**자료구조**

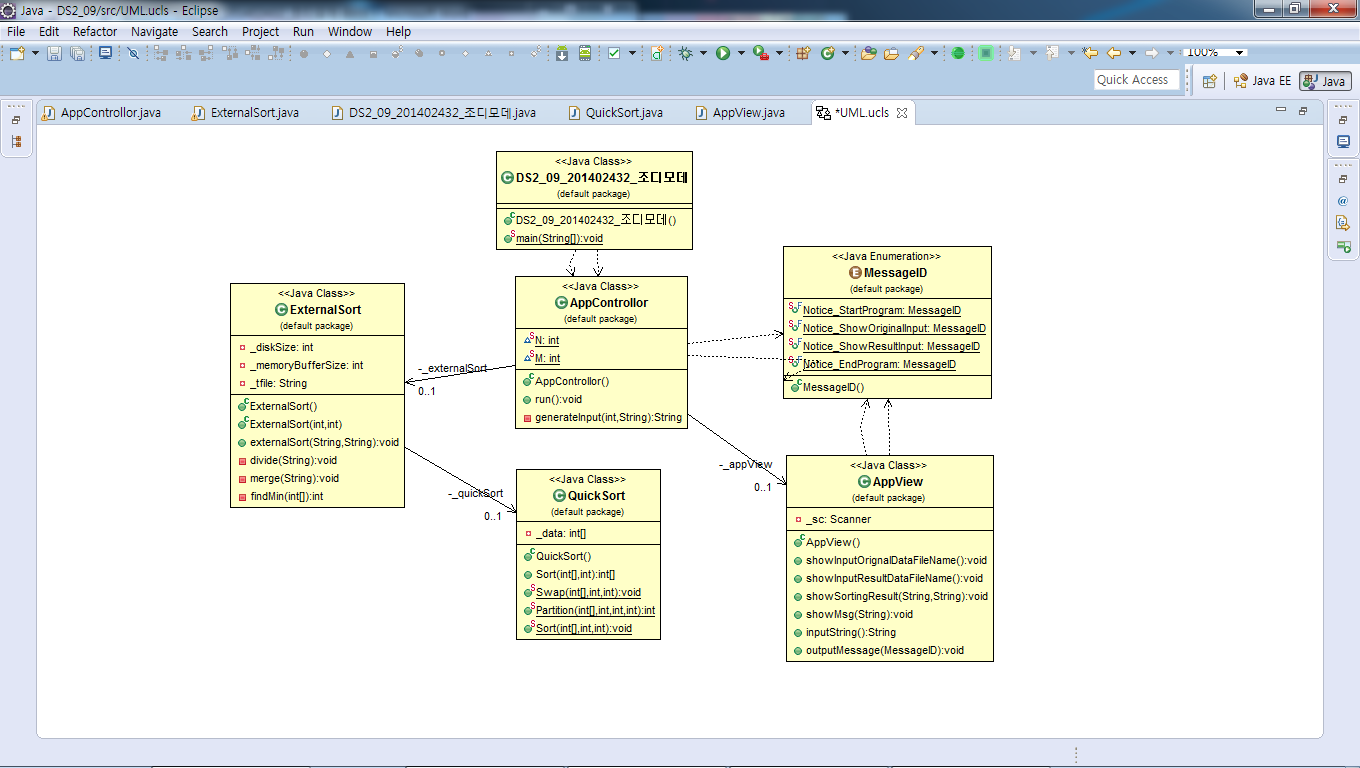
**실습 보고서**

[제 09주] 외부 정렬

제출일 : 2015.11.10

