알고리즘 2차 과제

선형탐색, 합병정렬, 이진탐색

2020. 10. 08. 목

컴퓨터공학과

2019305059

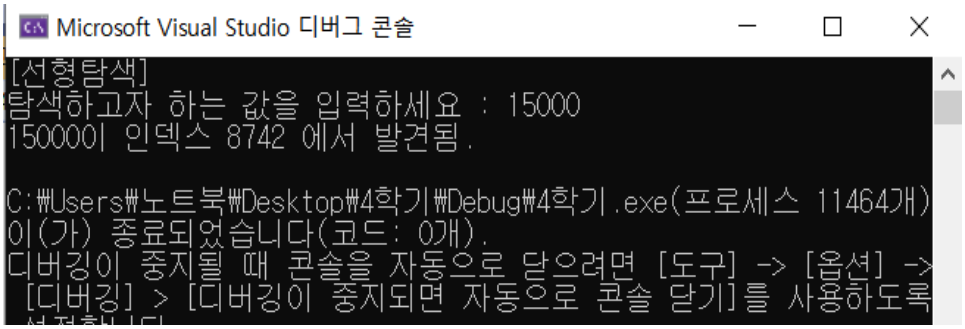
이현수

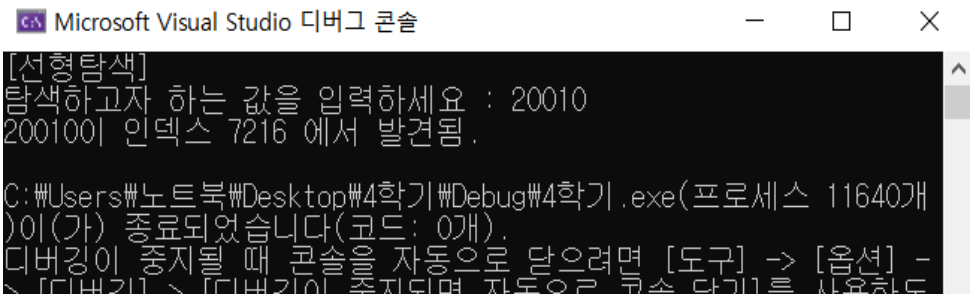
[선형탐색 - 소스코드]

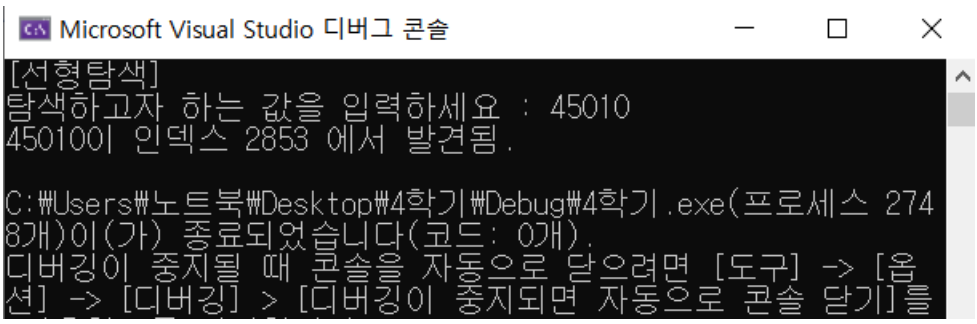
```
#include<iostream>
#include<cstdlib>
#include<ctime>
#define SIZE 10000
using namespace std;

int sequential_search(int *data, int size, int target) //선형탐색 함수
{
	for (int i = 0; i < size; i++)
	{
		if (data[i] == target) return i; //찾고자하는 값이 발견되면 인덱스반환 후 함수종료
	}
	return -1; //발견되지 않으면 -1 반환 후 함수종료
}

int main(void)
{
	srand((unsigned)time(NULL)); //난수를 생성하기 위한 코드

	int *data = new int[SIZE]; //배열 동적할당

	for (int i = 0; i < SIZE; i++)
	{
		//반복문으로 배열에 0~49999난수생성
		data[i] = (int)(((long)rand() << 15) | rand()) % 50000;
	}

	cout << "[선형탐색]" << endl;
	cout << "탐색하고자 하는 값을 입력하세요 : ";
	int target; cin >> target;	//탐색하려는 값 입력

	//선형탐색 함수 호출. 변수i에 반환된 인덱스값 저장
	int i = sequential_search(data, SIZE, target);

	if (i == -1) //반환받은 값이 -1이라면==찾으려는값이 없다면
	{
		cout << "발견되지 않음." << endl;
	}
	else  //찾으려는 값이 있다면
	{
		cout << target << "이 " << "인덱스 " << i << " 에서 발견됨." << endl;
	}

	delete[]data; //배열 해체
}
```

