1. 개요

이 실험에서는 다양한 정렬 알고리즘 중 교환, 삽입, 합병, 퀵 정렬 등 총 4가지의 정렬을 사용하여 정렬시간을 비교한다. 정렬에 필요한 N의 각 배열의 원소들은 -1부터 1까지의 임의의 실수들이며, 중복되지 않는다. 또한 이러한 N은 [10, 100, 200, 300, 400, 500, 750, 1000, 2000, 3000, 4000, 5000]으로 실험을 하였으며, 정렬 결과는 비오름차순으로 정렬된다. 첫 배열 초기화 시 initArray()메소드를 통해 크기가 5000인 배열을 생성 후 각 배열 인덱스마다 -1~1인 실수 원소들을 할당한다. 원본 배열인 original_array는 수정되지 않는다. 각 정렬 알고리즘마다 이 배열을 copyArray()메소드를 통해 부분적으로 배열을 메모리에 크기가 n만큼 동적할당 한 후, original_array의 필요 인덱스만큼 복사하여 copy_array라는 배열로 내용을 저장한 뒤, 정렬 후에 콘솔창에 출력한다.

각 정렬 알고리즘은 필요한 메소드를 갖는다. 교환(Exchange()), 삽입(Insert()), 합병(Combine(), Divide()), 퀵(Partition(), Conquer()) 은 각 정렬 알고리즘마다 핵심 동작을 모듈화되어있다. 이 외에도 Swap()과 printArray()같이 공통적으로 사용되는 메소드는 따로 분리하여 작성하였다.

또한 시간 측정을 위해 구조체로 임의로 Time이라는 구조체를 작성하였다. 이 구조체는 Windows 운영체제의 API중 QueryPerformanceFrequency()와 QueryPerformanceCounter()라는 두개의 시간측정 함수를 사용하여 시간을 측정하며, 출력 범위는 소숫점 넷째자리까지 출력한다. 단위는 ms(밀리세컨드)이다.

