

Data Structures

Lab # 14



- 1. Quick Sort
- 2. Merge Sort
- 3. Tag Sort
- 4. Hashing

■ 실습에서 사용할 파일

- ❖ Lab13에서 사용한 Student.h, Student.cpp, Swap.cpp
- ❖ ...₩labplus_CRLF₩labplus₩Lab, C++ 3rd₩Chapter10₩Sorts

1. Quick Sort

■ 문 제

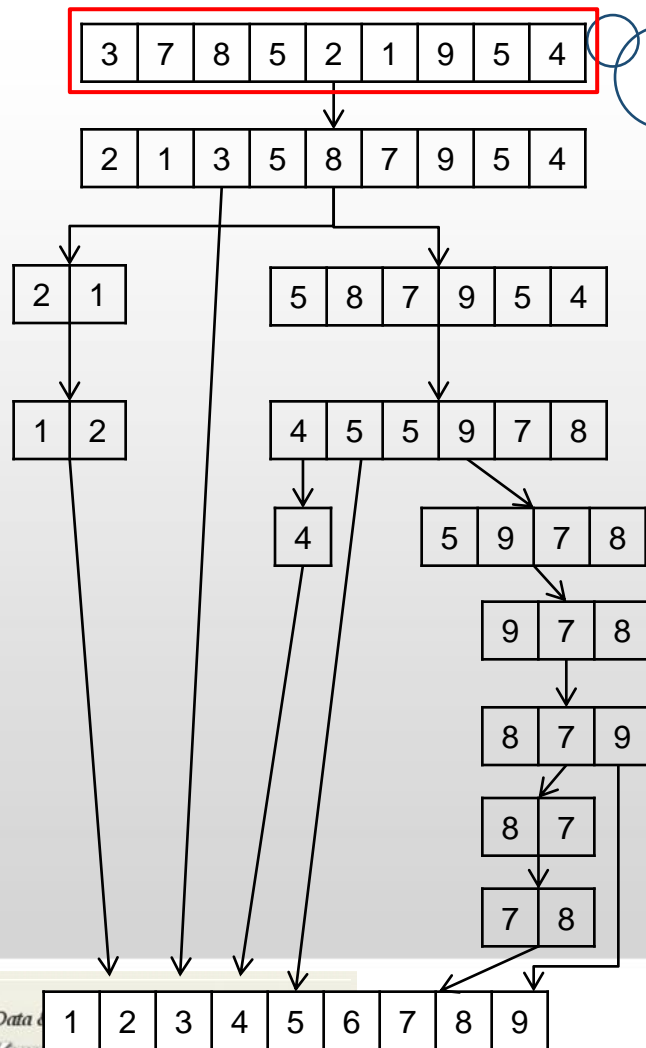
- ❖ 학생들의 정보를 담고있는 배열을 Quick Sort 기법을 이용하여 정렬하시오.
 - 정렬의 기준 : 이름, 오름차순 (문자열 비교함수의 결과를 이용하여 정렬함)
- ❖ QuickSort 함수를 QuickSort.cpp 파일에 구현할 것

