을 포함하는 모든 구간을 확인
    - 단순하게 확인하면 $O(N^2)$ / 더 빠르게 해야 함
    - [m, m]부터 시작해서 한 칸 씩 확장하는 방식으로 진행
    - 왼쪽으로 확장해야 할까? 오른쪽으로 확장해야 할까?
    - 확장 방향에 관계없이 (j-i+1)의 값은 동일하게 1 증가함
    - min(A[i], A[i+1], … , A[j])가 덜 작아지는 방향으로 확장해야 함
    - 확장은 r-l번 하므로 선형 시간에 확인할 수 있음
  - T(N) = 2T(N/2) + O(N)
  - T(N) = O(N log N)

```cpp
using ll = long long;

ll N, A[1010101];

ll Solve(int s, int e){
    if(s > e) return 0;
    if(s == e) return A[s];
    int m = (s + e) / 2;

    ll res = A[m], cnt = 1, mn = A[m];
    int i = m - 1, j = m + 1;

    while(s <= i && j <= e){
        if(A[i] > A[j]) cnt++, mn = min(mn, A[i--]);
        else cnt++, mn = min(mn, A[j++]);
        res = max(res, cnt * mn);
    }
    while(s <= i){
        cnt++; mn = min(mn, A[i--]);
        res = max(res, cnt * mn);
    }
    while(j <= e){
        cnt++; mn = min(mn, A[j++]);
        res = max(res, cnt * mn);
    }
    return max({ res, Solve(s, m-1), Solve(m+1, e) });
}
```

질문?

과제

- 필수
  - 15649 N과 M (1)
  - 15650 N과 M (2)
  - 11729 하노이 탑 이동 순서
  - 1629 곱셈
  - 2751 수 정렬하기 2 (합병 정렬 구현)
  - 2630 색종이 만들기
  - 1517 버블 소트
  - 1725 히스토그램
  - 21870 시철이가 사랑한 GCD
- 심화
  - 17613 점프
  - 2261 가장 가까운 두 점
  - 16885 벡터의 합
  - 18253 최단경로와 쿼리