알고리즘 및 실습 과제 1

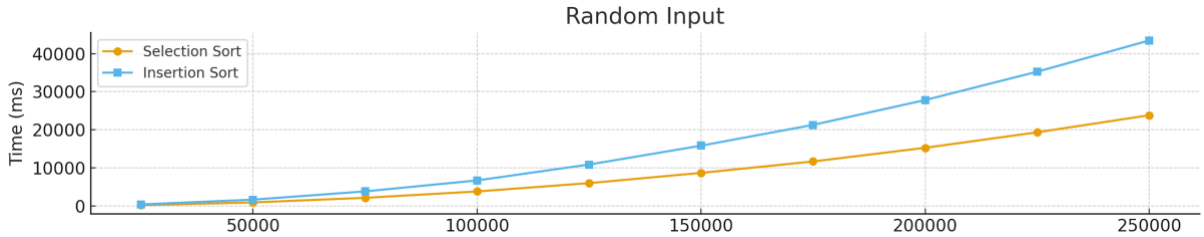
컴퓨터공학과 21011946 이선욱

실험 진행 방식

- **N 은 25000 으로 시작해 250000 까지 실험한다.**
  (N 이 10 만으로 시작한 경우 실험시간이 너무 오래 걸려 2.5 만으로 결정)
- 각 입력에 대해 두가지 정렬 알고리즘의 시간을 비교한다.

실험 1. 랜덤 입력

	2.5 만	5 만	7.5 만	10 만	12.5 만	15 만	17.5 만	20 만	22.5 만	25 만
선택 정렬	249.7ms	965.9ms	2175.2ms	3842.7ms	6031.9ms	8714.9ms	11708.7ms	15293.7ms	19354.4ms	23826.8ms
삽입 정렬	443.5ms	1694.6ms	3862.9ms	6736.3ms	10904.3ms	15849.9ms	21302.2ms	27811.6ms	35241.6ms	43431.5ms



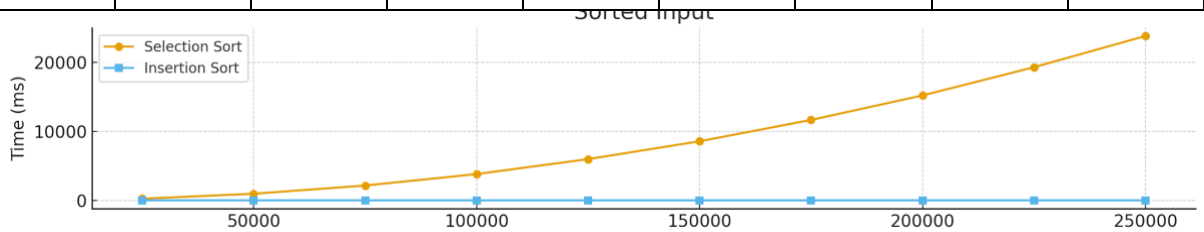
선택 정렬: $O(n^2)$ 복잡도. 실행 시간이 대체로 n^2에 비례하여 증가.

삽입 정렬: $O(n^2)$ → 선택 정렬보다 더 느림.

실험 결과 삽입 정렬이 항상 선택 정렬보다 1.5~2배 정도 더 오래 걸림.
(삽입 정렬이 선택 정렬보다 교환 연산이 많이 발생하기 때문)

실험2. 정렬된 입력

	2.5 만	5 만	7.5 만	10 만	12.5 만	15 만	17.5 만	20 만	22.5 만	25 만
선택 정렬	236.881	952.0	2141.5	3797.3	5965.0	8542.2	11632.0	15175.6	19262.6	23776.4
삽입 정렬	0.034	0.086	0.098	0.131	0.165	0.195	0.230	0.266	0.303	0.330

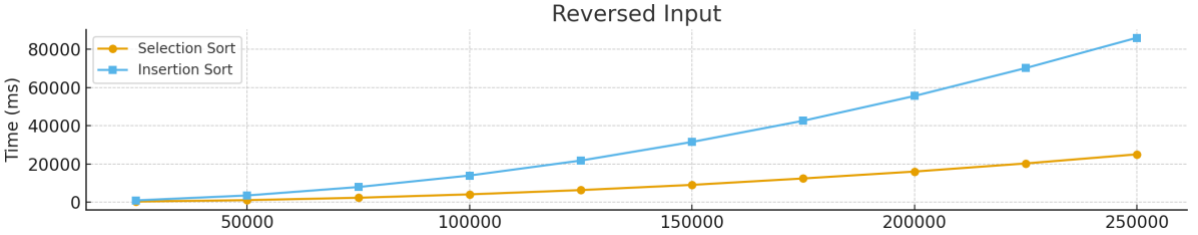


- **삽입 정렬**: 최선의 경우 O(n), 이미 정렬된 상태에서는 교환이 거의 없음. → ms 단위가 아닌 μs 수준으로 매우 빠름.
- **선택 정렬**: 데이터 상태와 무관하게 항상 $O(n^2)$ 수행. → 시간이 계속 크게 나옴.

→ 따라서 정렬된 경우 삽입 정렬이 압도적으로 빠름.