■ 참고 사항

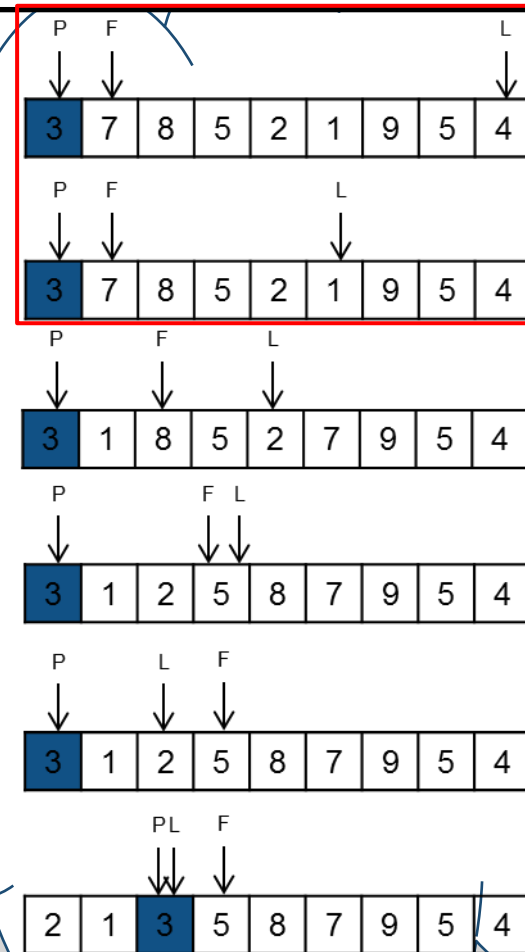
- ❖ QuickSort.cpp 파일을 이용하여 구현할 것

■ Quick Sort 예제

❖ P : Pivot, F : first, L : Last

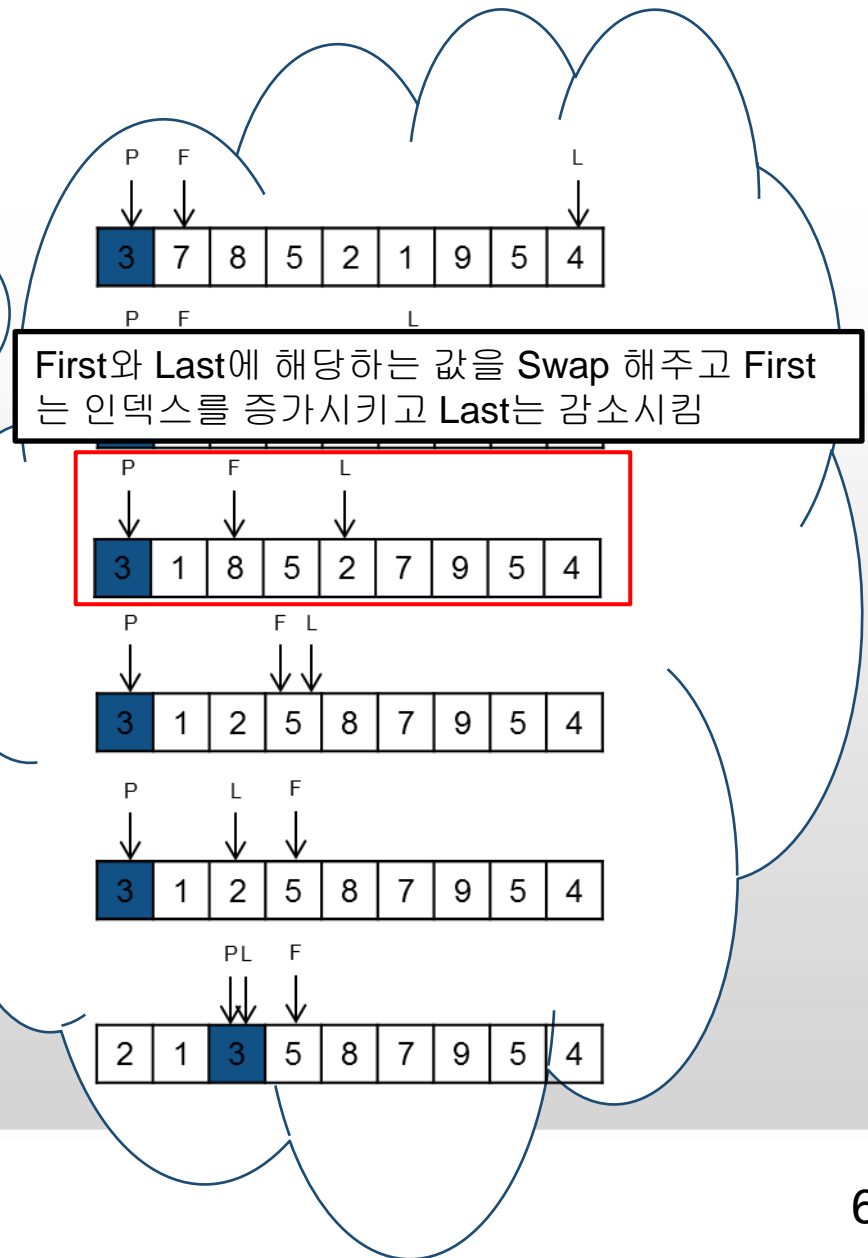
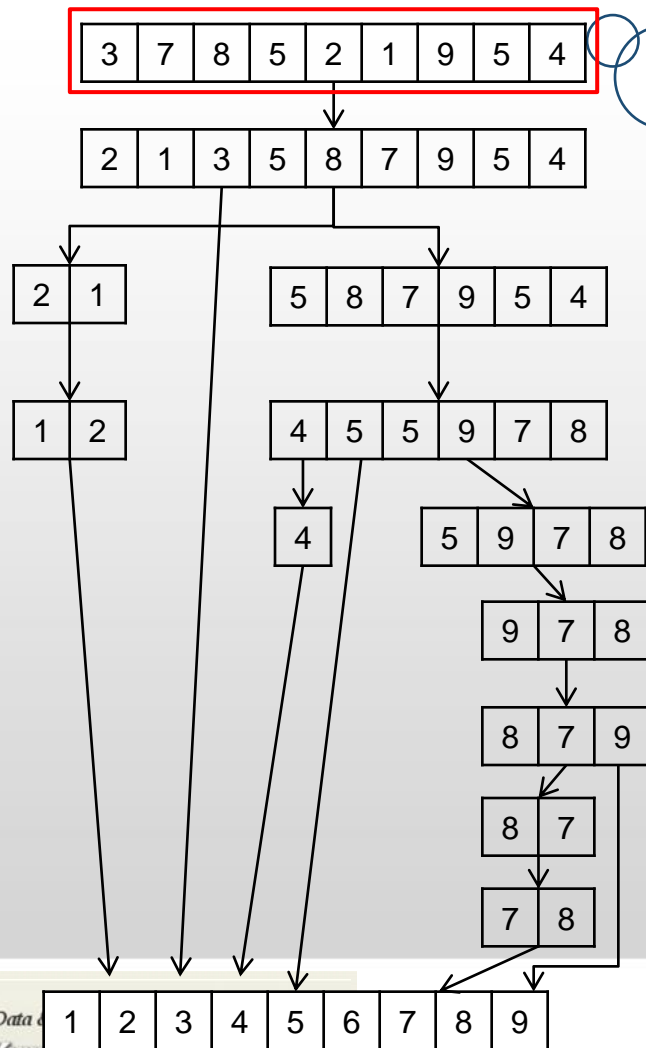