2. Source Code

```
// Header Declaration
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <Windows.h>

// Constant variable Declaration
#define SIZE 5000     // Max n size

// Method declaration: Time class
void destroyTime(struct Time* time_ptr);
void setStart(struct Time* this);
void setFinish(struct Time* this);
double getTime(struct Time* this);
void toString(struct Time* this);

// Global variable declaration
double* original_array = { 0 };
double* copy_array = { 0 };

// Time object abstract
typedef struct Time {
        struct Time* this;
        LARGE_INTEGER timefreq, start, end;
        double time;
        void (*setStart)(struct Time* this);
        void (*setFinish)(struct Time* this);
        double (*getTime)(struct Time* this);
        void (*toString)(struct Time* this);
}Time;
```

```
// Time object constructor
Time* newTime() {
	Time* temp = (Time*)malloc(sizeof(Time));
	temp->this = temp;
	temp->setStart = setStart;
	temp->setFinish = setFinish;
	temp->getTime = getTime;
	temp->toString = toString;
	return temp;
}

// Time object destructor
void destroyTime(struct Time* time_ptr) { free(time_ptr); }

// Time object method: Start time setter
void setStart(struct Time* this) {
	QueryPerformanceFrequency(&this->timefreq);
	QueryPerformanceCounter(&this->start);
}

// Time object method: Finish time setter
void setFinish(struct Time* this) {
	QueryPerformanceCounter(&this->end);
	// 1s = 1000ms
	this->time = (double)((this->end.QuadPart - this->start.QuadPart) * 1000) /
this->timefreq.QuadPart;
}

// Time object method: Time getter
double getTime(struct Time* this) { return this->time; }

// Time object method: Print time
void toString(struct Time* this) { printf("Spend Time = %.4lfms\n", this->time); }

// Method: Print array
void printArray(double* array, int n) {
	for (int i = 0; i < n; i++) {
		printf("[%d] = %.3lf\n", i, array[i]);
	}
	printf("\n");
}

// Method: Initialize Array
double* initArray() {
	double *temp = calloc(SIZE, sizeof(double));
	for (int i = 0; i < SIZE; i++) {
		temp[i] = (rand() / (double)RAND_MAX * 2.0f) - 1;
		for (int j = 0; j < i; j++) {
			if (i != 0 && temp[i] == temp[j]) {
				i--;
				break;
			}
		}
	}
	return temp;
}

// Method: Copy Array
```

```
double *copyArray(double* array, int n) {
	double *temp = calloc(n, sizeof(double));
	for (int i = 0; i < n; i++) {
		temp[i] = array[i];
	}
	return temp;
}

// Method: Swap array element
void Swap(double* array, int end_index, int obj_index) {
	double temp;
	temp = array[obj_index];
	array[obj_index] = array[end_index];
	array[end_index] = temp;
}

// Mergesort method: Combine
void Combine(double* array, int left, int middle, int right, int n) {
	double* temp = copyArray(array, n);
	int i = left;
	int j = middle + 1;
	int position = left;

	/* 분할 정렬된 list의 합병 */
	while (i <= middle && j <= right) {
		if (array[i] <= array[j]) {
			temp[position++] = array[i++];
		}
		else {
			temp[position++] = array[j++];
		}
	}

	// 남아 있는 값들을 일괄 복사
	if (i > middle) {
		for (int l = j; l <= right; l++)
			temp[position++] = array[l];
	}
	// 남아 있는 값들을 일괄 복사
	else {
		for (int l = i; l <= middle; l++)
			temp[position++] = array[l];
	}

	for (int i = 0; i < n; i++) {
		array[i] = temp[i];
	}
}

// Mergesort method: Divide
void Divide(double* array, int left, int right, int n) {
	int middle = (left + right) / 2;
	if (left < right) {
		Divide(array, left, middle, n);
		Divide(array, middle + 1, right, n);
		Combine(array, left, middle, right, n);
	}
```

