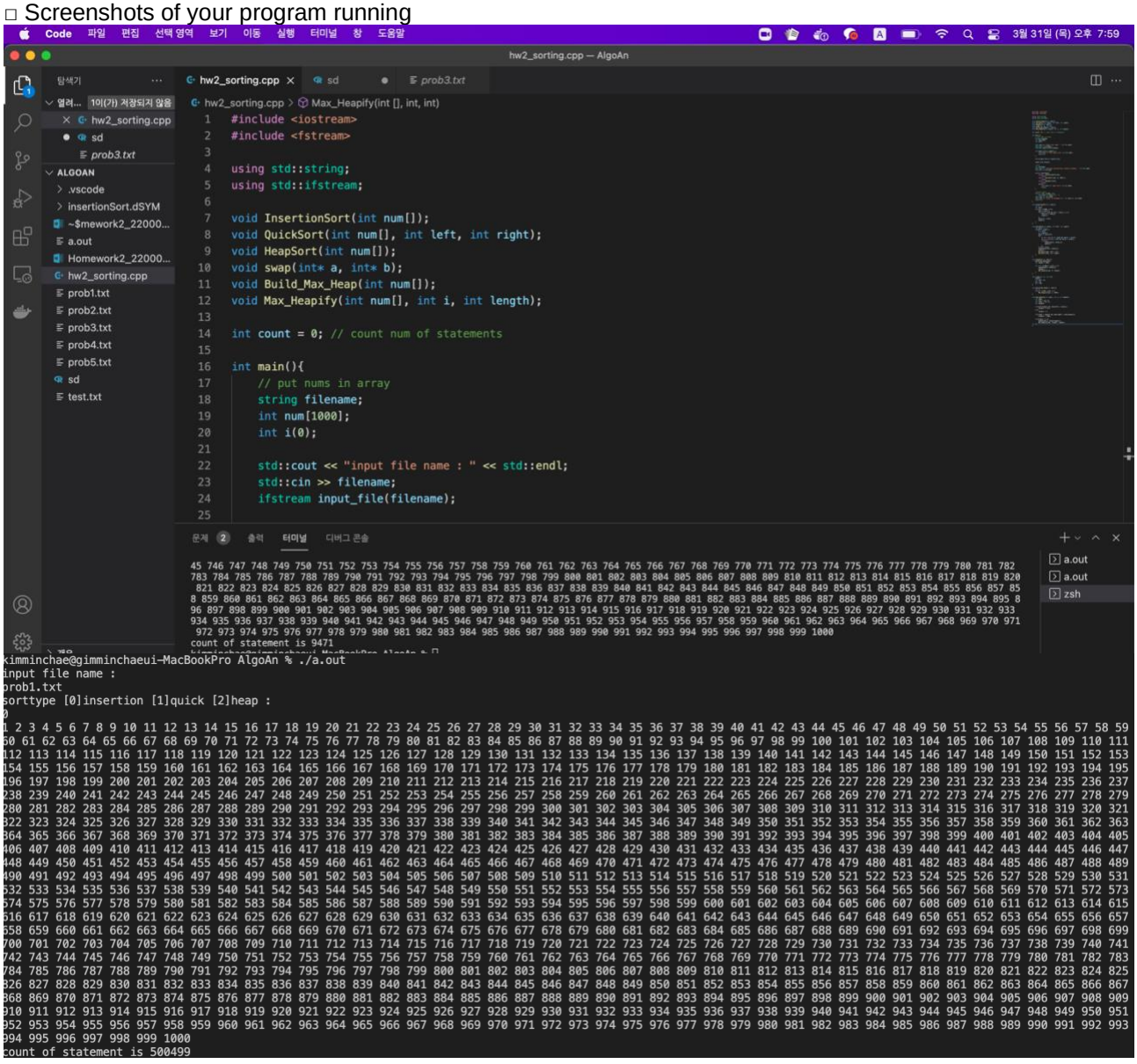
Algorithms analysis	Section	01
	Student number	22000080
Homework 2 – Sorting algorithms	Name	Kim, Min Chae

If your explanation is less informative and insufficient, then you may not get any points.
Also, you should provide discussion, otherwise you will get penalty.

□ Number of statements

Problem	Insertion sort	Heap sort	Quick sort
Prob 1	500499	8317	999
Prob 2	999	9709	999
Prob 3	253403	9081	2411
Prob 4	254571	9033	2385
Prob 5	64147	9471	1602

□ Screenshots of your program running



□ Discussion about the results

I created three sort functions along with the main function, and swap, Build_max_Heap and Max_Heapify functions used for the sorting function.

This program assumes that the txt file containing 1000 numbers is located in the same location as the executable.
It first asks the user the name of the file to be read, reads the number of the file, and stores it in an array. It then calls the sorting function by receiving input from the user which of the three algorithms to use. Finally, it outputs a sorted list and the number of statements.