1. Pivot과 First 값을 비교함
만약 First가 더 작다면 First 값을 증가시킴
2. Pivot과 Last값을 비교함
만약 Last값이 더 크다면 Last값을 감소시킴



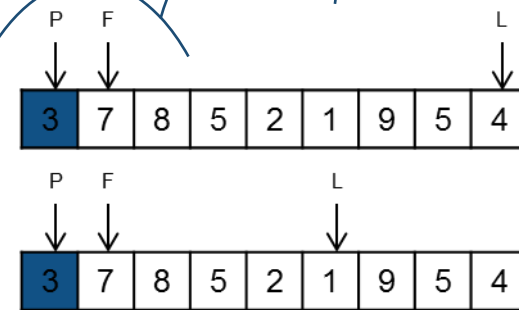
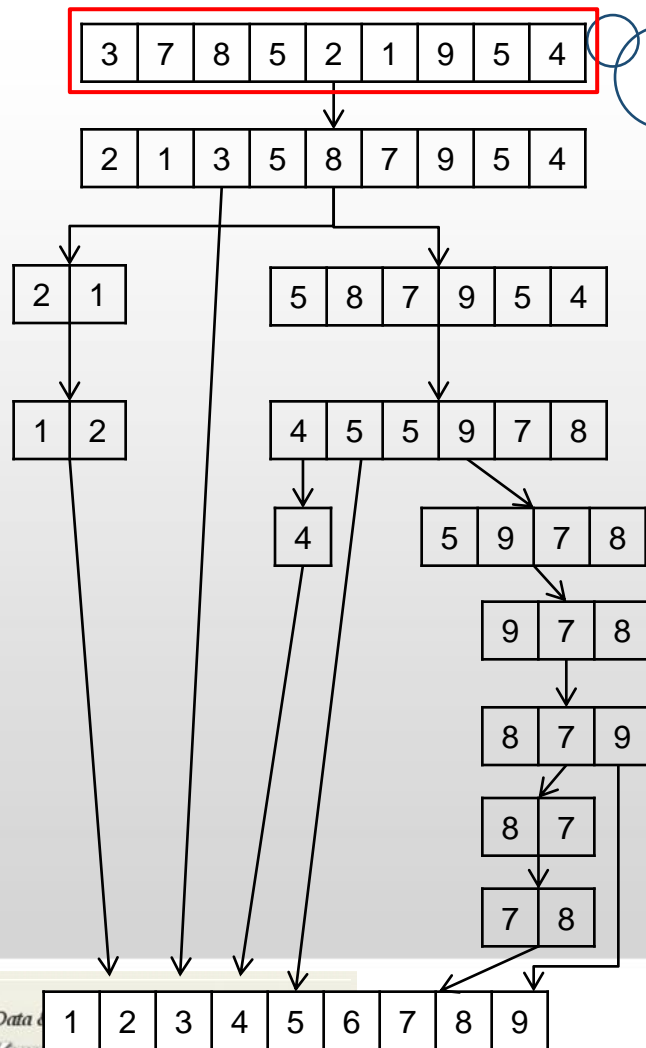
■ Quick Sort 예제

