Algorithm

1. Class Date (information을 저장하는 클래스)

```cpp
//201624588 조영우 -> 알고리즘 과제2
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
using namespace std;

class Date {
private:
    int start;
    int end;
    int value;
public:
    Date() : start(0), end(0), value(0) {}
    Date(int s, int e, int v) :start(s), end(e), value(v) {}

    int getStart() const { return start; }
    int getEnd() const { return end; }
    int getValue() const { return value; }

    void setStart(int _start) { start = _start; }
    void setEnd(int _end) { end = _end; }
    void setValue(int _value) { value = _value; }

    bool operator<(Date s) const {
        return this->end < s.getEnd();
    }

    void print() const {
        cout << start << ' ' << end << ' ' << value;
    }
};
```

입력되는 정보를 저장하는 객체를 생성하기 위해 class를 정의하였고, 입력되는 데이터의 크기가 최대 1,000,000,000이기에 최대 2,147,483,647의 값을 저장할 수 있는 int type을 지정하였다.

bool operator<(Date s) const; // sort과정에서 객체를 비교할 때 끝나는 날 기준으로 sort하기 위해 operator overloading 하였다. (algorithm library에 구현되어 있는 sort는 O(nlogn)의 time complexity를 갖는 Quick sort로 구현되어 있다.)

2. Class Money (정보들을 저장하고 관리하는 클래스)

```
class Money {
private:
    int n;
    long long output;
    vector<Date> v;
public:
    Money();
    ~Money() { vector<Date>().swap(v); }
    int getSize() const { return n; }
    long long getOutput() const { return output; }

    void printDate() const {
        cout << n << endl; //size
        for (auto& data : v) { //vector의 모든요소 출력
            data.print();
            cout << endl;
        }
    }
    void printOutput() const {
        cout << endl << endl;
        cout << "output" << endl << output << endl;
    }

    int binarySearch(int first, int last, int pivot);
    int binarySearch(int current);

    void maxEarn(); //최대비용 구하는 함수
};
```

(1) Date객체를 저장하고 관리하기 위해 class Money를 정의한다.

(2) 최종 output 결과가 int type으로 담을 수 없는 데이터의 크기 이기에 long long type으로 선언하여 멤버변수로 선언하였다.

(3) constructor(생성자)에서는 객체생성 시 정보에 대한 입력이 이루어 져서 멤버변수에 대한 초기화가 이루어 지도록 구현하였다. (뒤페이지에 구현)

(4) destructor(소멸자)에서는 vector의 capacity를 제거하기 위한 구현이다.

(5) binarySearch(); 최대비용을 구할 때 현재의 비용이 최대비용 안에 포함된 비용이 되려면 이전 step 중 현재 index와 일정이 겹치지 않는 그때의 최대비용 존재할 것이며, 그 step에 해당하는 index를 구하는 함수이다. (Time Complexity O(log)을 갖는 Search, 뒷페이지에 구현)

(6) Void maxEarn(); 프로그램의 알고리즘이 구현되어 있는 함수이다. 최대로 벌어들일 수 있는 액수는 함수가 호출되면 알고리즘에 의해 계산되어 money객체의 멤버변수 output에 저장된다. (뒤페이지에 구현)

3. Constructor of class Money (input)

```
Money::Money() : n(0), output(0) {
    /*
    //file I/O for testfile
    ifstream ifs;
    ifs.open("test_input3.txt");
    if (ifs.is_open()) {
        ifs >> n;
        v.reserve(n);

        Date tmp;
        int tmp1, tmp2, tmp3;
        for (int i = 0; i < n; i++) {
            ifs >> tmp1 >> tmp2 >> tmp3;
            tmp.setStart(tmp1);
            tmp.setEnd(tmp2);
            tmp.setValue(tmp3);
            v.push_back(tmp);
        }
    }
    ifs.close();
    */
    cin >> n;
    v.reserve(n);
    Date tmp;
    int tmp1, tmp2, tmp3;

    for (int i = 0; i < n; i++) {
        cin >> tmp1 >> tmp2 >> tmp3;
        tmp.setStart(tmp1);
        tmp.setEnd(tmp2);
        tmp.setValue(tmp3);
        v.push_back(tmp);
    }
}
```

(주석으로 표현된 라인은 test파일 입출력에 사용한 구현이다.)