201402432 / 조디모데

1.프로그램설명서

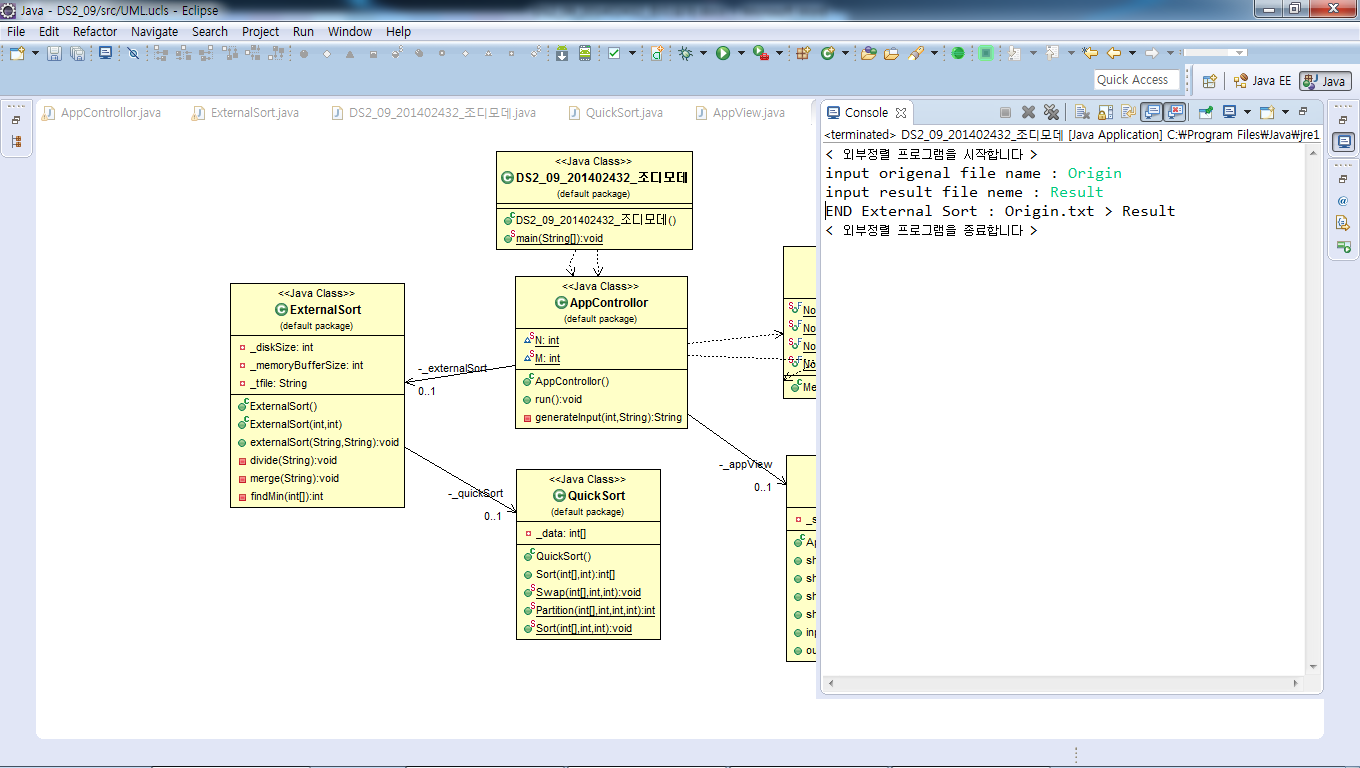


**자료 구조 & 알고리즘 : Array, QuickSort, 외부 정렬, 분할 정렬**

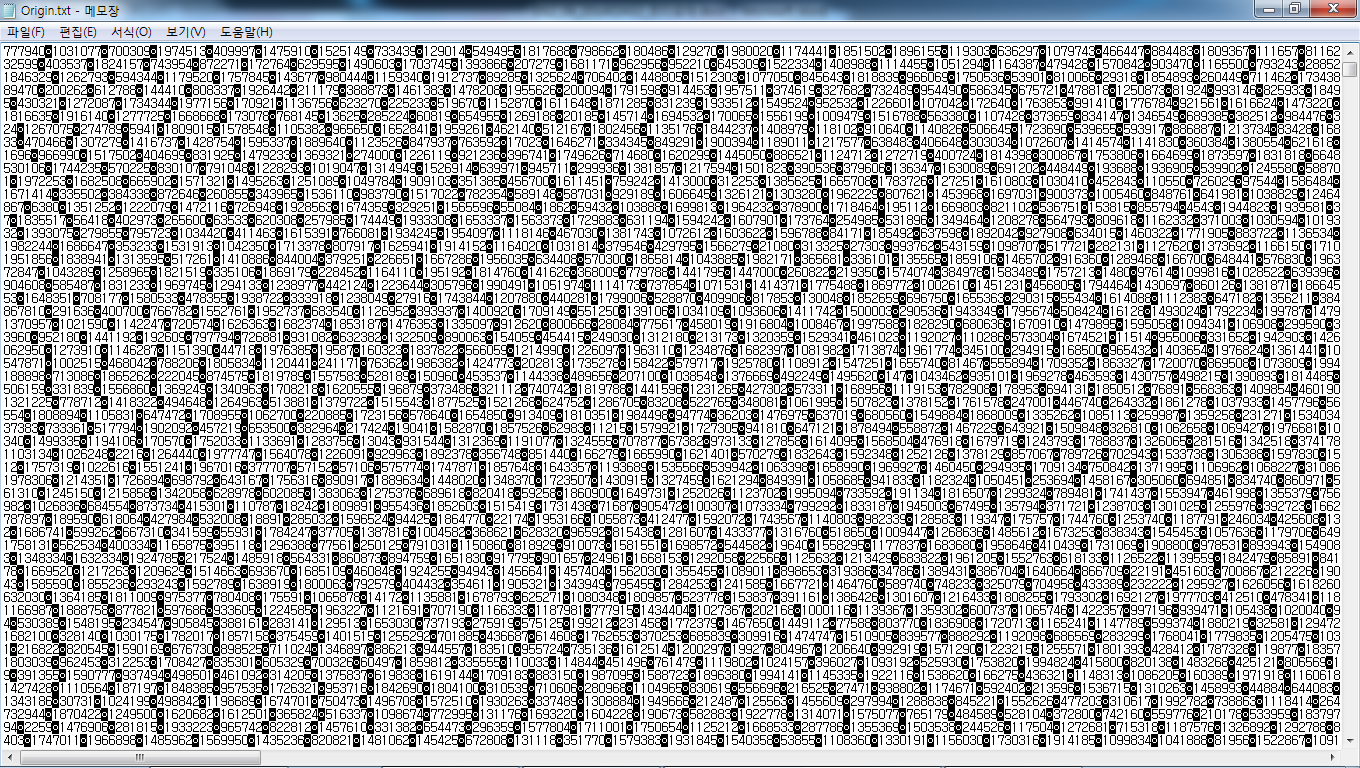
2.실행 결과 분석

1.입력과출력