❖ P : Pivot, F : first, L : Last

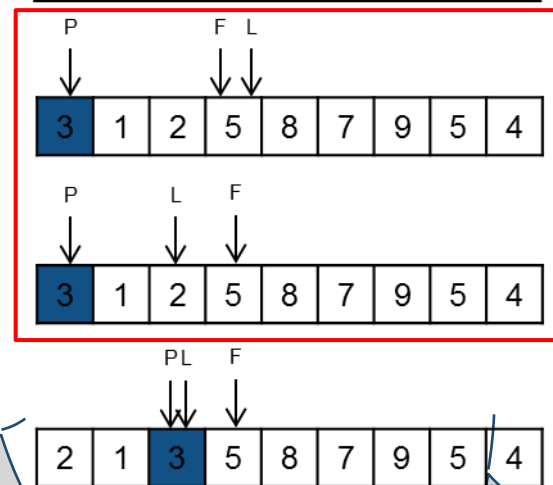


■ Quick Sort 예제

❖ P : Pivot, F : first, L : Last

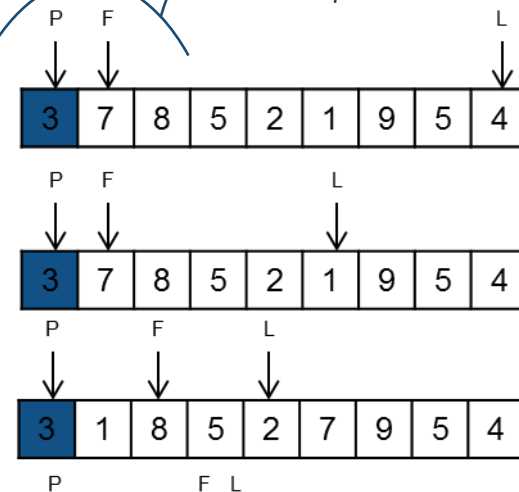
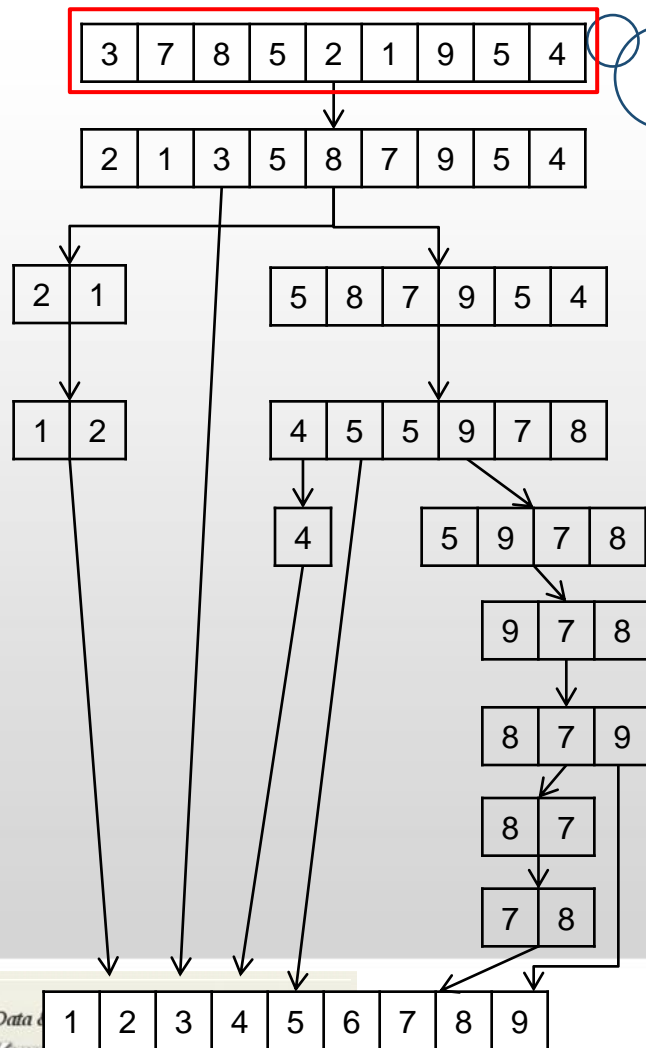


위 과정을 Last값이 First값보다 작아질 때까지 반복하여 수행한다.

