*Beijing-Dublin International College*

# Data Structures Algs I Project 1

**Author: Haoran Yan**

**BJUT:** 20372119

**UCD:** 20205723

**Class:** SE-1

November 1, 2021

# Data structures & Algs project1 report

## Contents

# Project - A

## 1. The code and logic of the program

In this project-A have two class and one class is Pa which have six static methods:

1. `selectionSort`
2. `selectionSort2D`
3. `changerTwoElement`
4. `changeTwoArray`
5. `insertionSort`
6. `insertionSort2D`

And other class is the test class which named `testToProjectA`.

## 2. Selection Sort

**Algorithm**: for a unsorted array, find the maximum element (considering descending order) and put it at the end.

### 2.1 The methods of selection sort

#### 2.1.1 Method 1 (`selectionSort`)

```
public static int[] selectionSort(int[] test){
    // store test in sortedArray
    int[] sortedArray = test.clone();
    // outer loop
    // The variable times represents the number of numbers that
have been arranged
    int times = 0;
    for(int i = 0; i < test.length; i++){
        int theIndex = 0;
        int largest = sortedArray[0];
        // inner loop
        for(int j = 0; j < test.length-times; j++){
            if(largest < sortedArray[j]){
```

```
                largest = sortedArray[j];
                // theIndex represents the index of the largest
number
                theIndex = j;
            }
        }
        // Find the maximum value and swap it with the last digit
of the unaligned portion
        int exchangeElement = sortedArray[test.length-times-1];
        sortedArray[test.length - times-1] = largest;
        sortedArray[theIndex] = exchangeElement;
        times++;
    }
    return sortedArray;
}
```

In this method.

1. First, the input test is stored in `sortedArray`.
2. Then there is a **nested loop** where the **outer loop** loops through the array to **put each number in place**, and the **inner loop** loops through the unsorted portion to **find the maximum value** in it.
3. Finally, return `sortedArray` of the array that **has been sorted**.

### 2.1.2 Method 2 (`selectionSort2D`)

The second test case is different from a normal 2D array, requiring that only the **first digit** be used as the key to sort.

`int[][] testB = new int[][]{{71,2},{64,8},{31,56},{98,1},{3,6},{59,837},{49,58},{61,8}};`

So I made some changes based on the first method A.

```
public static int[][] selectionSort2D(int[][] test){
    // store test in sortedArray
    int[][] sortedArray = test.clone();
    // The variable times represents the number of array that have
been arranged
    int times = 0;
    // out loop
    for(int i = 0; i < test.length; i++){
```

```
        int theIndex = 0;
        int largest = sortedArray[0][0];
        // inner loop
        for(int j = 0; j < test.length-times; j++){
            if(largest < sortedArray[j][0]){
                largest = sortedArray[j][0];
                // theIndex represents the index of the largest
number
                theIndex = j;
            }
        }
        // Find the maximum key value and swap it with the last
array of the unaligned portion
        int[] exchangeElement = sortedArray[test.length-times-1];
        sortedArray[test.length - times-1] = sortedArray[theIndex];
        sortedArray[theIndex] = exchangeElement;
        times++;
        }
    return sortedArray;
}
```

In this method:

1. First, the input test is stored in `sortedArray`.
2. Then there is a **nested loop** where the **outer loop** loops through the 2D array to **put each array in place**, and the **inner loop** loops through the unsorted portion to **find the maximum key value** in it.
3. Finally, return `sortedArray` of the array that **has been sorted**.

### 2.1.3 Method 3 (`changerTwoElement`) & Method 4 (`changeTwoArray`)

```
public static void changeTwoElement(int[] array,int e1,int e2,int
index1,int index2){
    if(e1>e2){
        array[index1] = e2;
        array[index2] = e1;
    }
}
public static void changeTwoArray(int[][] array,int[] e1,int[]
e2,int index1,int index2){
    if(e1[0]>e2[0]){
        array[index1] = e2;
        array[index2] = e1;
    }
}
```

These two methods are very similar in that they **swap two elements** (if e1 greater than e2 then will swap), need the two elements and the index of the two elements.