[선형탐색 – 실행]

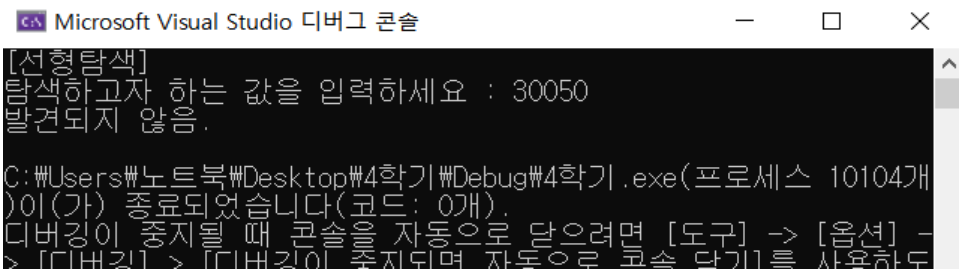
(값이 찾아지는 경우)

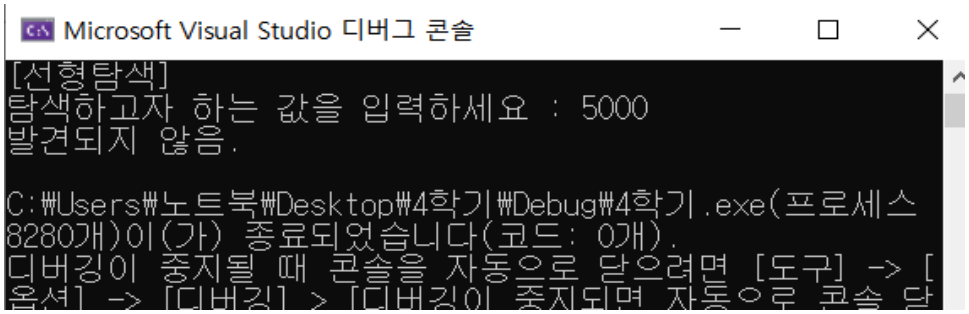






(값이 찾아지지 않는 경우)





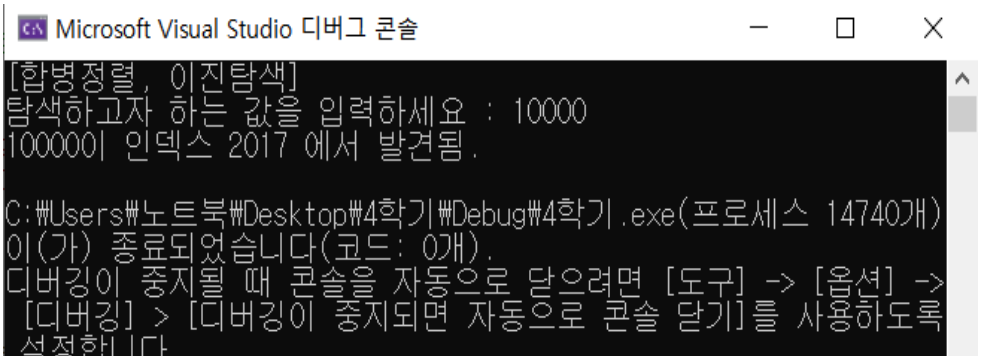
[합병정렬, 이진탐색 - 소스코드]