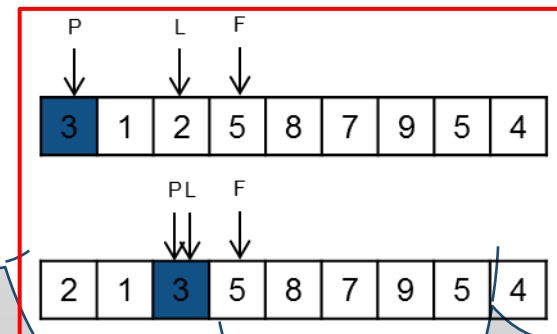


■ Quick Sort 예제

❖ P : Pivot, F : first, L : Last

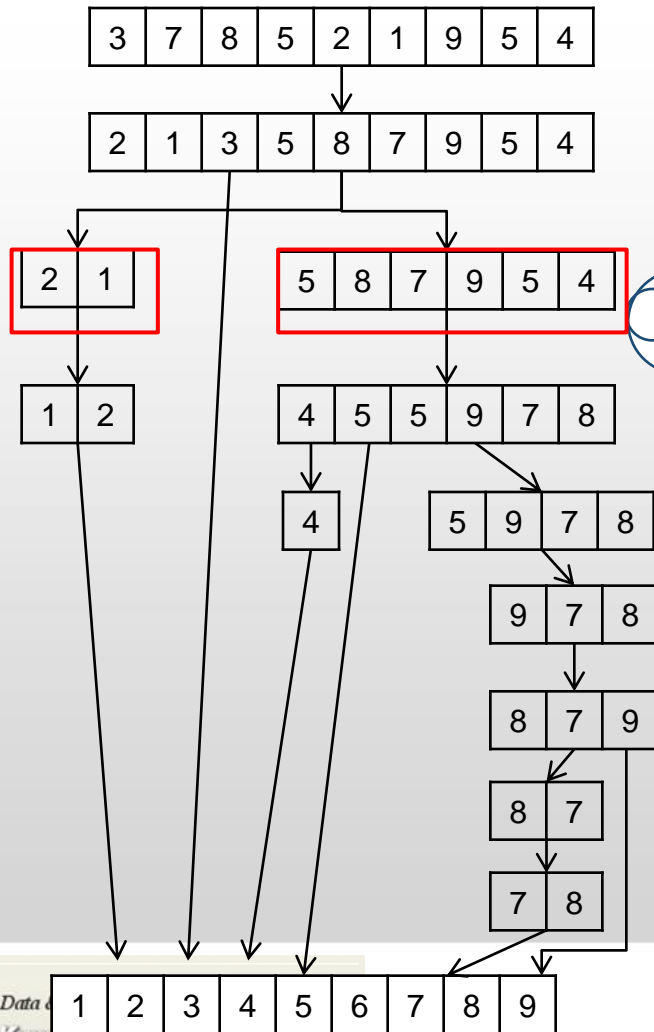


Last와 Pivot에 해당하는 값을 **Swap**하고
Pivot을 기준으로 배열을 두개의 영역으로 나눈다

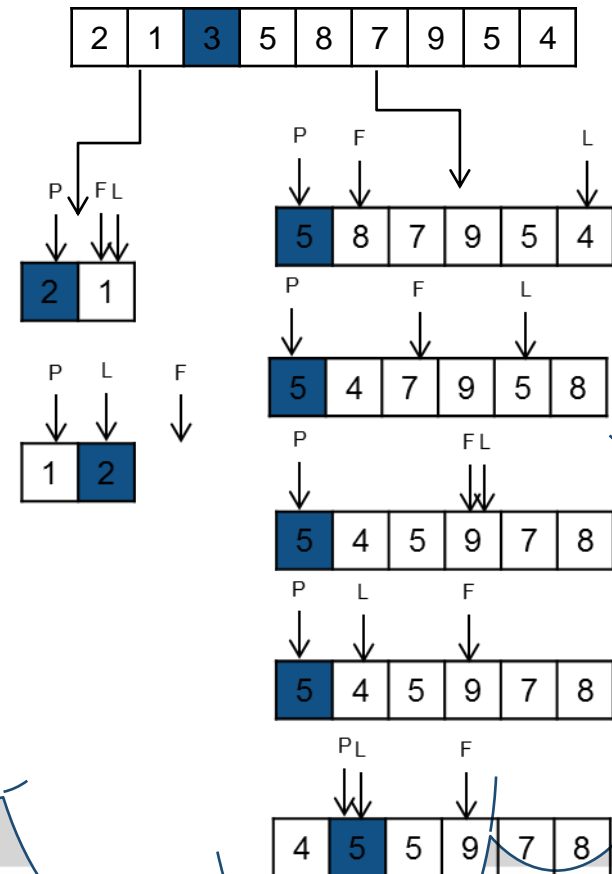


■ Quick Sort 예제

❖ P : Pivot, F : first, L : Last



각각의 영역에 대해서 이전 과정들을 더 이상 분할하지 못할때까지 수행하고 분할한값들을 다시 하나의 결과로 합친다.