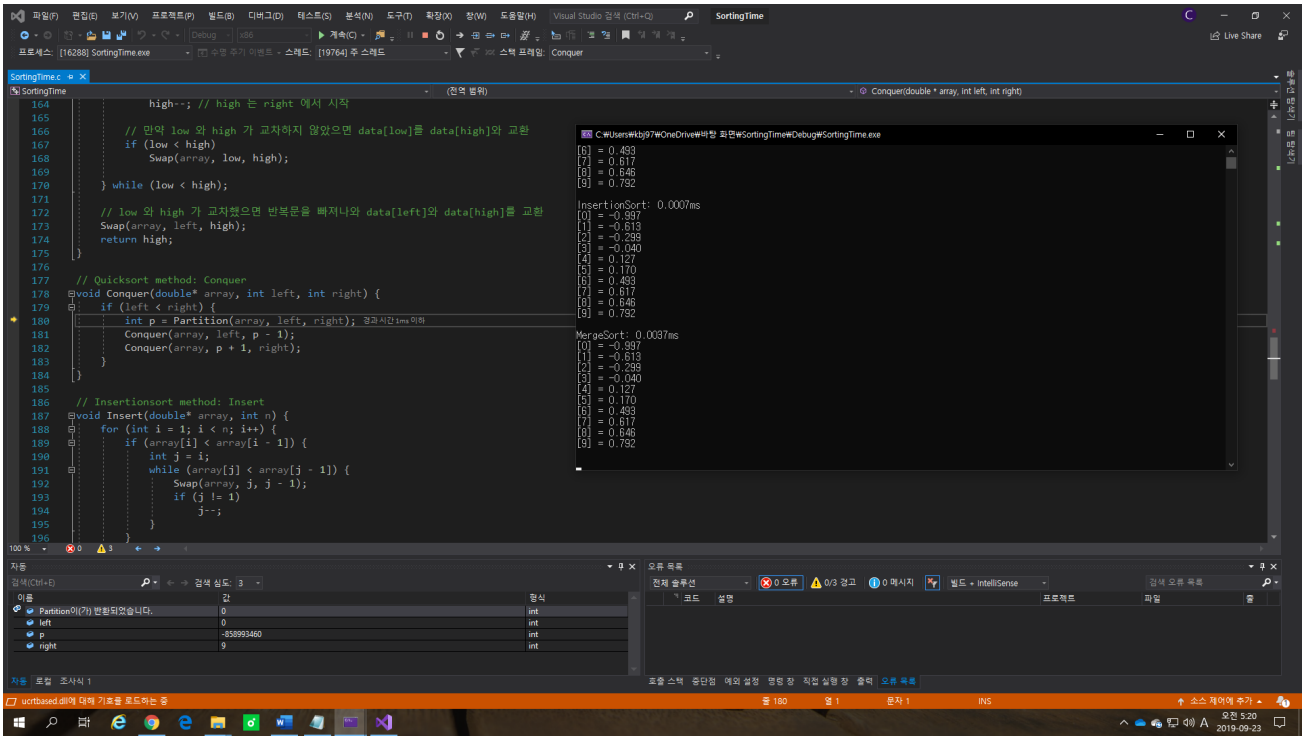
```
}

// Quicksort method: Partition
int Partition(double* array, int left, int right) {
	int low = left;
	int high = right;
	double pivot = array[left];

	// low 와 high 가 교차할 때까지 반복(low < high)
	do {
		do { // data[low] 가 pivot 보다 작으면 계속 low 를 증가
			low++; // low = left + 1 에서 시작
		} while (low <= right && array[low] < pivot);

		// data[high] 가 pivot 보다 크면 계속 high 를 감소
		while (high >= left && array[high] > pivot)
			high--; // high 는 right 에서 시작

		// 만약 low 와 high 가 교차하지 않았으면 data[low]를 data[high]와 교환
		if (low < high)
			Swap(array, low, high);

	} while (low < high);

	// low 와 high 가 교차했으면 반복문을 빠져나와 data[left]와 data[high]를 교환
	Swap(array, left, high);
	return high;
}

// Quicksort method: Conquer
void Conquer(double* array, int left, int right) {
	if (left < right) {
		int p = Partition(array, left, right);
		Conquer(array, left, p - 1);
		Conquer(array, p + 1, right);
	}
}

// Insertionsort method: Insert
void Insert(double* array, int n) {
	for (int i = 1; i < n; i++) {
		if (array[i] < array[i - 1]) {
			int j = i;
			while (array[j] < array[j - 1]) {
				Swap(array, j, j - 1);
				if (j != 1)
					j--;
			}
		}
	}
}

// Exchangesort method: Exchange
void Exchange(double* array, int n) {
	for (int i = 0; i < n; i++) {
		for (int j = i + 1; j < n; j++) {
```

```
                        if (array[i] > array[j])
                                Swap(array, i, j);
                }
        }
}

// Method: Exchangesort
void ExchangeSort(double* array, int n) {
        Time* spendTime = newTime();
        copy_array = copyArray(array, n);