1. insertion sort
Its algorithm is simple. It holds number one next, enters back from number zero, and then changes its position if the next is smaller than that. Repeat by increasing the following one by one. Count the next time you change its position. It has O(n^2) as the worst case and O(n) as the best case. Therefore, this is Θ(n^2).

2. quick sort
It makes use of recursion. It first receives the leftmost and rightmost sides. It makes i to the left and j to the right+1 until the left and right positions are normal. Then hold the pivot to the left. If i is less than or equal to the right and numi is less than pivot, continue to increase i. In addition, if j is greater than left and numj is greater than pivot, j continues to increase. At this time, if i<j, numi and numj are exchanged. Repeat this until i<j and switch left and j. Then recurs the quick sort for the left and right parts. Do the same thing to each partition untill all elements are sorted. Its worst case is Θ(n^2). This is when the list is already sorted. The best case is Θ(n log n). Therefore, its average case is O(n log n).

3. heap sort
The heap sort first creates a max heap. At this time, O(n) takes time. Then from the end, switch it with the first parent and perform Max_heapify as you go forward. Since the interval for performing Max heap is gradually reduced, O(log n) is repeated by n-1. Therefore, the average time of the heat sort is O(n log n).
To illustrate the most important Max_Heapify function here, after receiving the left and right, make the largest one to the left if it is larger than the i-th digit on the left. Otherwise, the biggest one is i. It then calculates largest by comparing the right side in the same way. If the largest is not equal to i, it recurs by changing the position of i with the largest.

□ Codes // you should also submit the separate executable C or C++ files, TA will try run your code.

```
#include <iostream>
#include <fstream>

using std::string;
using std::ifstream;

void InsertionSort(int num[]);
void QuickSort(int num[], int left, int right);
void HeapSort(int num[]);
void swap(int* a, int* b);
void Build_Max_Heap(int num[]);
void Max_Heapify(int num[], int i, int length);

int count = 0; // count num of statements

int main(){
	// put nums in array
	string filename;
	int num[1000];
	int i(0);

	std::cout << "input file name : " << std::endl;
	std::cin >> filename;
	ifstream input_file(filename);

	if(!input_file.is_open()) {
		std::cout << "file open error" << std::endl;
		return 1;
	}

	while(input_file >> num[i++]){};

	input_file.close();

	//sort
```

```
	int sorttype;
	std::cout << "sorttype [0]insertion [1]quick [2]heap : " << std::endl;
	std::cin >> sorttype;

	switch (sorttype){
		case 0 : InsertionSort(num);
			break;
		case 1 : QuickSort(num, 0, 1000-1);
			break;
		case 2 : HeapSort(num);
			break;
		default :
			std::cout << "type error" << std::endl;
			break;
	}

	//print result
	for(int j=0; j<1000; j++)
		std::cout << num[j] << " ";
	std::cout << std::endl;
	std::cout << "count of statement is " << count << std::endl;
	return 0;
}

void InsertionSort(int num[]){
	int i, j;
	int next;
	for(i=1; i<1000; i++){
		next = num[i];
		for(j=i-1; j>=0 && next < num[j]; j--){
			num[j+1] = num[j];
			count++;
		}
		num[j+1] = next;
		count++;
```

```
	}
}

void QuickSort(int num[], int left, int right){
	int pivot, i, j;
	if(left < right) {
		i = left;
		j = right+1;
		pivot = num[left];
		do{
			do i++; while(i <= right && num[i] < pivot);
			do j--; while(j > left && num[j] > pivot);
			if(i<j){
				swap(&num[i], &num[j]);
				count++;
			}
		} while (i<j);
		swap(&num[left], &num[j]);
		count++;
		QuickSort(num, left, j-1);
		QuickSort(num, j+1, right);
	}
}

void HeapSort(int num[]){
	Build_Max_Heap(num);
	int length = 1000;

	for (int i=1000-1; i>=0; i--){
		swap(&num[0], &num[i]);
		length--;
		Max_Heapify(num, 0, length);
	}
}
```

```
void swap(int* a, int* b){
	count++;
	int tmp = *a;
	*a = *b;
	*b = tmp;
}

void Build_Max_Heap(int num[]){
	int i;
	for(int i=499; i>=0; i--)
		Max_Heapify(num, i, 1000);
}

void Max_Heapify(int num[], int i, int length){
	int *tmp;
	int left = 2*i +1;
	int right = 2*i +2;
	int largest;

	if((left<length) && (num[left] > num[i]))
		largest = left;
	else
		largest = i;

	if((right < length) && (num[right] > num[largest]))
		largest = right;

	if(largest != i){
		swap(&num[i], &num[largest]);
		Max_Heapify(num, largest, length);
	}
```