■ QuickSort 예제 소스코드 (Split 함수)

```
#include "QuickSort.h"

void Split(Student values[], int first, int last, int& splitPoint)
{
    //비교 기준 설정
    Student splitVal = values[first];
    int saveFirst = first;
    bool onCorrectSide;

    first++;
    do
    {
        onCorrectSide = true;
        //비교 대상을 인덱스가 더 큰쪽으로 이동하며 학생들을 비교함
        while (onCorrectSide) // Move first toward last.
        {
            //이름을 기준으로 현재 인덱스의 학생과 기준점의 학생을 비교한다.
            if (/*-----*/)
                onCorrectSide = false;
            else
            {
                first++;
                onCorrectSide = (first <= last);
            }
        }
        onCorrectSide = (first <= last);
        while (onCorrectSide) // Move last toward first.
        {
            //이름을 기준으로 현재 인덱스의 학생과 기준점의 학생을 비교한다.
            if (/*-----*/)
                onCorrectSide = false;
            else
            {
                last--;
                onCorrectSide = (first <= last);
            }
        }

        if (first < last)
        {
            Swap(values[first], values[last]);
            first++;
            last--;
        }
    } while (first <= last);

    splitPoint = last;
    Swap(values[saveFirst], values[splitPoint]);
}
```

■ QuickSort 예제 소스코드 (QuickSort 함수)

```
void QuickSort(Student values[], int first, int last)
{
    if (first < last)
    {
        int splitPoint;

        Split(values, first, last, splitPoint);
        // values[first]..values[splitPoint-1] <= splitVal
        // values[splitPoint] = splitVal
        // values[splitPoint+1]..values[last] > splitVal

        QuickSort(values, first, splitPoint-1);
        QuickSort(values, splitPoint+1, last);
    }
}
```

■ 테스트 드라이버

```
#include <iostream>
#include "Student.h"
#include "QuickSort.h"
using namespace std;

int main()
{
    Student stu[100];
    stu[0].InitValue(2003200111, "이웅재", 3.0);
    stu[1].InitValue(2004200121, "권오준", 3.2);
    stu[2].InitValue(2005200132, "김진일", 2.7);

    QuickSort(stu, 0, 2);
    Print(cout, stu, 3);
    return 0;
}
```

2. Merge Sort

■ 문 제

- ❖ 학생들의 정보를 담고있는 배열을 Merge Sort 기법을 이용하여 정렬하시오.
 - 정렬의 기준 : 이름, 오름차순 (문자열 비교함수의 결과를 이용하여 정렬함)
- ❖ Merge Sort 함수를 MergeSort.cpp 파일에 구현할 것

■ 참고 사항

- ❖ MergeSort.cpp 파일을 이용하여 구현할 것

■ MergeSort 예제 및 소스 코드 (Merge 함수)

6 5 3 1 8 7 2 4

```
#include "Student.h"

void Merge (Student values[], int leftFirst, int leftLast,
            int rightFirst, int rightLast)
// Post: values[leftFirst]..values[leftLast] and
//        values[rightFirst]..values[rightLast] have been merged.
//        values[leftFirst]..values[rightLast] is now sorted.
{
    int arySize = rightLast-leftFirst+1;
    Student* tempArray= new Student[arySize];
    int index = leftFirst;
    int saveFirst = leftFirst;

    while ((leftFirst <= leftLast) && (rightFirst <= rightLast))
    {
        // 같은 배열 내의 분할된 두개의 영역 내의 학생들의 값을 이들을 기준으로 비교한다
        if (/*-----*/)
        {
            tempArray[index] = values[leftFirst];
            leftFirst++;
        }
        else
        {
            tempArray[index] = values[rightFirst];
            rightFirst++;
        }
        index++;
    }

    while (leftFirst <= leftLast)
    // Copy remaining items from left half.
    {
        tempArray[index] = values[leftFirst];
        leftFirst++;
        index++;
    }

    while (rightFirst <= rightLast)
    // Copy remaining items from right half.
    {
        tempArray[index] = values[rightFirst];
        rightFirst++;
        index++;
    }

    for (index = saveFirst; index <= rightLast; index++)
        values[index] = tempArray[index];
    delete[] tempArray;
}
```

■ MergeSort 소스 코드

```
void MergeSort(Student values[], int first, int last)
// Post: The elements in values are sorted by key.
{
    if (first < last)
    {
        int middle = (first + last) / 2;
        MergeSort(values, first, middle);
        MergeSort(values, middle + 1, last);
        Merge(values, first, middle, middle + 1, last);
    }
}
```

■ 테스트 드라이버

```
#include <iostream>
#include "Student.h"
#include "MergeSort.h"
using namespace std;

int main()
{
    Student stu[100];
    stu[0].InitValue(2003200111, "이응재", 3.0);
    stu[1].InitValue(2004200121, "권오준", 3.2);
    stu[2].InitValue(2005200132, "김진일", 2.7);

    MergeSort(stu, 0, 2);
    Print(cout, stu, 3);
    return 0;
}
```

3. Tag Sort

■ 문 제

- ❖ Student 배열을 직접 sort하는 대신에 Student 레코드를 가리키는 포인터의 배열을 만들어서 포인터의 배열을 정렬하려고 한다. 위에서 구현한 다섯가지 알고리즘을 포인터 배열 정렬을 할 수 있도록 수정하시오.
 - A. QuickSortPointer 함수를 구현하시오.
 - B. MergeSortPointer 함수를 구현하시오.