        spendTime->setStart(spendTime);
        Exchange(copy_array, n);
        spendTime->setFinish(spendTime);
        printf("ExchangeSort: %.4lfms\n", spendTime->getTime(spendTime));
        destroyTime(spendTime);
        printArray(copy_array, n);
        free(copy_array);
}

// Method: Insertionsort
void InsertionSort(double* array, int n) {
        Time* spendTime = newTime();
        copy_array = copyArray(array, n);
        spendTime->setStart(spendTime);
        Insert(copy_array, n);
        spendTime->setFinish(spendTime);
        printf("InsertionSort: %.4lfms\n", spendTime->getTime(spendTime));
        destroyTime(spendTime);
        printArray(copy_array, n);
        free(copy_array);
}

// Method: Mergesort
void MergeSort(double* array, int n) {
        Time* spendTime = newTime();
        copy_array = copyArray(array, n);
        spendTime->setStart(spendTime);
        Divide(copy_array, 0, n - 1, n);
        spendTime->setFinish(spendTime);
        printf("MergeSort: %.4lfms\n", spendTime->getTime(spendTime));
        destroyTime(spendTime);
        printArray(copy_array, n);
        free(copy_array);
}

// Method: Quicksort
void QuickSort(double* array, int n) {
        Time* spendTime = newTime();
        copy_array = copyArray(array, n);
        spendTime->setStart(spendTime);
        Conquer(copy_array, 0, n - 1);
        spendTime->setFinish(spendTime);
        printf("QuickSort: %.4lfms\n", spendTime->getTime(spendTime));
        destroyTime(spendTime);
        printArray(copy_array, n);
        free(copy_array);
}

// Method: Main
```