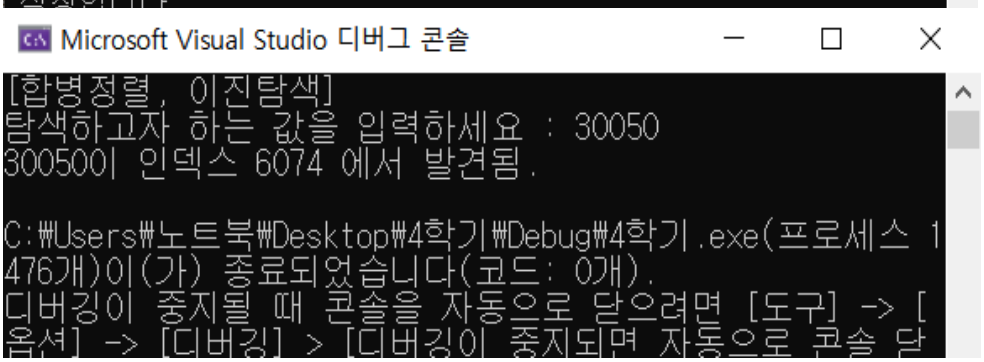
```
#include<iostream>
#include<cstdlib>
#include<ctime>
#define SIZE 10000
using namespace std;

int Binary_search(int *data, int first, int last, int target)
{
	if (first > last) //찾으려는 값(target)이 없다면
	{
		return -1;  // -1 반환
	}

	int mid = (first + last) / 2; //배열 중간인덱스 구하기

	if (data[mid] == target) //배열중간값과 target이 일치하면
	{
		return mid; //mid인덱스 반환
	}
	else if (target < data[mid]) //찾으려는 값이 배열중간값보다 작을때
	{
		return Binary_search(data, first, mid - 1, target);
	}
	else //찾으려는 값이 배열중간값보다 클때
	{
		return Binary_search(data, mid + 1, last, target);
	}
}

void MergeTwoArea(int *data, int left, int mid, int right) //합병시키는 함수
{
	int fIdx = left;    //앞부분 시작인덱스
	int rIdx = mid + 1; //뒷부분 시작인덱스

	int* sortArr = new int[right + 1]; //right+1 크기의 합병결과 저장할 배열 동적할당
	int sIdx = left;  //합병결과 배열 시작인덱스

	//앞부분,뒷부분 중 한개라도 합병배열에 완전히 옮겨지지 않은경우 계속반복
	while (fIdx <= mid && rIdx <= right)
	{
		if (data[fIdx] <= data[rIdx]) //앞부분배열 값이 뒷부분배열 값보다 같거나 더 작을때
		{
			sortArr[sIdx] = data[fIdx++]; //합병배열에 앞부분배열값 저장 후 앞부분배열인덱스+1
		}
		else //뒷부분배열의 값이 앞부분배열 값보다 작을 때
		{
			sortArr[sIdx] = data[rIdx++]; //합병배열에 뒷부분배열값 저장 후 뒷부분배열인덱스+1
		}

		sIdx++; //합병배열 인덱스+1
	}

	if (fIdx > mid) //앞부분 배열이 모두 옮겨졌을 경우
	{
		for (int i = rIdx; i <= right; i++, sIdx++)
		{
			sortArr[sIdx] = data[i]; //뒷부분배열 나머지 옮기기
		}
	}
	else //뒷부분 배열이 모두 옮겨졌을 경우
	{
		for (int i = fIdx; i <= mid; i++, sIdx++)
		{
			sortArr[sIdx] = data[i]; //앞부분배열 나머지 옮기기
		}
```

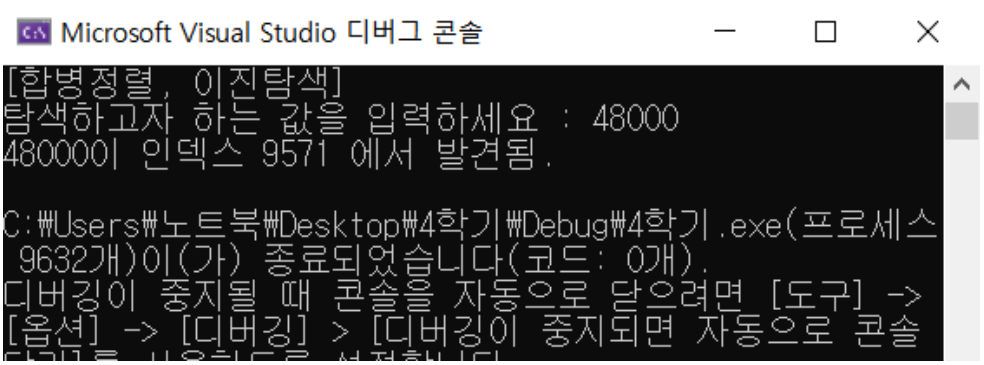
```
        }

        for (int i = left; i <= right; i++)
        {
                data[i] = sortArr[i]; //원본배열에 합병한 결과 저장
        }

        delete []sortArr; //배열 해체
}




void MergeSort(int *data, int left, int right)
{
        if (left < right)
        {
                int mid = (left + right) / 2;    //배열의 중간 인덱스 구하기

                MergeSort(data, left, mid);      //앞부분 재귀호출
                MergeSort(data, mid + 1, right); //뒷부분 재귀호출

                MergeTwoArea(data, left, mid, right); //앞부분+뒷부분 합병시키는 함수 호출
        }
}




int main(void)
{
        srand((unsigned)time(NULL)); //난수생성을 위한 선언

        int *data = new int[SIZE]; //배열 동적 할당

        for (int i = 0; i < SIZE; i++)
        {
                //반복문으로 배열에 0~49999난수생성
                data[i] = (int)(((long)rand() << 15) | rand()) % 50000;
        }

        cout << "[합병정렬, 이진탐색]" << endl;
        cout << "탐색하고자 하는 값을 입력하세요 : ";
        int target; cin >> target; //찾으려는 값 입력

        MergeSort(data, 0, SIZE-1); //합병정렬함수

        int i = Binary_search(data, 0, SIZE-1, target); //이진탐색으로 변수i에 인덱스 반환

        if (i == -1) //반환받은 값이 -1이라면==찾으려는값이 없다면
        {
                cout << "발견되지 않음." << endl;
        }
        else //찾으려는 값이 있다면
        {
                cout << target << "이 " << "인덱스 " << i << " 에서 발견됨." << endl;
        }

        delete[]data; //배열 해체
}
```

