**CSCI 334**

**Aashish Gopali**

**Source Documentation**

Main. Java

package pkg334project;

// imported java libraries

import java.util.\*;

import java.io.\*;

// this is the main class

public class Main {

public static void main(String[] args) throws Exception {

// if we need the input file

//Scanner inputFile = new Scanner(System.in);

//Scanner option = new Scanner(System.in);

//System.out.print("Please give the name of the file to be created and sorted: ");

//String fileName = inputFile.next();

//we create a file namae random.txt and pass it as file type

File out = new File("random.txt");

FileWriter fw = null;

// Size of data

long n = 10000;

// creating random numbers and writing on random.txt file with

// help of file writer

fw = new FileWriter(out);

BufferedWriter writer = new BufferedWriter(fw);

int line;

Random random = new Random();

while (n > 0) {

line = random.nextInt(1000);

writer.write(line + "\n");

n--;

}

writer.close();

File file = new File("random.txt");

// arraylist used to read the value

ArrayList<Integer> values = new ArrayList<Integer>();

// reading the value till the end of file.

try(Scanner s = new Scanner(file))

{

while(s.hasNext())

{

values.add(s.nextInt());

}

}

int j = 0;

System.out.println("\n\n\n\n");

System.out.println("Before QuickSorting");

displayArrayList(values);

quicksort(values);

values.removeAll(values);

// reading the value till the end of file.

try(Scanner s = new Scanner(file))

{

while(s.hasNext())

{

values.add(s.nextInt());

}

}

System.out.println("\n\n\n\n");

System.out.println("Before HeapSorting");

displayArrayList(values);

heapsort(values);

//removing the data in the values arraylist to use it again.

values.clear();

// reading the value till the end of file.

try(Scanner s = new Scanner(file))

{

while(s.hasNext())

{

values.add(s.nextInt());

}

}

System.out.println("\n\n\n\n");

//displaying the arraylist which is not sorted

//System.out.println("Before RadixSorting");

//displayArrayList(values);

radixsort(values);

//removing the data in the values arraylist to use it again.

values.clear();

// reading the value till the end of file.

try(Scanner s = new Scanner(file))

{

while(s.hasNext())

{

values.add(s.nextInt());

}

}

System.out.println("\n\n\n\n");

//displaying the arraylist which is not sorted

//System.out.println("Before MergeSorting");

//displayArrayList(values);

mergesort(values);

//removing the data in the values arraylist to use it again.

values.clear();

// reading the value till the end of file.

try(Scanner s = new Scanner(file))

{

while(s.hasNext())

{

values.add(s.nextInt());

}

}

System.out.println("\n\n\n\n");

System.out.println("Before BubbleSorting");

//displaying the arraylist whcih is not sorted

//displayArrayList(values);

bubblesort(values);

//removing the data in the values arraylist to use it again.

values.clear();

}

// this functions displays arraylist

static void displayArrayList(ArrayList<Integer> values)

{

int j =0;

for(int i = 0; i < values.size()-1;i++)

{

if(j==100)

{

System.out.println("\n");

j = 0;

}

System.out.print(values.get(i) + " ");

j++;

}

}

// this functions displays Array.

static void displayArray(int[] array)

{

int j =0;

for(int i = 0 ; i < array.length; i++)

{

if(j == 100)

{

System.out.println("\n");

j = 0;

}

System.out.print(array[i] + " ");

j++;

}

}

// function for quicksorting

static void quicksort(ArrayList<Integer> values)

{

long startTime,endTime,timeElapsed;

QuickSort qsu = new QuickSort(values);

startTime = System.nanoTime();

qsu.startQuickStart(0, values.size()-1);

endTime = System.nanoTime();

ArrayList<Integer> sortedArray = qsu.getSortedArray();

timeElapsed = endTime - startTime;

int j = 0;

System.out.println("\n\n\n\n");

System.out.println("After QuickSorting");

displayArrayList(sortedArray);

System.out.println("");

System.out.println("Execution time in milliseconds for QuickSort: " +

timeElapsed / 1000000 + " ms");

sortedArray.clear();

}

// function for heapsorting

static void heapsort(ArrayList<Integer> values)

{

long startTime,endTime,timeElapsed;

HeapSort hs = new HeapSort();

startTime = System.nanoTime();

ArrayList<Integer> sortedArray = hs.heapSort(values);

endTime = System.nanoTime();

System.out.println("\n\n\n\n");

System.out.println("After HeapSorting");

timeElapsed = endTime - startTime;

displayArrayList(sortedArray);

System.out.println("\nExecution time in milliseconds for HeapSort: " +

timeElapsed / 1000000 + " ms");

sortedArray.clear();