### 2.2 What I've learned

This is my first attempt to write a Selection sorted program in Java. I wrote a similar program in Python, and I use nested loops more skilfully.

### 2.3 Problems and bugs encountered and solutions

1. **Loop out of array range**: Not knowing where the loop ends in a nested loop results in going out of scope

   **solution**: use debug to find where loss maybe - 1 or + 1.

2. **Exception in thread "main" java.lang.NullPointerException**: A brace was accidentally removed from the modified code.

3. **Not store the input array**: which cause that the input array has be changed, cause some mistake in the nest loop.

## 3. Insertion Sort

**Algorithm**: insertion sort iterates through the array and keeps part of the array in sorted order until it reaches the end.

Compare two numbers at a time and swap if large, or not swap if less, swap in pairs until the first digit of the data

## 3.1 The methods of selection sort

### 3.1.1 Method 5 (`insertionSort`)

```
public static int[] insertionSort(int[] test){
    // outer loop
    for(int i = 1; i < test.length; i++){
        if(i==1){
            Pa.changeTwoElement(test,test[i-1],test[i],0,i);
        }else{
            // inner loop
            // compare and swap in pairs until the first digit of
the data
            for(int theIndex = i;theIndex>0;theIndex--){
                Pa.changeTwoElement(test,test[theIndex-
1],test[theIndex],theIndex-1,theIndex);
            }
        }
    }
    return test;
}
```

In this method:

1. Contains a **nested loop** that starts at the second part of the array and, if $i = 1$, compares the first two number and puts the larger number to the right. **Inner loop** compare two numbers at a time and swap if large, or not swap if less, swap in pairs **until the first digit of the data**.
2. Finally, return `test` of the array that **has been sorted**.

### 3.1.2 Method 6 (`insertionSort2D`)

It also based on the method `insertionSort`.

```
public static int[][] insertionSort2D(int[][] test){
    for(int i = 1; i < test.length; i++){
        if(i==1){
            Pa.changeTwoArray(test,test[i-1],test[i],0,i);
        }else{
            for(int theIndex = i;theIndex>0;theIndex--){
                Pa.changeTwoArray(test,test[theIndex-
1],test[theIndex],theIndex-1,theIndex);
            }
        }
    }
    return test;
}
```

In this method is same as `insertionSort`.

### 3.2 What I've learned

Know how to use insertion sort algorithm in Java.

### 3.3 Problems and bugs encountered and solutions

This calculation is relatively simple, with no bugs or difficulties.

## 4. Test Part & How to run the code?

### 4.1 Test Part

```
class testToProjectA{
    public static void main(String[] args) {
        int[] testA = new int[]{1, 62, 81, 0, 23, 55, 76, 87, 20,
54, 65, 76, 1};
        int[][] testB = new int[][]{{71,2},{64,8},{31,56},{98,1},
{3,6},{59,837},{49,58},{61,8}};

        // testA print
        System.out.print("The initial testA is: ");
        for (int i = 0; i < testA.length; i++){
```

```
            System.out.print(testA[i]+" ");
        }
        System.out.println();

        // testA sorted by selectionSort print
        int[] selectionSortedTestA = Pa.selectionSort(testA);
        System.out.print("The testA sorted by selection sort is:
");
        for (int i = 0; i < selectionSortedTestA.length;i++){
            System.out.print(selectionSortedTestA[i]+" ");
        }
        System.out.println();

        // testA sorted by insertionSort print
        int[] insertionSortedTestA = Pa.insertionSort(testA);
        System.out.print("The testA sorted by insertion sort is:
");
        for (int i = 0; i < insertionSortedTestA.length;i++){
            System.out.print(insertionSortedTestA[i]+" ");
        }
        System.out.println();

        // testB print
        System.out.print("The initial testB is: ");
        for (int i = 0; i < testB.length;i++){
            System.out.print("("+testB[i][0] + " " + testB[i]
[1]+")"+" ");
        }
        System.out.println();
        int[][] selectionSortedTestB = Pa.selectionSort2D(testB);
        int[][] insertionSortedTestB = Pa.insertionSort2D(testB);

        // testB sorted by selectionSorted print
        System.out.print("The testB sorted by selectionSorted is:
");
        for (int i = 0; i < selectionSortedTestB.length;i++){
            System.out.print("("+selectionSortedTestB[i][0] + " " +
selectionSortedTestB[i][1]+")"+" ");
        }
        System.out.println();

        // testB sorted by insertionSorted print
```

```
        System.out.print("The testB sorted by insertionSorted is:
");
        for (int i = 0; i < insertionSortedTestB.length;i++){
            System.out.print("("+insertionSortedTestB[i][0] + " " +
insertionSortedTestB[i][1]+")"+" ");
        }
        System.out.println();
    }
}
```