■ 참고 사항

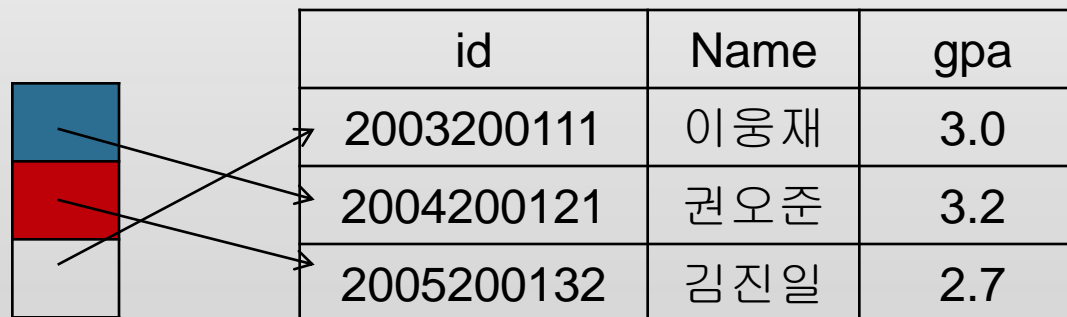
- ❖ 구현할 함수의 파라미터를 포인터 배열로 변경해서 진행할 것
 - 예) `void SelectionSortPointer(Student* values[], int numValues);`

■ 예 제

- ❖ 실제 데이터는 정렬하지 않고 포인터의 배열을 정렬함
- ❖ 정렬된 값은 포인터 배열을 통해서 확인할 수 있음



Student* stu[3]



Student* stu[3]

■ 수정 사항

❖ Swap.h 와 cpp에 다음과 같은 함수를 추가

```
void Swap(Student*& item1, Student*& item2)
```

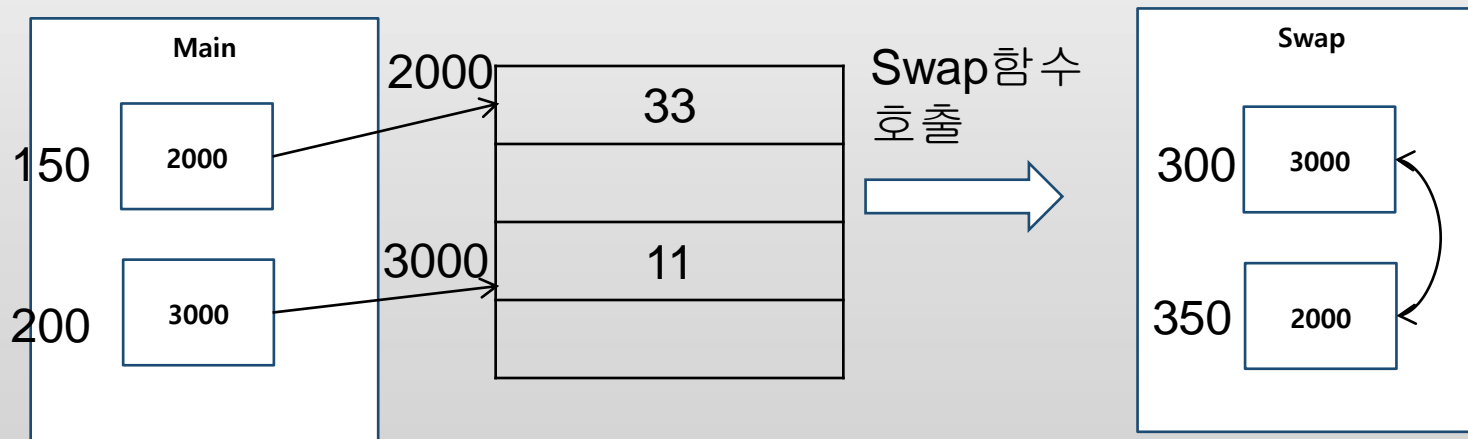
[Student.h 에 추가]

```
void Swap(Student*& item1, Student*& item2)
// Post: Contents of item1 and item2 have been swapped.
{
    Student* templtem;

    templtem = item1;
    item1 = item2;
    item2 = templtem;
}
```

[Swap.cpp 에 추가]