}

// function for radixsorting

static void radixsort(ArrayList<Integer> values)

{

long startTime,endTime,timeElapsed;

RadixSort rx = new RadixSort();

startTime = System.nanoTime();

int[] sortedArray = rx.radixSort(values);

endTime = System.nanoTime();

System.out.println("\n\n\n\n");

System.out.println("After RadixSorting");

displayArray(sortedArray);

timeElapsed = endTime - startTime;

System.out.println("\nExecution time in milliseconds for RadixSort: " +

timeElapsed / 1000000 + " ms");

sortedArray = null;

}

// function for bubblesorting

static void mergesort(ArrayList<Integer> values)

{

long startTime,endTime,timeElapsed;

MergeSort ms = new MergeSort();

startTime = System.nanoTime();

ArrayList<Integer> sortedArray = ms.mergeSort(values);

endTime = System.nanoTime();

System.out.println("\n\n\n\n");

System.out.println("After MergeSorting");

timeElapsed = endTime - startTime;

displayArrayList(sortedArray);

System.out.println("\nExecution time in milliseconds for MergeSort: " +

timeElapsed / 1000000 + " ms");

sortedArray.clear();

}

// function for bubblesorting

static void bubblesort(ArrayList<Integer> values)

{

long startTime,endTime,timeElapsed;

BubbleSort bs = new BubbleSort();

startTime = System.nanoTime();

ArrayList<Integer> sortedArray = bs.bubbleSort(values);

endTime = System.nanoTime();

System.out.println("\n\n\n\n");

System.out.println("After BubbleSorting");

timeElapsed = endTime - startTime;

displayArrayList(sortedArray);

System.out.println("\nExecution time in milliseconds for BubbleSort: " +

timeElapsed / 1000000 + " ms");

sortedArray.clear();

}

}

MergeSort.java

package pkg334project;

import java.util.ArrayList;

public class MergeSort{

//megesort constructor empty.

public MergeSort()

{

}

// function that takes care of mergesorting and return sorted arraylist.

public ArrayList<Integer> mergeSort(ArrayList<Integer> unsortedList)

{

if(unsortedList.size() <= 1)

{

return unsortedList;

}

ArrayList<Integer> sortedList = new ArrayList<Integer>();

ArrayList<Integer> left = new ArrayList<Integer>();

ArrayList<Integer> right = new ArrayList<Integer>();

int middle = unsortedList.size()/2;

//Splits the array into unsortedList size lists of size one

for(int i = 0; i < unsortedList.size(); i++)

{

if(i < middle)

{

left.add(unsortedList.get(i));

}

else

{

right.add(unsortedList.get(i));

}

}

left = mergeSort(left);

right = mergeSort(right);

//combines the lists

sortedList = merge(left, right);

return sortedList;

}

// merges two arraylist together and return merged arraylist

public ArrayList<Integer> merge(ArrayList<Integer> left, ArrayList<Integer> right)

{

ArrayList<Integer> mergedList = new ArrayList<Integer>();

while(left.size() > 0 || right.size() > 0)

{

if(left.size() > 0 && right.size() > 0)

{

if(left.get(0) < right.get(0))

{

mergedList.add(left.get(0));

left.remove(0);

}

else

{

mergedList.add(right.get(0));

right.remove(0);

}

}

else if(left.size() > 0)

{

mergedList.add(left.get(0));

left.remove(0);

}

else if(right.size() > 0)

{

mergedList.add(right.get(0));

right.remove(0);

}

}

return mergedList;

}

}

Bubblesort.java

package pkg334project;

import java.util.\*;

//Class for bubblesort

public class BubbleSort {

// empty constructor which does nothing

public BubbleSort() {

}

// function to return arraylist when bubble sorted

public ArrayList<Integer> bubbleSort(ArrayList<Integer> unsortedList)

{

ArrayList<Integer> sortedList = new ArrayList<Integer>();

boolean swapped = true;

//swapp when needed

while(swapped)

{

swapped = false;

for(int i = 1; i < unsortedList.size(); i++)

{

if(unsortedList.get(i) < unsortedList.get(i - 1))

{

//Swapping neighboring data points

int holder = unsortedList.get(i);

unsortedList.set(i, unsortedList.get(i - 1));

unsortedList.set(i - 1, holder);

swapped = true;

}

}

}

sortedList = unsortedList;

return sortedList;

}

}

QuickSort.java

package pkg334project;

import java.util.\*;

public class QuickSort {

//arraylist to work on

private static ArrayList<Integer> inputArray = new ArrayList<Integer>();

//constructor to aassign inputArray.

public QuickSort(ArrayList<Integer> inputArray){

QuickSort.inputArray = inputArray;

}

// recursive function to partition and sort.