**Two examples** are included in the test, one is **one-dimensional array** and the other is **two-dimensional array**, and all the results can be printed out.

The following figure shows the output:

```
The initial testA is: 1 62 81 0 23 55 76 87 20 54 65 76 1
The testA sorted by selection sort is: 0 1 1 20 23 54 55 62 65 76 76 81 87
The testA sorted by insertion sort is: 0 1 1 20 23 54 55 62 65 76 76 81 87
The initial testB is: (71 2) (64 8) (31 56) (98 1) (3 6) (59 837) (49 58) (61 8)
The testB sorted by selectionSorted is: (3 6) (31 56) (49 58) (59 837) (61 8) (64 8) (71 2) (98 1)
The testB sorted by insertionSorted is: (3 6) (31 56) (49 58) (59 837) (61 8) (64 8) (71 2) (98 1)
```

## 4.2 How to run the code?

This program **already has two examples** in the main function, `testA` and `testB`, which can be **run directly** to get sorted arrays. You can also modify the two test instances, **but pay attention** to the format of `testB` because `testB` is sorted by key.

# Project - B

## 1. The code and logic of the program

In the project - B, have two class and one is Pb which have three methods:

1. `changeTwoElement`
2. `quickSort`
3. `quickSort2D`

And other class is the test class which named `testToProjectB`.

## 2. Quick Sort

**Algorithms**: based on the sorting $\frac{n}{x}$ elements is more efficient that sorting $n$ elements. And other idea is we can sort an array into three partitions (less than, equal to and greater than)

### 2.1 The methods of quick sort

#### 2.1.1 Method 1 (`changeTwoElement`)

```
public static void changeTwoElement(int[] array,int e1,int e2,int
index1,int index2){
    array[index1] = e2;
    array[index2] = e1;
}
```

this method is to exchange two element.

#### 2.1.2 Method 2 (`quickSort`)

```
public static void quickSort(int[] a, int lo, int hi){
    int low, high, pivotNumber;
    low = lo;
    high = hi;
```

```
    // this if statement make sure when the low >= high, Stop the
iteration
    if(low >= high){
        return;
    }
    // set the first number is the pivotNumber
    pivotNumber = a[low];
    while(low<high) {
        // find the number of greater than pivotNumber
        while (pivotNumber < a[high] && high>low) {
            high--;
        }
        // find the number of less than or equal to pivotNumber
        while (pivotNumber >= a[low] && low<high) {
            low++;
        }
        // exchange these two number
        if(low<high){
            Pb.changeTwoElement(a, a[high], a[low], high, low);
        }
    }
    // move the pivot number
    Pb.changeTwoElement(a, a[lo], a[low], lo, low);
    // Quick sort for the part less than the reference value
    quickSort(a,lo,low-1);
    // Quick sort for portions that are larger than the reference
value
    quickSort(a,high+1,hi);
}
```

In the method:

1. Require input one array, the low index and high index.
2. The nest while loop will make the number **less** than pivot value to the **left**,number **greater** than pivot to the **end**, and number equal to pivot value comes **after**.
3. Then quick sort is done for the parts less than the Pivot number and greater than the pivot number (**Using iterations**)

#### 2.1.3 Method 3 (`quickSort2D`)

```
public static void quickSort2D(int[][] a, int lo, int hi){
    // use loop to quick sort each rows in the 2D array
    for(int i = 0; i < a.length; i++){
        quickSort(a[i],lo,hi);
    }
}
```

This method makes use of method `quickSort` to sort each row.