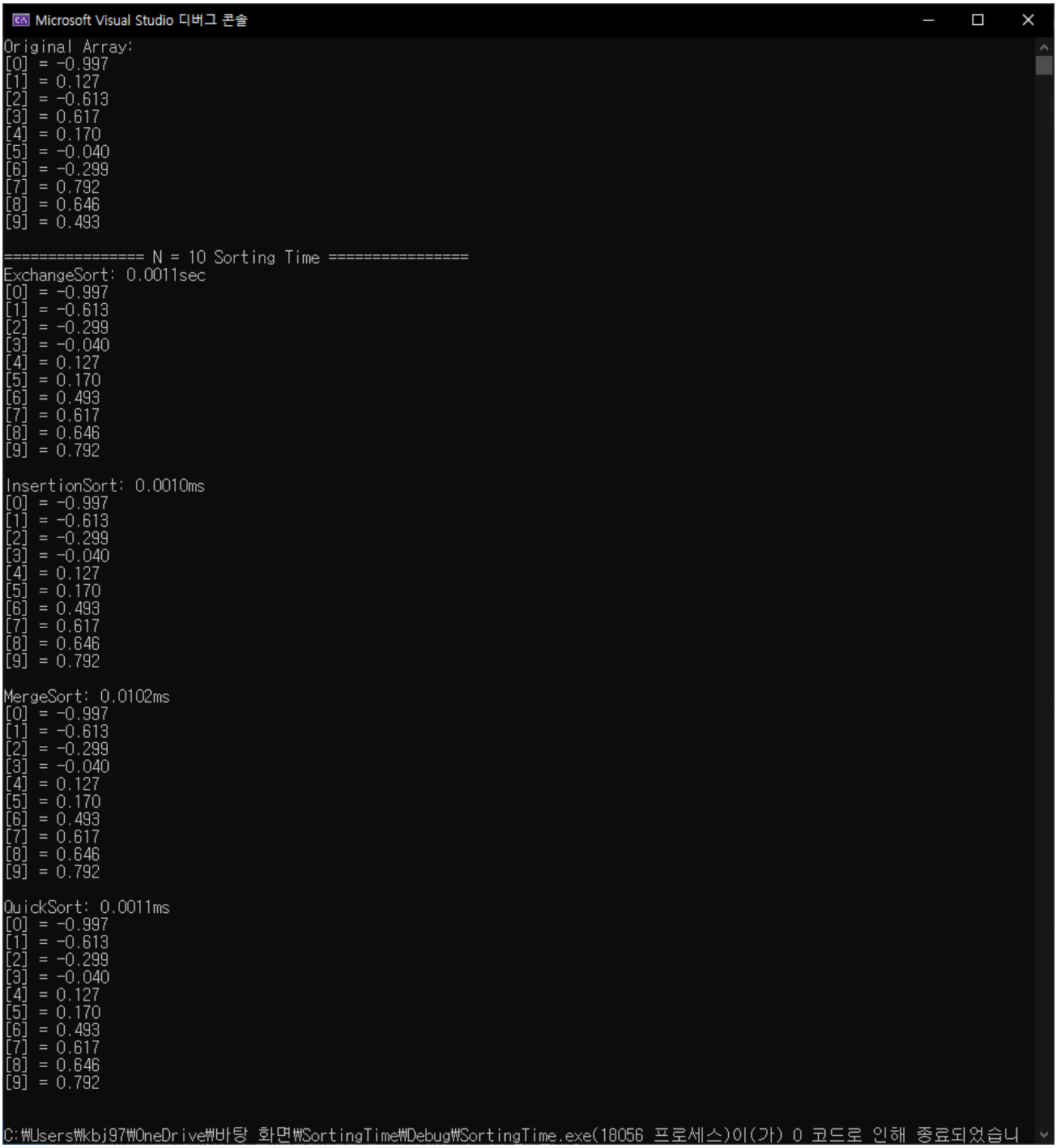
```
int main(void) {
	double *copy_array = { 0 };
	int n[12] = { 10, 100, 200, 300, 400, 500, 750, 1000, 2000, 3000, 4000, 5000 };
	original_array = initArray();
	printf("Original Array: \n");
	printArray(original_array, SIZE);
	for (int i = 0; i < sizeof(n)/sizeof(int); i++) {
		printf("================ N = %d Sorting Time ================\n", n[i]);
		ExchangeSort(original_array, n[i]);
		InsertionSort(original_array, n[i]);
		MergeSort(original_array, n[i]);
		QuickSort(original_array, n[i]);
	}
	return 0;
}
```