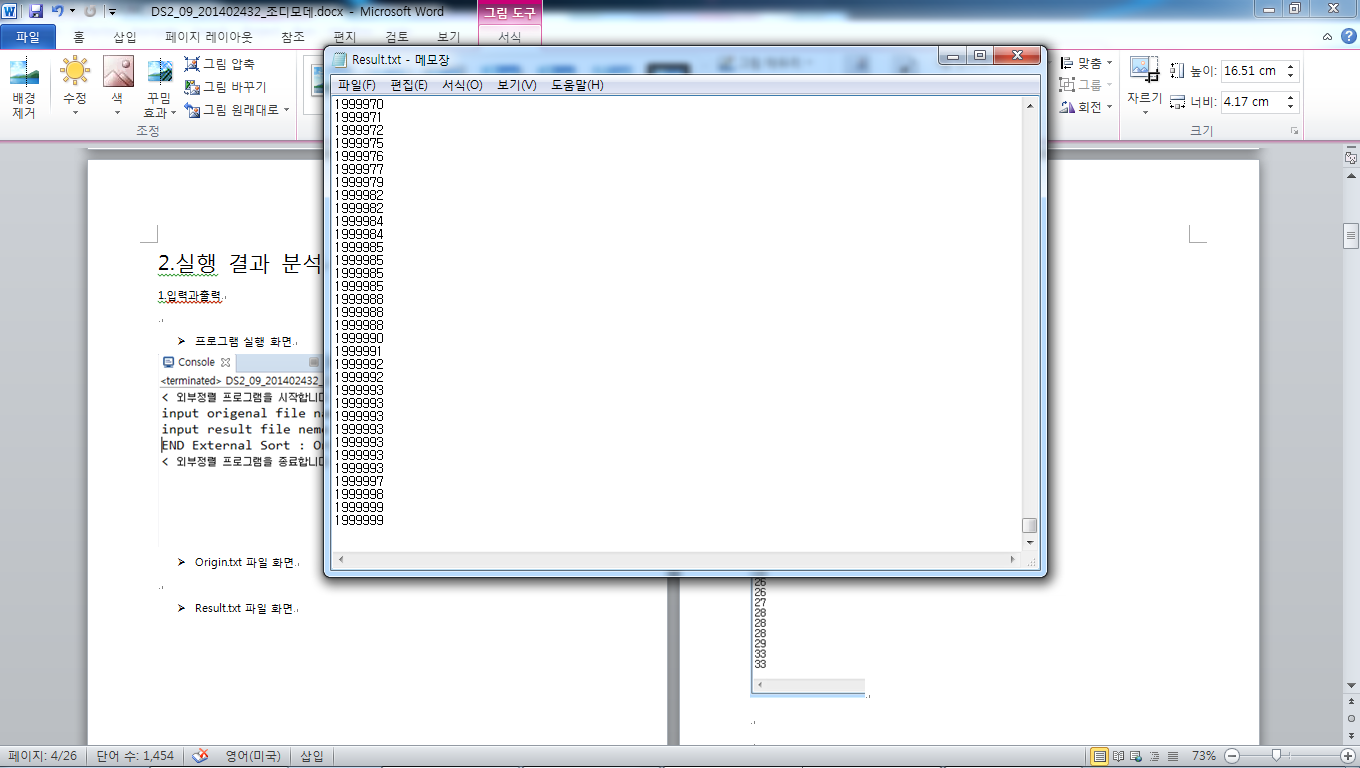
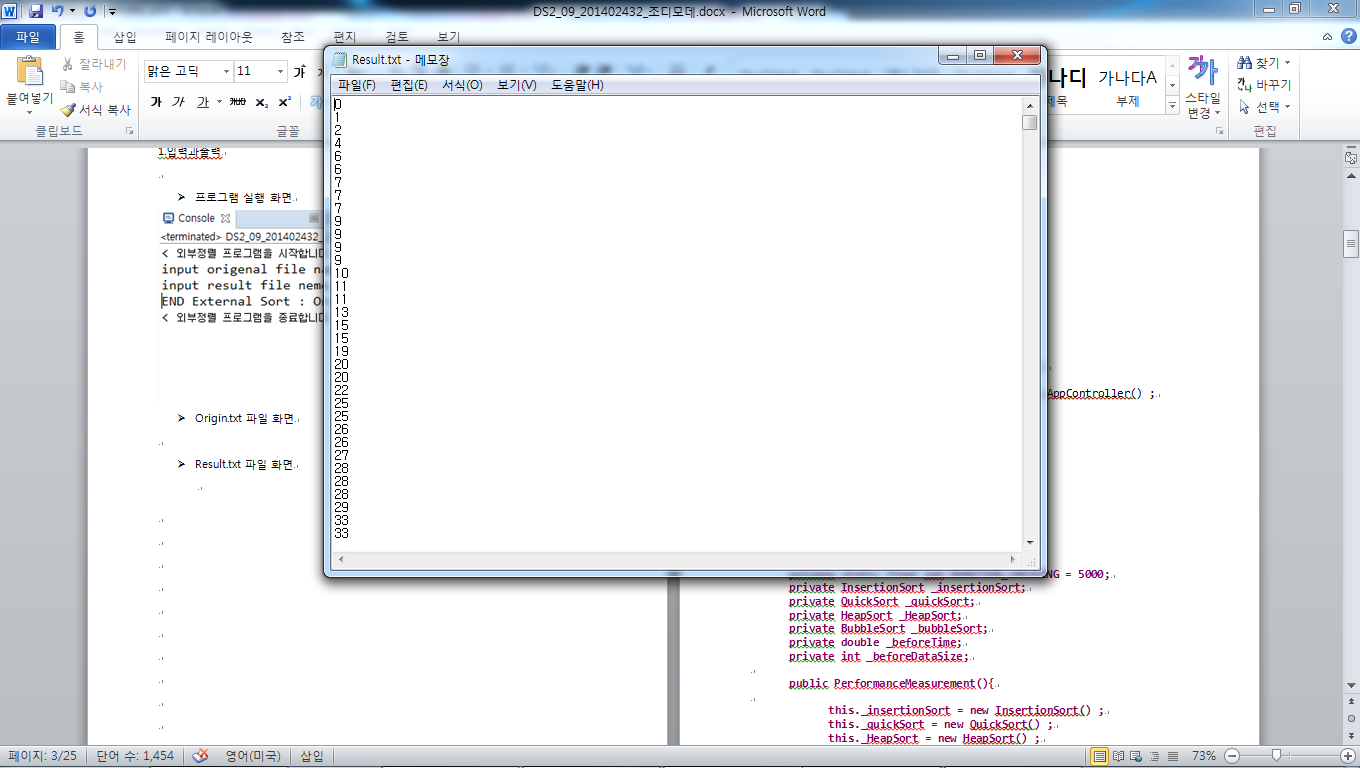
* 프로그램 실행 화면