### 2.2 What I've learned

It was the first time I used the iterative method in a Java program, and I fully understood the algorithm principle of Quick Sort.

### 2.3 Problems and bugs encountered and solutions

1. Initially, I wanted to split the entire array **into three parts** (less than, equal to, and greater than Pivot), just like in class. I then wrote the following code, but found it difficult to iterate this way because I would need to concatenate each part, so I abandoned the idea and decided to sort **directly inside the array**.

```
public static void addArray(int[] array,int number){
    int[] temporaryArray = new int[array.length+1];
    temporaryArray[array.length] = number;
    for(int i = 0; i < array.length; i++){
        temporaryArray[i] = array[i];
    }
    array = temporaryArray;
}
public static void getThreePartition(int[] lessPartition,
int[] equalPartition, int[] greaterPartition, int n, int
pivotNumber){
    if(n == pivotNumber){
        pB.addArray(equalPartition,n);
    }
    if(n > pivotNumber){
        pB.addArray(greaterPartition,n);
    }
```

```
    if(n < pivotNumber){
        pB.addArray(lessPartition,n);
    }
}
    public static int[] quickSort1(int[] array,int lo,int
hi){
        int[] lessPartition = new int[0];
        int[] equalPartition = new int[0];
        int[] greaterPartition = new int[0];
        int pivotNumber = array[0];
        for(int i = 0; i < array.length; i++){

 pB.getThreePartition(lessPartition,equalPartition,greaterPa
rtition,array[i],pivotNumber);
        }
        return array;
}
```

2. There were a number of times when we wrote this class that we went out of the loop. The problem was with the condition of the while loop.

   **solution**: I sketched out the sequence of the entire program on paper, using small array instances to derive it.

## 3. Test Part & How to run the code?

### 3.1 Test Part

```
class testToProjectB{
    public static void main(String[] args) {
        int[] testCase1 = new int[]{31,33,27,15,42,11,40,5,19,21};
        // testCase1 print
        System.out.print("The initial testCase1 is: ");
        for (int i = 0; i < testCase1.length; i++){
            System.out.print(testCase1[i] + " ");
        }
        System.out.println();

        int[][] testCase2 = new int[][]{
                {98,34,100,36,44,64,3,99,59},
                {20,88,55,91,14,58,25,29,44},
```

```
                {66,62,4,65,49,71,71,24,12},
                {14,3,58,23,12,66,11,45,36},
                {55,64,35,24,85,73,33,85,46},
                {94,76,23,36,57,26,8,92,17},
                {85,68,52,34,53,93,4,37,34},
                {70,9,15,42,31,16,72,61,62},
                {11,38,34,21,81,9,45,68,11},
                {20,83,27,6,69,26,5,31,8},
                {74,97,11,60,1,68,14,27,46}};
        // testCase2 print
        System.out.println("The initial testCase2 is: ");
        for(int i = 0; i < testCase2.length; i++){
            for(int j = 0; j < testCase2[i].length;j++){
                System.out.print(testCase2[i][j] + " ");
            }
            System.out.println();
        }
        System.out.println();

        // testCase1 sorted by quickSort print
        System.out.print("The testCase1 sorted by quickSort is: ");
        Pb.quickSort(testCase1,0,testCase1.length-1);
        for(int i = 0; i < testCase1.length; i++){
            System.out.print(testCase1[i]+" ");
        }
        System.out.println();

        // testCase2 sorted by quickSort2D print
        System.out.println("The testCase2 sorted by quickSort2D is:
");
        Pb.quickSort2D(testCase2,0,testCase2[0].length-1);
        for(int i = 0; i < testCase2.length; i++){
            for(int j = 0; j < testCase2[i].length;j++){
                System.out.print(testCase2[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

**Two examples** are included in the test, one is **one-dimensional array** and the other is **two-dimensional array**, and all the results can be printed out.

The following figure shows the output:

```
The initial testCase1 is: 31 33 27 15 42 11 40 5 19 21
The initial testCase2 is:
98 34 100 36 44 64 3 99 59
20 88 55 91 14 58 25 29 44
66 62 4 65 49 71 71 24 12
14 3 58 23 12 66 11 45 36
55 64 35 24 85 73 33 85 46
94 76 23 36 57 26 8 92 17
85 68 52 34 53 93 4 37 34
70 9 15 42 31 16 72 61 62
11 38 34 21 81 9 45 68 11
20 83 27 6 69 26 5 31 8
74 97 11 60 1 68 14 27 46
```

```
The testCase1 sorted by quickSort is: 5 11 15 19 21 27 31 33 40 42
The testCase2 sorted by quickSort2D is:
3 34 36 44 59 64 98 99 100
14 20 25 29 44 55 58 88 91
4 12 24 49 62 65 66 71 71
3 11 12 14 23 36 45 58 66
24 33 35 46 55 64 73 85 85
8 17 23 26 36 57 76 92 94
4 34 34 37 52 53 68 85 93
9 15 16 31 42 61 62 70 72
9 11 11 21 34 38 45 68 81
5 6 8 20 26 27 31 69 83
1 11 14 27 46 60 68 74 97
```

### 3.2 How to run the code?

This program **already has two examples** in the main function, `testCase1` and `testCase2`, which can be **run directly** to get sorted arrays. You can also modify the two test instances.