시각적 편의를 위해 스크린샷에서는 배열이 정렬된 모습과 정렬까지 걸린 시간으로 두장으로 나누어 캡처하였다. 그러나 코드상에서는 한번에 출력되도록 작성 되어 있음

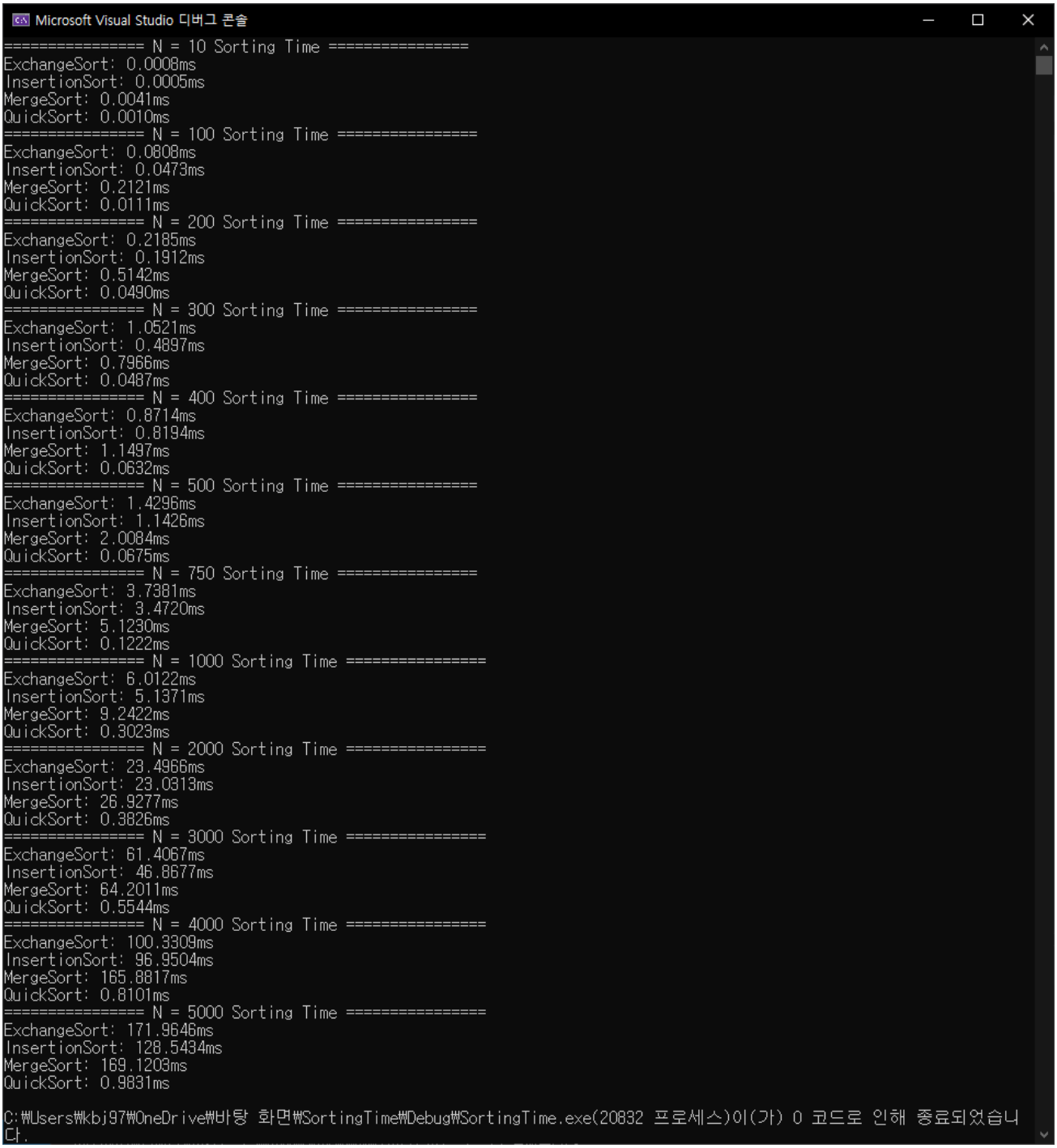
3. 실행 화면


SortingTime
SortingTime.c
Conquer(double * array, int left, int right)
high--; // high 는 right 에서 시작
// 만약 low 와 high 가 교차하지 않았으면 data[low]를 data[high]와 교환
if (low < high)
Swap(array, low, high);
} while (low < high);
// low 와 high 가 교차했으면 반복문을 빠져나와 data[left]와 data[high]를 교환
Swap(array, left, high);
return high;
// Quicksort method: Conquer
void Conquer(double* array, int left, int right) {
if (left < right) {
int p = Partition(array, left, right);
Conquer(array, left, p - 1);
Conquer(array, p + 1, right);
// Insertionsort method: Insert
void Insert(double* array, int n) {
for (int i = 1; i < n; i++) {
if (array[i] < array[i - 1]) {
int j = i;
while (array[j] < array[j - 1]) {
Swap(array, j, j - 1);
if (j != 1)
j--;
C:₩Users₩kbj97₩OneDrive₩바탕 화면₩SortingTime₩Debug₩SortingTime.exe
[6] = 0.493
[7] = 0.617
[8] = 0.646
[9] = 0.792
InsertionSort: 0.0007ms
[0] = -0.997
[1] = -0.613
[2] = -0.299
[3] = -0.040
[4] = 0.127
[5] = 0.170
[6] = 0.493
[7] = 0.617
[8] = 0.646
[9] = 0.792
MergeSort: 0.0037ms
[0] = -0.997
[1] = -0.613
[2] = -0.298
[3] = -0.040
[4] = 0.127
[5] = 0.170
[6] = 0.493
[7] = 0.617
[8] = 0.646
[9] = 0.792

