1. 문제 해결 및 논의

(실험1) 구현된 정렬 알고리즘 테스트

구현한 정렬 알고리즘이 제대로 동작하는지 보기 위해 테스트를 진행했습니다.

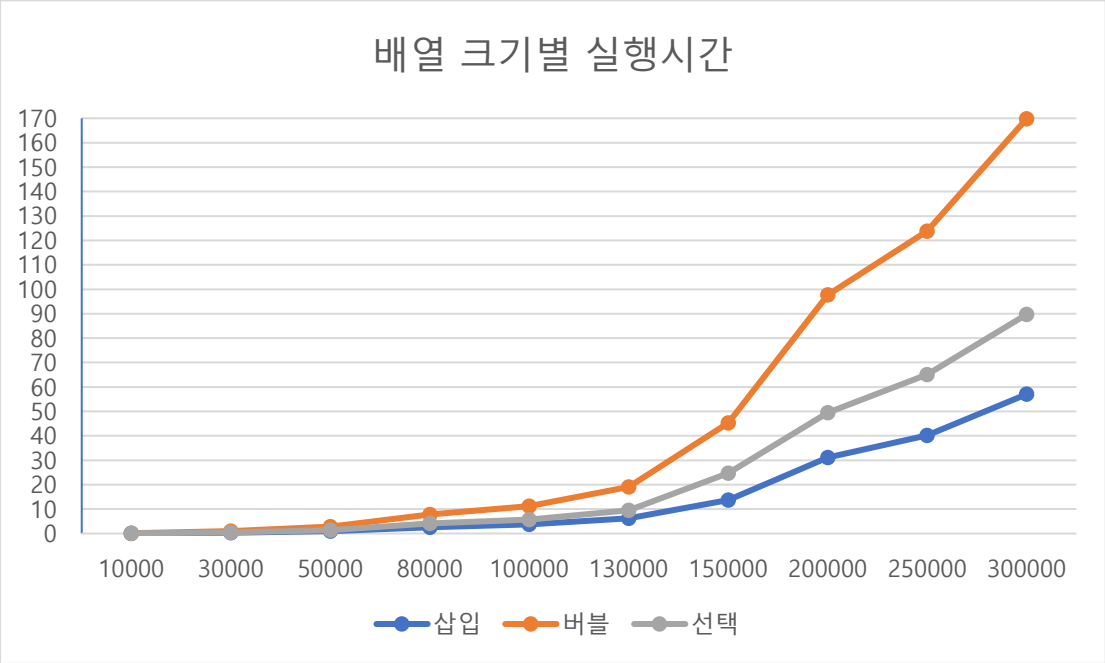
```
생성한 배열
11 17 4 10 29 4 18 18 22 14

삽입 정렬
4 4 10 11 14 17 18 18 22 29
버블 정렬
4 4 10 11 14 17 18 18 22 29
선택 정렬
4 4 10 11 14 17 18 18 22 29
):₩단국대학교₩2021-2₩자료구조₩과제2₩Debug₩과제2.exe(프로세스 18536개)이(가) 종료도
이 창을 닫으려면 아무 키나 누르세요...
```

동작여부를 보는 것이기 때문에 데이터의 범위는 0~29로 했습니다.

(실험2)

실행시간의 변화를 확실히 관찰하고, 실제 현실에서는 큰 데이터를 다룬다는 점을 고려하여 배열의 크기와 데이터의 크기를 10000~100000 사이의 값으로 정하여 10개의 원소를 갖는 다양한 크기의 배열을 각 5번씩 실행하여 평균 실행시간을 구했습니다.



실험 결과, 배열의 크기가 커질수록 실행시간도 증가했습니다. 배열의 크기가 30만개가 넘어가니

프로그램 실행 시간이 너무 오래 걸려 다음과 같이 크기를 설정하여 실험하였고, 세가지 정렬 알고리즘 중 실행시간의 변화가 가장 큰 것은 버블정렬 이였습니다. 버블 정렬의 경우 교환이 없더라도 계속해서 두 값을 비교하는 것은 반복하기 때문에 실행시간이 길고 비효율적이라고 생각합니다.

2. 작성한 코드

Sorting.h

```
int* array(int n);
void freeArray(int* arr);
void mrand(int* arr, int n);
void Insertion(int* arr, int n);
void Bubble(int* arr, int n);
void Selection(int* arr, int n);
void printArray(int* arr, int n);
```

Sorting.c

```
/*sorting algorithm /kyungmin Kim/2021.10.09*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "Sorting.h"

int* array(int n) {  //배열 동적생성
	int* arr;

	arr = (int*)malloc(sizeof(int) * n);
	mrand(arr,n);

	return arr;
}

void freeArray(int* arr) {
	free(arr);
}

void printArray(int* arr, int n) {
	int r;

	for (r = 0; r < n; r++){
		printf("%d ", arr[r]);
	}

}

void mrand(int * arr,int n) {
	srand((unsigned)time(NULL)); //실행마다 다른 난수 생성
```

```
	for (int i = 0; i < n; i++) {
		arr[i] = ((double)rand() / 32767) * (1000000 - 10000) + 10000 +
0.5;//10000~1000000
	}

}

//삽입 정렬
void Insertion(int* arr, int n) {

	for (int i = 1; i < n; i++) {
		int key = arr[i];
		int j;
		for (j = i - 1; j >= 0 && arr[j] > key; j--) {
			arr[j+1] = arr[j];
		}
		arr[j+1] = key;
	}
	//printf("₩n삽입 정렬₩n");
	//printArray(arr, n);

}

//버블 정렬
void Bubble(int* arr, int n){
	int temp;

	for (int i = 0; i < n - 1; i++) {
		for (int j = 1; j < n; j++) {
			if (arr[j-1] > arr[j]) {
				temp = arr[j-1];
				arr[j-1] = arr[j];
				arr[j] = temp;
			}
		}
	}
	//printf("₩n버블 정렬₩n");
	//printArray(arr, n);

}

//선택 정렬
void Selection(int* arr, int n){
	int temp;

	for (int i = 0; i < n-1;i++){
		int least = i;
		for (int j = i+1; j < n; j++) {
			if ( arr[j]< arr[least]) {
				least = j;
			}
		}
		temp = arr[i];
		arr[i] = arr[least];
		arr[least] = temp;
```

```
        }
        //printf("₩n선택 정렬₩n");
        //printArray(arr, n);
}
```

main.c

```
/*main code for sorting/kyungmin Kim/2021.10.09*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "Sorting.h"

int main() {

        double start, end;
        double t1=0, t2=0, t3=0;
        int* a;
        int arr_size = 130000;

        for (int i = 0; i < 5; i++) {
                a = array(arr_size);

                start = clock();
                Insertion(a, arr_size);
                end = clock();
                t1 += (double)((end - start) / CLOCKS_PER_SEC);


                start = clock();
                Bubble(a, arr_size);
                end = clock();
                t2 += (double)((end - start) / CLOCKS_PER_SEC);


                start = clock();
                Selection(a, arr_size);
                end = clock();
                t3 += (double)((end - start) / CLOCKS_PER_SEC);

                freeArray(a);
        }
        printf("₩n%f %f %f", t1 / 5, t2 / 5, t3 / 5);


}
```

Makefile

```
sort_func: Sorting.o main_sort.o
	gcc -o sort_func Sorting.o main_sort.o

Sorting.o: Sorting.o Sorting.h
	gcc -c Sorting.c -o Sorting.o

main_sort.o: main_sort.c
	gcc -c main_sort.c -o main_sort.o

clean:
	rm -f Sorting *.o
```