# Project - C

## 1. The code and logic of the program

In the project - C, have two class and one is Pc which have three methods:

1. `merge`
2. `mergeSort`
3. `mergeSort2D`

And other class is the test class which named `testToProjectC`.

## 2. Merge Sort

**Merging sort list**: put two arrays into one array and sort it

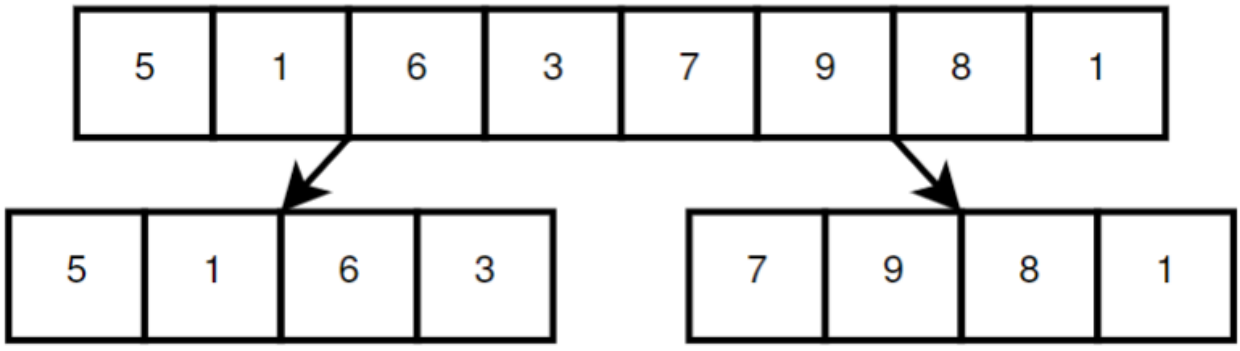
this operation can be completed in O($n$) time.

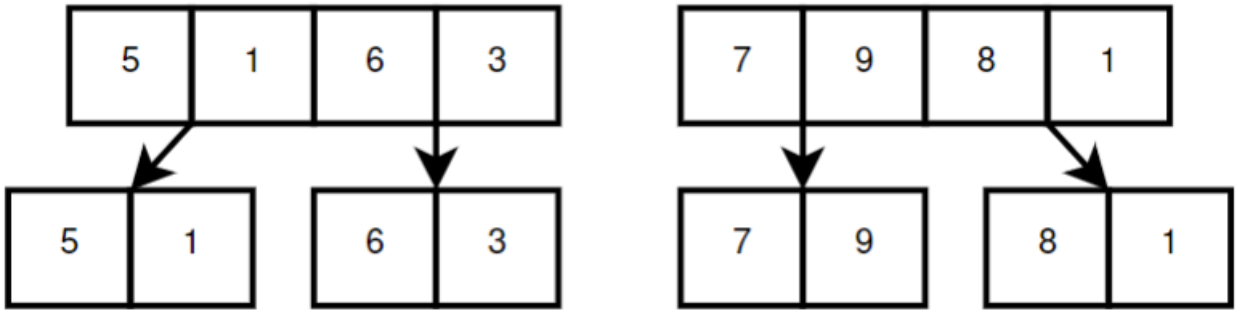
the process of combining two lists is called **merging**.

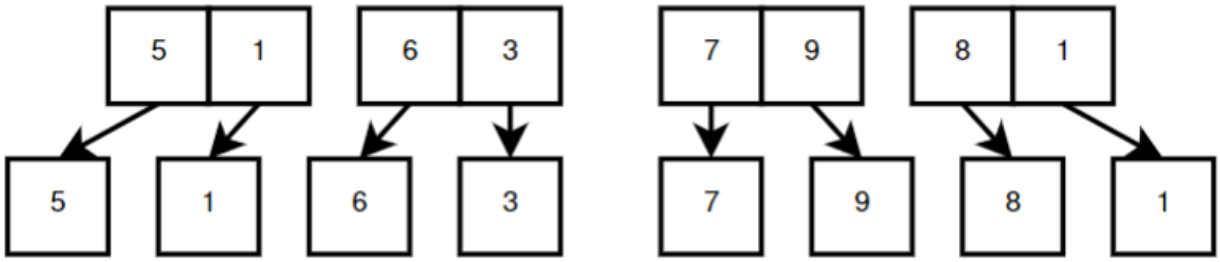
**Merge Sort**:

Mergesort takes advantage of the efficiency of merging already sorted list

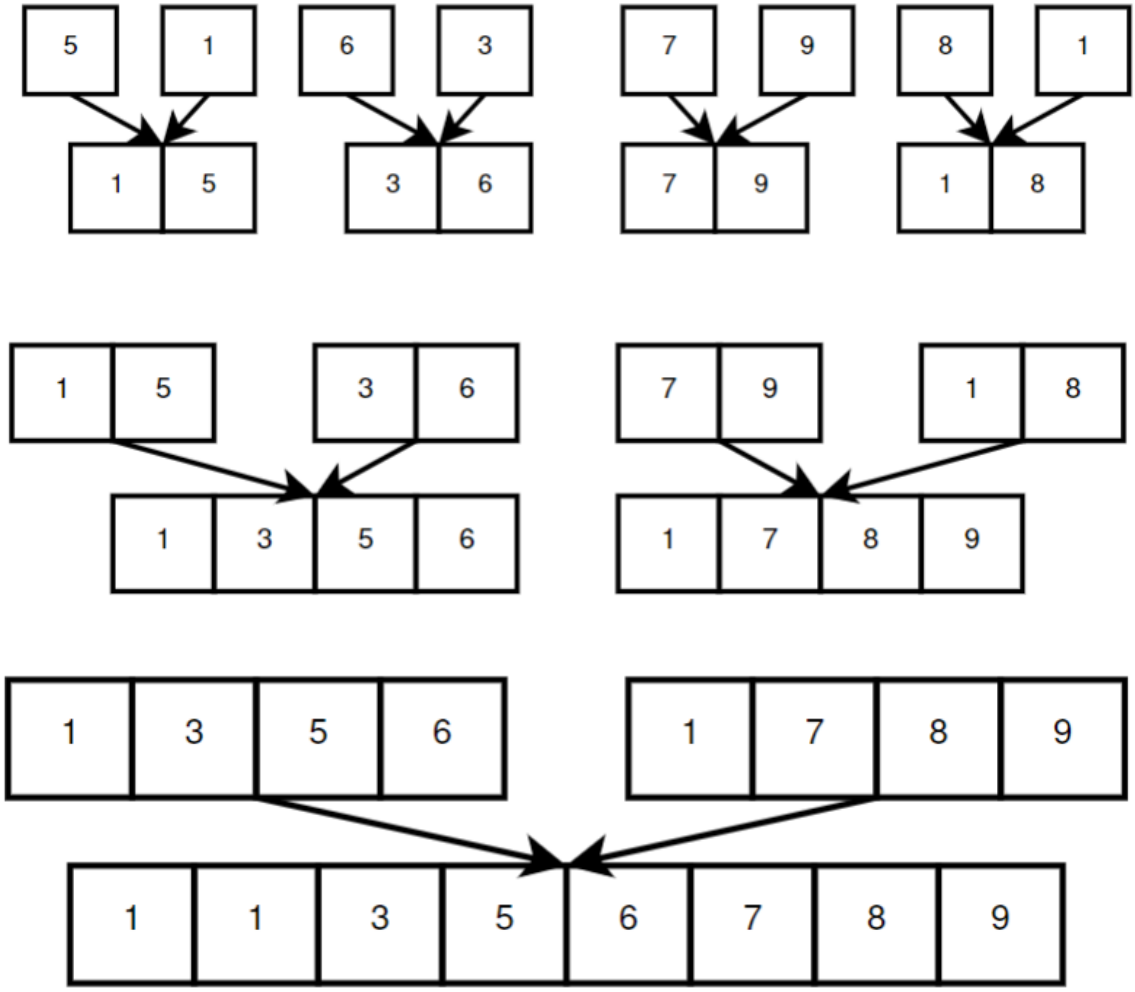
1. **Mergesort divides the array in half repeatedly until there is only one element in each partitions**






2. then use merging sorted lists to combine two arrays into one array



## 2.1 The methods of merge sort

### 2.1.1 Method 1 (`merge`)