(1) 데이터 묶음의 Size에 해당하는 데이터를 멤버변수 n에 저장한다.

(2) 입력될 데이터의 크기를 파악했기에 입력이 될 때마다 capacity를 할당하는 것은 비효율적이기에 n사이즈에 해당하는 만큼의 capacity를 vector에 할당한다.

(3) 임시 저장 공간으로 사용할 Date 객체와 int형 변수 3개를 선언한다.

(4) 입력되는 데이터를 차례로 받아 객체에 저장한다.

(5) tmp객체를 vector<Date> v에 push_back(tmp)하여 데이터를 저장한다.

4. Money::binarySearch() → O(logn)을 갖는 Search

```
int Money::binarySearch(int first, int last, int pivot) {
    int observe = (first + last) / 2;

    if (first + 1 < last) { //3개 이상일때
        if (v[pivot].getStart() > v[observe].getEnd()) {
            return binarySearch(observe, last, pivot);
        }
        else {
            return binarySearch(first, observe - 1, pivot);
        }
    }
    else { //1개 or 2개
        if (v[pivot].getStart() > v[last].getEnd()) { return last; }
        else if (v[pivot].getStart() > v[first].getEnd()) { return first; }
        else { return -1; }
    }
}

int Money::binarySearch(int current) {
    return binarySearch(0, current - 1, current);
}
```

현재 index(indexOfCurrent)를 pivot으로 하여, 이 pivot index의 시작날짜보다 작으며 겹쳐질 수 있는 step중에서 가장 최대비용을 가지는 index를 return 하는 것이 목표인 함수이다.

(1) First와 last의 중간 값을 갖는 index의 끝나는 날짜와 pivot에 해당하는 객체의 시작날짜를 비교하여 데이터를 절반씩 줄여서 함수를 recursive 호출한다.

(2) 여기서 위의 재귀호출은 observe 또한 목표index가 될 수 있기에 포함하여 범위에 포함하였고, 밑의 재귀호출은 pivot의 시작날짜가 observe의 끝나는 날짜보다 작거나 같기 때문에 기간이 겹쳐서 목표index가 될 수 없기에 범위에서 제외시켰다.

(3) 벡터의 객체들이 쪼개져서 1개 또는 2개가 되었다면, last와 first의 끝나는 날짜와 pivot의 시작날짜에 겹치지 않게 들어갈 수 있는 상태라면 해당 index를 return 한다.

(4) 여기서 쪼개진 벡터는 끝에까지 쪼개지더라도 최적의 step의 index를 못 구하는 경우가 발생한다. 예를 들어 과제에서 주어진 test_input2 file에서 추가적으로 데이터가 존재한다면 <시작날짜:1, 끝나는 날짜:100> 이 데이터의 시작날짜보다 작은 끝나는 날짜를 발견하지 못한다면 논리적 오류가 발생할 것이다. 그래서 이런 경우와 같이 적절한 index를 구하지 못했다면 -1을 return 하여 적절한 조치가 필요하다.

일반적으로 발견즉시 return 하는 binarySearch와 다르게 이 함수는 최적의 값을 찾는 것이기에 해당 범위에 맞게 함수호출을 해주고 완전히 쪼개고 난 뒤 조건문으로 알맞게 return한다.

binarySearch()의 Worse-case에 해당하며, Time Complexity O(logn)을 갖는다.