[합병정렬, 이진탐색 – 실행]

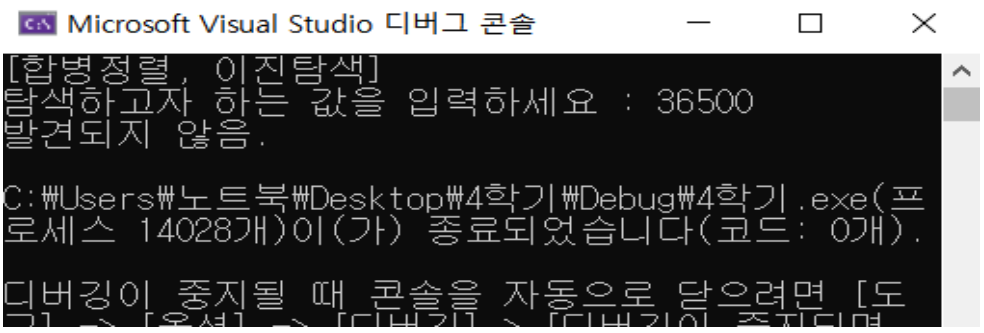
(값이 찾아지는 경우)

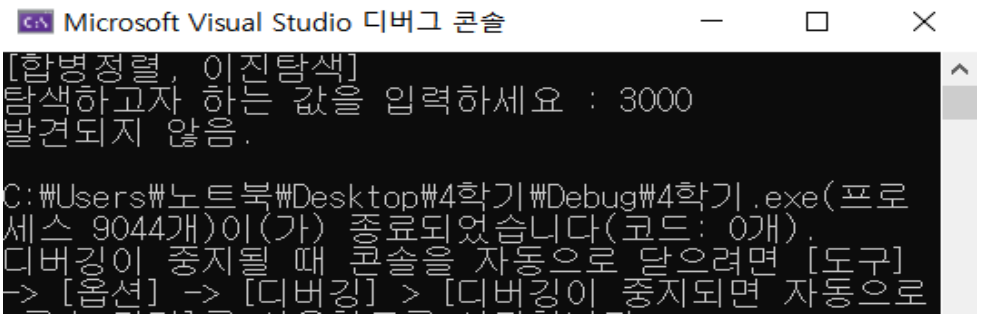






(값이 찾아지지 않는경우)





[과제 수행 소감]

이번 과제를 하면서 rand 함수값은 기본적으로 32767 까지밖에 출력이 안된다는 것을 알게되었다.
그래서 0~49999 난수를 생성하기 위해서 인터넷에서 검색을 해보다가 더 많은 범위까지 난수를 생성하는 방법을 알게되었다. 그동안 난수를 생성할 때 이렇게 큰 숫자를 생성해본적이 없어서 rand()%숫자 를 하면 원하는 무한대 값 난수가 생성되는 줄 알았는데 그게 아니라는 것을 알게되어 유익했다.

수업시간에 이론으로배운 순차탐색, 이진탐색, 합병정렬을 직접 코딩해서 정상적으로 정렬, 탐색되는 것을 확인해서 의미있었고, 과제를 하면서 공부가 되어 유익했다.