* Origin.txt 파일 화면



* Result.txt 파일 화면



3.소스 코드

<main>

**public** **class** DS2\_09\_201402432\_조디모데 {

**public** **static** **void** main(String[] args) {

AppControllor app = **new** AppControllor() ;

app.run() ;

}

}

<AppControllor>

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.Random;

public class AppControllor {

static int N = 2000000; // size of the file in disk

static int M = 100000; // max items the memory buffer can hold

private AppView \_appView;

private ExternalSort \_externalSort;

public AppControllor(){

}

public void run(){

this.\_appView= new AppView();

this.\_externalSort= new ExternalSort(N, M);

this.\_appView.outputMessage(MessageID.Notice\_StartProgram);

this.\_appView.outputMessage(MessageID.Notice\_ShowOriginalInput);

String orignalDataFileName= this.generateInput(N, this.\_appView.inputString());

this.\_appView.outputMessage(MessageID.Notice\_ShowResultInput);

String resultDataFileName= this.\_appView.inputString();

\_externalSort.externalSort(orignalDataFileName,resultDataFileName);

this.\_appView.showSortingResult(orignalDataFileName, resultDataFileName);

this.\_appView.outputMessage(MessageID.Notice\_EndProgram);

}

private String generateInput(int n, String inputString){

Random r = new Random() ;

FileWriter fileWriter ;

try{

fileWriter = new FileWriter (inputString+".txt") ;

for(int i=0 ; i<n ; i++)

fileWriter.write(Integer.toString( r.nextInt(2000000) ) +"\n" ) ;

}catch(Exception e){

System.out.println(e.getMessage()) ;

}

return inputString+".txt" ;

}

}

<AppView>

**import** java.util.\* ;

**public** **class** AppView {

**private** Scanner \_sc ;

**public** AppView(){

**this**.\_sc = **new** Scanner(System.***in***) ;

}

**public** **void** showInputOrignalDataFileName(){

}

**public** **void** showInputResultDataFileName(){

}

**public** **void** showSortingResult(String orignalDataFileName, String resultDataFileName){

System.***out***.println("END External Sort : " + orignalDataFileName+ " > " + resultDataFileName);

}

**public** **void** showMsg(String aString){

System.***out***.println(aString);

}

**public** String inputString(){

**return** **this**.\_sc.nextLine() ;

}

**public** **void** outputMessage(MessageID noticeStartprogram) {

**switch**(noticeStartprogram){

**case** ***Notice\_StartProgram*** :

System.***out***.println("< 외부정렬 프로그램을 시작합니다 >");

**break** ;

**case** ***Notice\_ShowOriginalInput*** :

System.***out***.print("input origenal file name : ");

**break** ;

**case** ***Notice\_ShowResultInput*** :

System.***out***.print("input result file neme : ");

**break** ;

**case** ***Notice\_EndProgram*** :

System.***out***.println("< 외부정렬 프로그램을 종료합니다 >");

**break** ;

**default** :

}

}

}

<ExternalSort>

**import** java.io.BufferedReader;

**import** java.io.FileNotFoundException;

**import** java.io.FileReader;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.util.Arrays;

**import** java.util.Collections;

**import** java.util.Stack;

**public** **class** ExternalSort {

**private** **int** \_diskSize;

**private** **int** \_memoryBufferSize;

**private** String \_tfile = "temp";

**private** QuickSort \_quickSort ;

**public** ExternalSort(){

}

**public** ExternalSort(**int** givenDiskSize, **int** givenMemorySize) {

**this**.\_diskSize = givenDiskSize ;

**this**.\_memoryBufferSize = givenMemorySize ;

}

**public** **void** externalSort(String dataName, String resultName){

// Divide 와 merge작업 수행

**this**.divide(dataName);

**this**.merge(resultName);

}

**private** **void** divide(String dataName){

// 정렬하려는파일을원하는개수만큼잘라서정렬하여저장

**int**[] buffer = **new** **int**[**this**.\_memoryBufferSize < **this**.\_diskSize ? **this**.\_memoryBufferSize : **this**.\_diskSize] ;

**this**.\_quickSort = **new** QuickSort() ;

**try**{

FileReader fileReader = **new** FileReader(dataName) ;

BufferedReader bufferedReader = **new** BufferedReader(fileReader) ;

**int** slices = (**int**)Math.*ceil*((**double**)**this**.\_diskSize / **this**.\_memoryBufferSize) ;

**int** i, j ;

i = j = 0 ;

**for**(i = 0 ; i< slices ; i++){

String str ;