5. Money::maxEarn() → 알고리즘 구현

```
void Money::maxEarn() {
    sort(v.begin(), v.end()); //sort algorithm은 Quick sort이며 O(nlogn)이다.

    long long* optimal = new long long[n] {0}; //현재상태의 최대값을 넣을 배열
    optimal[0] = v[0].getValue(); //초기화
    int indexOfCurrent = 1; //현재 index
    int k; //요놈잡았다 index

    while (indexOfCurrent < n) {
        k = binarySearch(indexOfCurrent); //이전step중 최적의 index를 계산해온다.

        if (k < 0) {
            if (v[indexOfCurrent].getValue() > optimal[indexOfCurrent - 1]) {
                optimal[indexOfCurrent] = v[indexOfCurrent].getValue();
            }
            else {
                optimal[indexOfCurrent] = optimal[indexOfCurrent - 1];
            }
            indexOfCurrent++;
            continue;
        }

        // if(넣었을때의 값 > 안넣을때의 값)
        if (optimal[k] + v[indexOfCurrent].getValue() > optimal[indexOfCurrent - 1]) {
            optimal[indexOfCurrent] = optimal[k] + v[indexOfCurrent].getValue();
        }
        else {
            optimal[indexOfCurrent] = optimal[indexOfCurrent - 1];
        }
        indexOfCurrent++;
    }
    output = optimal[n - 1];
}
```

(1) 끝나는 날 기준으로 quick sort를 시행한다.

(2) 0~n-1 step에서 각각의 step에서의 최대비용을 넣을 배열을 선언한다.

(3) 현재 index에서의 값을 넣었을 경우 현재 step의 index와 겹치지 않고 더해질 수 있는 이전 step의 index **k**를 선언한다.

(4) While문의 경우 현재의 index가 n이 되는 순간까지 반복한다.

(5) K = binarySearch(indexOfCurrent)의 코드는 만약 현재 index를 의미하는 indexOfCurrent의 값이 최대비용을 계산하는데 더해져야 할 항목이라면, 더해질 수 있는 비용 중 가장 큰 비용이 저장된 step의 index를 k에 저장함을 의미한다.

(6) 위의 (5)에서 올바른 k의 값을 구하지 못해서 음수가 된 경우에 대한 조건문이다. K가 음수가 되었다는 것은 현재 객체의 시작날짜는 이전의 객체의 끝나는 날짜와 모두 겹치게

됨을 의미한다. 그렇기에 현재 step에 해당하는 객체의 비용과 바로 전 step에서의 최대 비용을 비교하여 큰 값을 현재최대비용에 해당하는 optimal[indexOfCurrent]에 저장한다. 그 후, 현재 index의 값을 +1하고 해당 (4)의 조건문을 다시 시행하도록 continue 한다. (step2의 경우는 0,1 step에서 이런 경우가 존재하지만, step3에서는 존재하지 않았다.)

(7) 마지막 if-else문에 해당하며, 위의 오류가 생기지 않고 정상적인 k값을 구했다면 실행되는 조건문이다. 현재step에서의 객체의 비용이 들어올지 말지를 결정하는 것이다. 들어오지 않았을 때의 값이 크다면 optimal[indexOfCurrent] = optimal[indexOfCurrent - 1]; 으로 이전 step에서의 값을 그대로 넣으면 된다. **(else에 해당한다.)**

(8) (7)에서의 현재step에서의 객체의 비용이 들어오는 것이 더 크다면, 현재step에서의 객체에서 시작하는 날짜와 겹치지 않는 끝나는 날짜를 갖는 step에서의 index k를 이용하여, k step에서의 최대비용에서 현재 객체의 비용을 더하여, 현재 step에서의 최대값에 저장한다. optimal[indexOfCurrent] = optimal[k] + v[indexOfCurrent].getValue(); **(if문에 해당한다.)**

(9) 마지막 if-else문을 빠져나왔다면 현재의 index를 +1하고 반복문을 계속 시행한다.

(10) 이제 while문에서의 반복과 조건이 모두 끝나면 각각의 step에서의 최대 비용의 값을 가지는 optimal 배열에서 (n-1)번째 step에서는 최종최대비용을 저장하고 있다. 그것을 멤버변수 output에 저장하며, 함수를 종료한다.