```
public static void merge(int[] arr, int start, int middle, int end)
{
    // create an all 0 same size array
    int[] temArr = new int[arr.length];
    // store start, middle and start in s,m and i
    int s = start;
    int m = middle;
    int i = start;
    // Start by comparing the first digit of the two parts and
place the smaller digit into the new array
    while(s <= middle && m + 1 <= end){
        if(arr[s] < arr[m + 1]){
            temArr[i] = arr[s];
            i++;
            s++;
        }else {
            temArr[i] = arr[m + 1];
            i++;
            m++;
        }
    }
    // If the right part is arranged, add the remaining digits on
the left to the new array
    while(s<=middle){
        temArr[i] = arr[s];
        s++;
        i++;
    }
    // If the left part is arranged, add the remaining numbers on
the right to the new array
    while (m+1<=end){
        temArr[i] = arr[m + 1];
        m++;
        i++;
    }
    // store the sorted array in arr
    for(int j = start; j <= end;j++){
        arr[j] = temArr[j];
    }
```

```
}
```

In this method:

1. This method is same as Merging sort list, merge two arranged arrays into one array and sort. In this method, two arranged arrays are the `arr` index from `start` to `middle` and the `arr` index from `middle+1` to `end`
2. The first while loop is start by comparing the first digit of the two parts and place the smaller digit into the new array.
3. If all the numbers on the right side of the `arr` are placed in the new array but not all the numbers on the left are placed in the new array, then the numbers on the left are placed in the new array in sequence
4. Third while loop is If all the numbers on the left side of the `arr` are placed in the new array but not all the numbers on the right side of the `arr` are placed in the new array in sequence

#### 2.1.2 Method 2 (`mergeSort`)

```
public static void mergeSort(int[] arr1, int left, int right){
    if (left < right){
        // iterate over binary arrays until each array has only one
number
        int middle = (left + right)/2;
        mergeSort(arr1,left,middle);
        mergeSort(arr1,middle+1,right);
        // Sort the two arrays into one array using mergesort
        merge(arr1,left,middle,right);
    }
}
```

The logic of this algorithm can be seen in 2.Merge Sort, the picture visually reflects the whole process

#### 2.1.3 Method 3 (`mergeSort2D`)