public void startQuickStart(int start,int end){

int q;

if(start<end){

q = partition(start, end);

startQuickStart(start, q);

startQuickStart(q+1, end);

}

}

// function to return sorted arraylist

public ArrayList<Integer> getSortedArray(){

return QuickSort.inputArray;

}

// partitioning the arraylist.

int partition(int start,int end){

int init = start;

int length = end;

Random r = new Random();

int pivotIndex = nextIntInRange(start,end,r);

int pivot = inputArray.get(pivotIndex);

while(true){

while(inputArray.get(length)>pivot && length>start){

length--;

}

while(inputArray.get(init)<pivot && init<end){

init++;

}

if(init<length){

int temp;

temp = inputArray.get(init);

inputArray.set(init,inputArray.get(length));

inputArray.set(length,temp);

length--;

init++;

}

else

{

return length;

}

}

}

// Below method is to just find random integer from given range

static int nextIntInRange(int min, int max, Random rng) {

if (min > max) {

throw new IllegalArgumentException("Cannot draw random int from invalid range [" + min + ", " + max + "].");

}

int diff = max - min;

if (diff >= 0 && diff != Integer.MAX\_VALUE) {

return (min + rng.nextInt(diff + 1));

}

int i;

do {

i = rng.nextInt();

} while (i < min || i > max);

return i;

}

}

Heapsort.java

package pkg334project;

import java.io.IOException;

import java.util.ArrayList;

//class for heapsort

public class HeapSort{

//heapsort empty constructor to assign

HeapSort()

{

}

// arraylist returning function when sorted

public ArrayList<Integer> heapSort(ArrayList<Integer> unsortedList)

{

int count = unsortedList.size();

heapify(unsortedList, count);

int end = count-1;

while(end > 0)

{

swap(unsortedList, end, 0);

end = end - 1;

siftDown(unsortedList, 0, end);

}

return unsortedList;

}

// heapiftying or changing to heap.

public void heapify(ArrayList<Integer> unsortedList, int count)

{

int start = count/2 - 1;

while(start >= 0)

{

siftDown(unsortedList, start, count - 1);

start -= 1;

}

}

// shifting the value when value is greater or less

public void siftDown(ArrayList<Integer> unsortedList, int start, int end)

{

int root = start;

while(root\*2+1 <= end)

{

int child = root\*2+1;

int swap = root;

if(unsortedList.get(swap) < unsortedList.get(child))

{

swap = child;

}

if(child+1 <= end && unsortedList.get(swap) < unsortedList.get(child+1))

{

swap = child+1;

}

if(swap != root)

{

swap(unsortedList, root, swap);

root = swap;

}

else

{

return;

}

}

}

// swapping the heapified value function

public void swap(ArrayList<Integer> unsortedList, int swapOne, int swapTwo)

{

int holder = unsortedList.get(swapOne);

unsortedList.set(swapOne, unsortedList.get(swapTwo));

unsortedList.set(swapTwo, holder);

}

}

Radixsort.java

package pkg334project;

import java.io.IOException;

import java.util.ArrayList;

// Class to work on radixsort

public class RadixSort{

// empty constructor for radix sort

public RadixSort() {

}

// returning radixsorted array which takes unsortedList as an parameter.

public int[] radixSort(ArrayList<Integer> unsortedList)

{

int largestNumber = 0;

int index = 0;

//Create storage unit for holding smaller arrays (size = digits of largest number)

ArrayList<ArrayList<Integer>> storageUnit = new ArrayList<ArrayList<Integer>>();

int[] sortedList = new int[unsortedList.size()];

//move everything into sortedList

for(int i = 0; i < unsortedList.size(); i++)

{

sortedList[i] = unsortedList.get(i);

}

//initialize all of the arraylists

for(int i = 0; i < 10; i++)

{

storageUnit.add(new ArrayList<Integer>());

}

//find largest number in set

for(int i = 0; i < unsortedList.size(); i++)

{

if(unsortedList.get(i) > largestNumber)

{

largestNumber = unsortedList.get(i);

}

}

//run through list, adding the numbers to the buckets based on its key

for(int i = 0; largestNumber != 0; largestNumber = largestNumber/10, i++)

{

for(int j = 0; j < sortedList.length; j++)

{

int key = (int) (sortedList[j]/Math.pow(10, i))%10;

storageUnit.get(key).add(sortedList[j]);

}

//the list is then put back together by dumping the buckets from 0 - 9 into the list

for(int k = 0; k < storageUnit.size(); k++)

{

for(int l = 0; l < storageUnit.get(k).size(); l++)

{

sortedList[index] = storageUnit.get(k).get(l);

index++;

}

storageUnit.get(k).clear();

}

index = 0;

}

return sortedList;

}

}