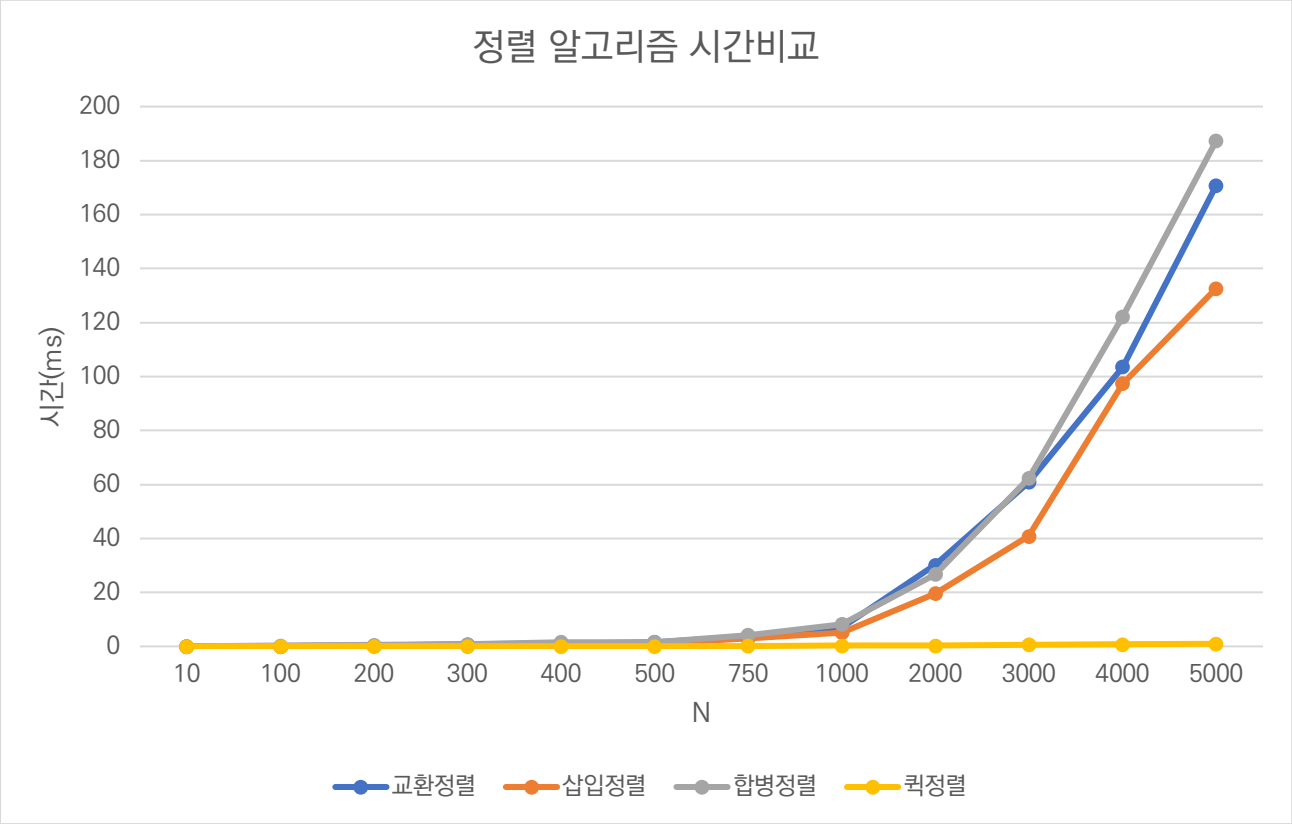
4. 정렬 결과


Microsoft Visual Studio 디버그 콘솔
Original Array:
[0] = -0.997
[1] = 0.127
[2] = -0.613
[3] = 0.617
[4] = 0.170
[5] = -0.040
[6] = -0.299
[7] = 0.792
[8] = 0.646
[9] = 0.493
========== N = 10 Sorting Time ==========
ExchangeSort: 0.0011sec
[0] = -0.997
[1] = -0.613
[2] = -0.299
[3] = -0.040
[4] = 0.127
[5] = 0.170
[6] = 0.493
[7] = 0.617
[8] = 0.646
[9] = 0.792
InsertionSort: 0.0010ms
[0] = -0.997
[1] = -0.613
[2] = -0.299
[3] = -0.040
[4] = 0.127
[5] = 0.170
[6] = 0.493
[7] = 0.617
[8] = 0.646
[9] = 0.792
MergeSort: 0.0102ms
[0] = -0.997
[1] = -0.613
[2] = -0.299
[3] = -0.040
[4] = 0.127
[5] = 0.170
[6] = 0.493
[7] = 0.617
[8] = 0.646
[9] = 0.792
QuickSort: 0.0011ms
[0] = -0.997
[1] = -0.613
[2] = -0.299
[3] = -0.040
[4] = 0.127
[5] = 0.170
[6] = 0.493
[7] = 0.617
[8] = 0.646
[9] = 0.792
C:₩Users₩kbj97₩OneDrive₩바탕 화면₩SortingTime₩Debug₩SortingTime.exe(18056 프로세스)이(가) 0 코드로 인해 종료되었습니

5. 출력 화면


Microsoft Visual Studio 디버그 콘솔
========== N = 10 Sorting Time ==========
ExchangeSort: 0.0008ms
InsertionSort: 0.0005ms
MergeSort: 0.0041ms
QuickSort: 0.0010ms
========== N = 100 Sorting Time ==========
ExchangeSort: 0.0808ms
InsertionSort: 0.0473ms
MergeSort: 0.2121ms
QuickSort: 0.0111ms
========== N = 200 Sorting Time ==========
ExchangeSort: 0.2185ms
InsertionSort: 0.1912ms
MergeSort: 0.5142ms
QuickSort: 0.0490ms
========== N = 300 Sorting Time ==========