**for**(j = 0 ; j < (**this**.\_memoryBufferSize < **this**.\_diskSize ? **this**.\_memoryBufferSize : **this**.\_diskSize) ; j++){

str = bufferedReader.readLine() ;

**if**(str==**null**)

**continue** ;

buffer[j] = Integer.*parseInt*(str) ;

}

buffer = **this**.\_quickSort.Sort(buffer, buffer.length);

FileWriter fileWriter = **new** FileWriter (**this**.\_tfile + Integer.*toString*(i) + ".txt") ;

PrintWriter printWriter = **new** PrintWriter(fileWriter) ;

**for**(**int** k = 0 ; k < buffer.length ; k++)

printWriter.println(buffer[k]) ;

fileWriter.close();

printWriter.close();

}

}**catch**(Exception e){

e.printStackTrace() ;

}

}

**private** **void** merge(String resultName){

// 정렬되어있는파일들을모아서하나의출력파일을만들어냄

**int** slices = (**int**)Math.*ceil*((**double**)**this**.\_diskSize / **this**.\_memoryBufferSize) ;

**int**[] top = **new** **int**[slices] ;

FileReader fileReader ;

BufferedReader[] bufferedReader = **new** BufferedReader[slices] ;

**try**{

FileWriter fileWriter = **new** FileWriter ("Result.txt") ;

@SuppressWarnings({ "unused", "resource" })

PrintWriter printWriter = **new** PrintWriter(fileWriter) ;

**for**(**int** i=0 ; i<slices ; i++){

fileReader = **new** FileReader("temp"+Integer.*toString*(i)+".txt") ;

bufferedReader[i] = **new** BufferedReader(fileReader) ;

}

String str ;

**int** min = 0 ;

**for**(**int** i=0 ; i<slices ; i++){

str = bufferedReader[i].readLine() ;

**if**(str==**null**)

**continue** ;

top[i] = Integer.*parseInt*(str) ;

**if**(top[min]!=Math.*min*(top[i], top[min]))

min=i ;

}

**while**(**true**){

printWriter.println( Integer.*toString*(top[min]) ) ;

str = bufferedReader[min].readLine() ;

**if**(str!=**null**){

top[min] = Integer.*parseInt*(str) ;

min = **this**.findMin(top) ;

}

**else**{

top[min]=Integer.***MAX\_VALUE*** ;

min = **this**.findMin(top) ;

}

**boolean** end = **true** ;

**for**(**int** x : top)

**if**(x!=Integer.***MAX\_VALUE***)

end = **false** ;

**if**(end){

printWriter.close() ;

fileWriter.close() ;

**break** ;

}

}

}**catch**(Exception e){

e.printStackTrace() ;

}

}

**private** **int** findMin(**int**[] num){

**int** min = 0 ;

**for**(**int** i=1 ; i<num.length ; i++){

**if**(num[min]>num[i])

min = i ;

}

**return** min ;

}

}

<MessageID>

**public** **enum** MessageID {

***Notice\_StartProgram***,

***Notice\_ShowOriginalInput***,

***Notice\_ShowResultInput***,

***Notice\_EndProgram***

}

<QuickSort>

**public** **class** QuickSort {

**private** **int**[] \_data ;

**public** **int**[] Sort(**int**[] data, **int** length){

**this**.\_data = data.clone() ;

QuickSort.*Sort*(**this**.\_data, 0, length-1);

**return** **this**.\_data ;

}

**public** **static** **void** Swap(**int**[] list, **int** idx1, **int** idx2) {

**int** swapTmp = list[idx1];

list[idx1] = list[idx2];

list[idx2] = swapTmp;

}

**public** **static** **int** Partition(**int**[] list, **int** left, **int** right, **int** pivot\_idx) {

**int** pivot = list[pivot\_idx];

*Swap*(list, pivot\_idx, right); //Move to end

**int** split\_idx = left;

**for**(**int** i=left ; i<right ; i++) {

**if**(list[i] <= pivot) {

*Swap*(list, split\_idx, i);

++split\_idx;

}

}

*Swap*(list, right, split\_idx); //Move to split index

**return** split\_idx;

}

**public** **static** **void** Sort(**int**[] list, **int** left, **int** right) {

**if**(right > left) {

**int** pivot\_idx = *Partition*(list, left, right, left);

*Sort*(list, left, pivot\_idx - 1);

*Sort*(list, pivot\_idx + 1, right);

}

}

}