실험 3. 역순으로 정렬된 입력

	2.5 만	5 만	7.5 만	10 만	12.5 만	15 만	17.5 만	20 만	22.5 만	25 만
선택 정렬	249.0	999.1	2253.3	4031.3	6269.6	8998.1	12359.9	15979.1	20231.3	25007.3
삽입 정렬	873.6	3447.7	7869.6	13902.4	21744.0	31489.9	42594.6	55566.8	70176.7	86070.6



- **삽입 정렬**: 최악의 경우 $O(n^2)$에서도 **모든 비교와 교환 발생** → 실행 시간이 증가.
- **선택 정렬**: 데이터 상태와 무관하게 $O(n^2)$ 동일.

→ 따라서 역순 입력에서는 삽입 정렬이 **선택 정렬보다 훨씬 느림**.

결론

- 선택 정렬은 **입력 상태와 무관하게 항상** O(n^2) 시간을 소요.
- 삽입 정렬은 **정렬된 데이터에서는 매우 효율적 (**O(n)**)**, 그러나 랜덤·역순 데이터에서는 오히려 더 느려짐.
- 따라서 삽입 정렬은 **데이터가 정렬되어 있거나 거의 정렬된 경우**에만 효율적이며, 일반적인 대규모 데이터 정렬에는 적합하지 않음.

실험코드

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h> //mac이라 clock_gettime함수 이용해 시간 측정
#include <string.h>

void swap(int *arr, int a, int b) {// 교환 함수
    int temp = arr[a];
    arr[a] = arr[b];
    arr[b] = temp;
}

void selection_sort(int *arr, int n) { // 선택 정렬
    int min_index;
    for (int i = 0; i < n - 1; i++) { // 앞부터 탐색 시작
        min_index = i;
        for (int j = i + 1; j < n; j++) { // 해당 칸의 뒤쪽 원소들 중 가장 작은 값을 찾음
            if (arr[j] < arr[min_index]) {
                min_index = j;
            }
        }
        swap(arr, i, min_index); // 찾은 최소값과 현재 인덱스 위치의 값을 교환
    }
}

void insertion_sort(int *arr, int n) {
    // 삽입 정렬
    for (int i = 1; i < n; i++) {
        for (int j = i - 1; j >= 0 && arr[j + 1] < arr[j]; j--) {// 바로 뒤의 값을 앞 원소들과 비교해 순서상 들어갈 위치에 넣는다.
            swap(arr, j + 1, j); // 이를 위해서 뒤쪽 값들을 한칸씩 뒤로 이동시키고 빈 자리에 삽입
        }
    }
}
// 시간 측정 함수
double measure_time(void (*sort_func)(int*, int), int *arr, int n) {// mac환경에서 진행하기에 clock_gettime함수 이용
    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC, &start);//시작 시간
    sort_func(arr, n);
    clock_gettime(CLOCK_MONOTONIC, &end);//종료시간
```

```c
    return (end.tv_sec - start.tv_sec) * 1000.0 + //시작과 종료시간의 차를 통해
ms 단위로 함수 실행 속도 계산
           (end.tv_nsec - start.tv_nsec) / 1e6;
}

// 배열 생성 함수
void make_random_array(int *arr, int n) {
    for (int i = 0; i < n; i++) {
        arr[i] = rand();
    }
}

void reverse_array(int *arr, int n) { // 배열 역순으로 뒤집는 함수
    for (int i = 0; i < n / 2; i++) {
        swap(arr, i, n - 1 - i);
    }
}

int main() {
    srand(time(NULL));// 랜덤 값을 위해 srand 설정
    int n;
    scanf("%d", &n);

    int *base = malloc(n * sizeof(int));
    int *arr_selection = malloc(n * sizeof(int));
    int *arr_insertion = malloc(n * sizeof(int));
    if (!base || !arr_selection || !arr_insertion) return 1;

    // A: 랜덤 input
    make_random_array(base, n);
    memcpy(arr_selection, base, n * sizeof(int));//배열 동일하게 초기화
    memcpy(arr_insertion, base, n * sizeof(int));// 배열 동일하게 초기화
    printf("[랜덤 입력]\n");
    printf("선택 정렬: %f ms\n", measure_time(selection_sort, arr_selection,
n));
    printf("삽입 정렬: %f ms\n\n", measure_time(insertion_sort,
arr_insertion, n));

    // B: 정렬 input
    memcpy(base, arr_selection, n * sizeof(int)); // 이미 정렬된 상태 활용
    memcpy(arr_selection, base, n * sizeof(int));
    memcpy(arr_insertion, base, n * sizeof(int));
    printf("[정렬된 입력]\n");
    printf("선택 정렬: %f ms\n", measure_time(selection_sort, arr_selection,
n));
    printf("삽입 정렬: %f ms\n\n", measure_time(insertion_sort,
arr_insertion, n));

    // ------------ 실험 3: 역순 입력 ------------
    reverse_array(base, n);
    memcpy(arr_selection, base, n * sizeof(int));
    memcpy(arr_insertion, base, n * sizeof(int));
    printf("[역순 입력]\n");
    printf("선택 정렬: %f ms\n", measure_time(selection_sort, arr_selection,
n));
    printf("삽입 정렬: %f ms\n\n", measure_time(insertion_sort,
arr_insertion, n));
```

```
    free(base);
    free(arr_selection);
    free(arr_insertion);
    return 0;
}
```