6. Main 함수

```
int main() {
    Money money;
    //money.printDate(); //입력된 사이즈와 데이터 출력

    money.maxEarn();
    money.printOutput();
    return 0;
}
```

(1) Class Money의 객체 money를 생성하면, constructor에서 알맞은 Input이 실행되고, 데이터를 저장하고 있다.

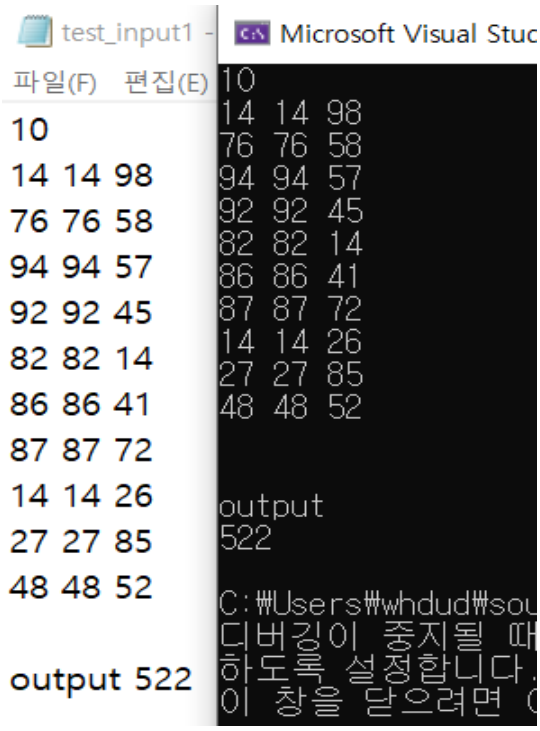
(2) maxEarn(); 함수로 최대비용을 계산하여 멤버변수 output에 저장한다. 그리고 printOutput(); 함수로 output멤버변수를 출력한다.

(3) 객체기반 프로그래밍으로 캡슐화와 정보은닉의 효과를 높여서 구현하였다.

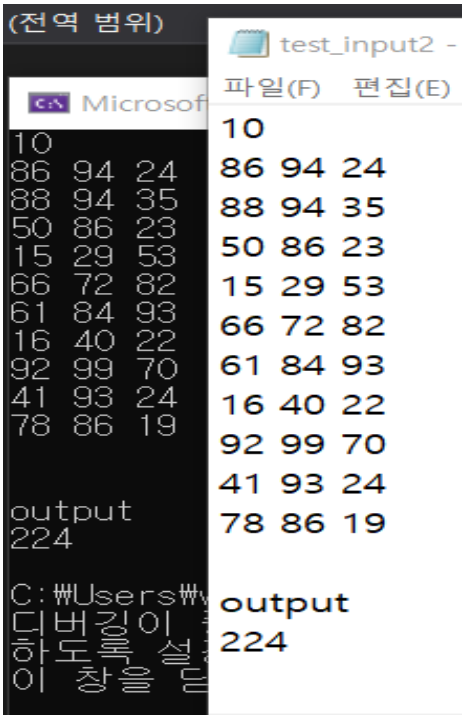
(4) **실행결과는 주석 처리된, 함수까지 실행한 결과를 캡쳐한 것이다.**