ExchangeSort: 1.0521ms
InsertionSort: 0.4897ms
MergeSort: 0.7966ms
QuickSort: 0.0487ms
========== N = 400 Sorting Time ==========
ExchangeSort: 0.8714ms
InsertionSort: 0.8194ms
MergeSort: 1.1497ms
QuickSort: 0.0632ms
========== N = 500 Sorting Time ==========
ExchangeSort: 1.4296ms
InsertionSort: 1.1426ms
MergeSort: 2.0084ms
QuickSort: 0.0675ms
========== N = 750 Sorting Time ==========
ExchangeSort: 3.7381ms
InsertionSort: 3.4720ms
MergeSort: 5.1230ms
QuickSort: 0.1222ms
========== N = 1000 Sorting Time ==========
ExchangeSort: 6.0122ms
InsertionSort: 5.1371ms
MergeSort: 9.2422ms
QuickSort: 0.3023ms
========== N = 2000 Sorting Time ==========
ExchangeSort: 23.4966ms
InsertionSort: 23.0313ms
MergeSort: 26.9277ms
QuickSort: 0.3826ms
========== N = 3000 Sorting Time ==========
ExchangeSort: 61.4067ms
InsertionSort: 46.8677ms
MergeSort: 64.2011ms
QuickSort: 0.5544ms
========== N = 4000 Sorting Time ==========
ExchangeSort: 100.3309ms
InsertionSort: 96.9504ms
MergeSort: 165.8817ms
QuickSort: 0.8101ms
========== N = 5000 Sorting Time ==========
ExchangeSort: 171.9646ms
InsertionSort: 128.5434ms
MergeSort: 169.1203ms
QuickSort: 0.9831ms
C:₩Users₩kbj97₩OneDrive₩바탕 화면₩SortingTime₩Debug₩SortingTime.exe(20832 프로세스)이(가) 0 코드로 인해 종료되었습니다.

6. 그래프


정렬 알고리즘 시간비교
200
180
160
140
120
100
80
60
40
20
0
시간(ms)
10
100
200
300
400
500
750
1000
2000
3000
4000
5000
N
교환정렬
삽입정렬
합병정렬
퀵정렬