■ Swap함수의 파라미터를 Call by value 설정할때의 문제점



Swap함수가 리턴될때
Call by value 이기때문에
포인터가 저장하고있는 주소값이 초기화됨

■ 테스트 드라이버

```
int main()  
{  
  
    Student stu[3];  
    stu[0].InitValue(2003200111, "이웅재", 3.0);  
    stu[1].InitValue(2004200121, "권오준", 3.2);  
    stu[2].InitValue(2005200132, "김진일", 2.7);  
    Student* stuPtrs[3];  
  
    for(int k=0; k<3; k++)  
        stuPtrs[k] = &stu[k];  
    SelectionSortPointer(stuPtrs, 3);  
    PrintByPointer(cout, stuPtrs, 3);  
    return 0;  
}
```

4. Hashing

■ 문제

- ❖ Student 배열이 주어졌을 때, 이 것을 Hash Table에 넣고, 이름이 주어졌을 때 Hash Table에서 검색해서 출력하는 프로그램을 작성하시오.
- ❖ HashTable class를 구현하시오

■ 예제

입력 배열		
id	Name	gpa
2003200111	이웅재	3.0
2004200121	권오준	3.2
2005200132	김진일	2.7

Hashing →

가정:

Hash(이웅재) = 73

Hash(권오준) = 29

Hash(김진일) = 3

Hash Table

[0]	
...	
[3]	... 김진일...
...	
[29]	... 권오준...
...	
[73]	... 이웅재...
...	
[99]	

■ Class HashTable

```
#include "Student.h"

const int MAX_ITEMS = 20000;

template <class ItemType>
class HashTable {
    public:
        HashTable() {}
        HashTable(ItemType emptyKey);
        int Hash(char* key) const;
        void RetrieveItem(ItemType& item, bool& found);
        void InsertItem(ItemType item);

    private:
        ItemType info[MAX_ITEMS];
        ItemType emptyItem;
        int length;
};
```

■ Hash()

- Hash() 멤버 함수는 키 값을 이용하여 hash address를 리턴함

```
int HashTable<ItemType>::Hash(char *key) const
{
    return (getIntFromString(key) % MAX_ITEMS);
}
```

- ❖ 문자열을 숫자로 구하는 함수는 아래 코드 참고

```
// returns an integer hash value for key for a 15 bit
int getIntFromString (char *key)
{
    int sum = 0;
    int len = strlen(key);
    if (len % 2 == 1) len++; // make len even
    for (int j = 0; j < len; j+=2)
        sum = (sum + 100 * key[j] + key[j+1]) % 19937;
    return sum;
}
```

■ InsertItem()

```
template<class ItemType>
void HashTable<ItemType>::InsertItem(ItemType item)
{
    int location;
    location = Hash(item.getKey());
    while (info[location] != emptyItem)
        location = (location + 1) % MAX_ITEMS;
    info[location] = item;
    length++;
}
```

■ Student 클래스에 Key() 멤버함수 추가

```
char* Student::Key()
{
    return name;
}
```

■ RetrievalItem()

```
template<class ItemType>
void HashTable<ItemType>::RetrieveItem(ItemType &item, bool &
found)
{
    int location;
    int startLoc;
    bool moreToSearch = true;

    startLoc = _____; // hash addr를 구한다
    location = startLoc;
    do {
        if (info[location] == item || info[location] == emptyItem)
            moreToSearch = false;
        else
            location = _____; // linear probing
    } while (_____);
    found = (info[location] == item);
    if (found)
        _____; // copy item
}
```

Student 클래스 operator== 오버로딩
Name이 같으면 true

emptyItem

Student 클래스에 EmptyKey() 멤버함수 추가
HashTable 생성자에서 emptyKey 설정

■ RetrieveItem() : Alternative Version

- ❖ HashTable에서 info를 ItemType info[MAX_ITEMS] (레코드 배열)가 아닌 ItemType *info[MAX_ITEMS] (포인터 배열)로 선언
- ❖ Insert/delete할 때 동적 메모리 할당으로 처리
- ❖ RetrieveItem은 다음과 같이 구현할 수 있음

```
template<class ItemType>
void HashTable<ItemType>::RetrieveItem(ItemType &item, bool &
found)
{
    ...
    do {
        if ( (strcmp(info[location]->Key(), item->Key()) == 0) ||
            (info[location] == NULL) )
            moreToSearch = false;
        else
            location = _____; // linear probing
    } while ( _____ );
    ...
}
```


■ 테스트 드라이버

```
#include "HashTable.h"
#include "Student.h"

int main()
{
    Student stu[100];

    // stu 배열 초기화
    ...

    HashTable<Student> ht;
    // stu 배열에 있는 아이템들을 HashTable에 삽입
    ...

    // 학생 이름 값을 키보드로 입력받는다
    ...

    // HashTable에서 학생 이름을 찾아 출력한다
    ht.RetrieveItem(...);
    cout << item;
    return 0;
}
```