7. 실행 결과

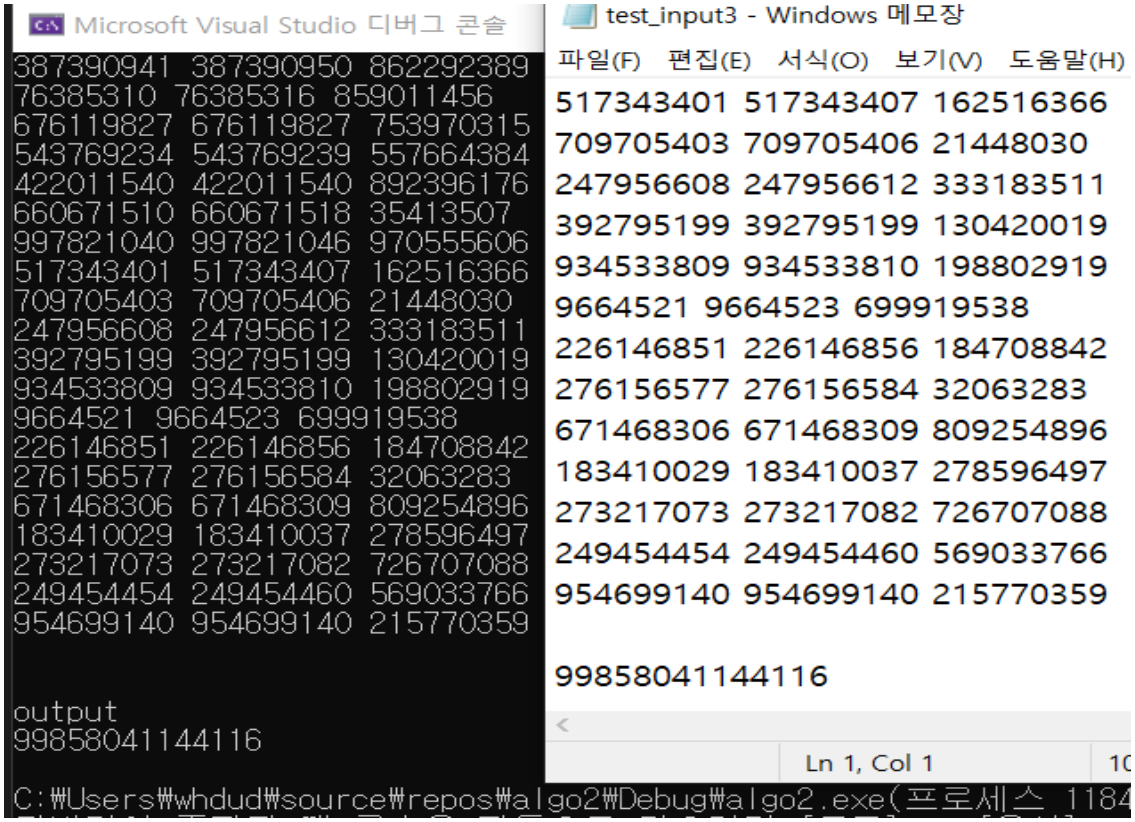
[test1]



[test2]



[test3]



8. 결론

(1) 사용한 메모리

- Test3의 경우를 살펴보면 200,000개의 Date 객체를 갖고 있고, Date 객체는 int type 변수 3개를 갖고 있다.
- 그래서 4*3*200,000 = 2,400,000 byte = 2.4 MB의 메모리를 사용하고 있다.
- 각각의 step 에서의 최대비용을 계산한 배열 optimal 은 long long type의 변수가 200,000개 존재하기에 1.6MB를 사용하였다.
- 세부적으로 사용한 메모리들은 바이트 단위이기에 MB단위에서는 아주 작은 값이다.
- Sort는 Quick sort를 이용해서 메모리는 사용하지 않았다.
- BinarySearch는 recursive 호출이기에 메모리를 사용하지 않았다.
- 따라서 사용한 메모리는 5MB를 넘지 않는다.

(2) 개발 환경

- Visual Studio 2019

(3) Time Complexity: O(nlogn)

- n개의 데이터를 받았고, 해당 알고리즘에서 Quicksort가 시행되어 O(nlogn)을 갖는다.
- 최적의 index를 구하는 BinarySearch를 시행하였고, 최대비용을 계산하기 위해 n개의 데이터를 while문으로 반복하였다.
- O(logn)을 갖는 BinarySearch를 n번 반복하였기에 time complexity는 O(nlogn)이다.