```
public static void mergeSort2D(int[][] arr,int left,int right){
    for(int i = 0; i < arr.length; i++){
        mergeSort(arr[i],left,right);
    }
}
```

This method use a for loop, to use `mergeSort` to sorting each rows of the 2D array.

### 2.2 What I've learned

I spent more time on this project, because I am not very familiar with the use of **iteration**, I started to think about using a loop, but found it too difficult and continued to think about how to use iteration. In Method 2 (`mergeSort`) it took me a lot of time and a lot of thought to understand the use of iteration.

### 2.3 Problems and bugs encountered and solutions

1. The most common problems are those that are out of the range

   **solution**: use debug to see each variable change and found where is wrong
2. Sometimes it's hard to imagine an iterative approach to a problem

   **solution**: Write more programs like this

## 3. Test Part & How to run the code?

### 3.1 Test Part

```
class testToProjectC{
    public static void main(String[] args) {
        int[] testCase1 = new int[]{31,33,27,15,42,11,40,5,19,21};
        // testCase1 print
        System.out.print("The initial testCase1 is: ");
        for (int i = 0; i < testCase1.length; i++){
            System.out.print(testCase1[i] + " ");
        }
        System.out.println();

        int[][] testCase2 = new int[][]{
                {98,34,100,36,44,64,3,99,59},
                {20,88,55,91,14,58,25,29,44},
                {66,62,4,65,49,71,71,24,12},
                {14,3,58,23,12,66,11,45,36},
                {55,64,35,24,85,73,33,85,46},
                {94,76,23,36,57,26,8,92,17},
                {85,68,52,34,53,93,4,37,34},
```

```
                {70,9,15,42,31,16,72,61,62},
                {11,38,34,21,81,9,45,68,11},
                {20,83,27,6,69,26,5,31,8},
                {74,97,11,60,1,68,14,27,46}};
        // testCase2 print
        System.out.println("The initial testCase2 is: ");
        for(int i = 0; i < testCase2.length; i++){
            for(int j = 0; j < testCase2[i].length;j++){
                System.out.print(testCase2[i][j] + " ");
            }
            System.out.println();
        }
        System.out.println();

        // testCase1 sorted by mergeSort print
        Pc.mergeSort(testCase1,0,testCase1.length-1);
        System.out.print("The testCase1 sorted by mergeSort is: ");
        for(int i = 0; i < testCase1.length; i++){
            System.out.print(testCase1[i]+" ");
        }
        System.out.println();

        // testCase2 sorted by mergeSort2D print
        System.out.println("The testCase2 sorted by mergeSort2D is:
");
        Pc.mergeSort2D(testCase2,0,testCase2[0].length-1);
        for(int i = 0; i < testCase2.length; i++){
            for(int j = 0; j < testCase2[i].length;j++){
                System.out.print(testCase2[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

**Two examples** are included in the test, one is **one-dimensional array** and the other is **two-dimensional array**, and all the results can be printed out.

The following figure shows the output:

```
The initial testCase1 is: 31 33 27 15 42 11 40 5 19 21
The initial testCase2 is:
98 34 100 36 44 64 3 99 59
20 88 55 91 14 58 25 29 44
66 62 4 65 49 71 71 24 12
14 3 58 23 12 66 11 45 36
55 64 35 24 85 73 33 85 46
94 76 23 36 57 26 8 92 17
85 68 52 34 53 93 4 37 34
70 9 15 42 31 16 72 61 62
11 38 34 21 81 9 45 68 11
20 83 27 6 69 26 5 31 8
74 97 11 60 1 68 14 27 46
```

```
The testCase1 sorted by mergeSort is: 5 11 15 19 21 27 31 33 40 42
The testCase2 sorted by mergeSort2D is:
3 34 36 44 59 64 98 99 100
14 20 25 29 44 55 58 88 91
4 12 24 49 62 65 66 71 71
3 11 12 14 23 36 45 58 66
24 33 35 46 55 64 73 85 85
8 17 23 26 36 57 76 92 94
4 34 34 37 52 53 68 85 93
9 15 16 31 42 61 62 70 72
9 11 11 21 34 38 45 68 81
5 6 8 20 26 27 31 69 83
1 11 14 27 46 60 68 74 97
```

### 3.2 How to run the code?

This program **already has two examples** in the main function, `testCase1` and `testCase2`, which can be **run directly** to get sorted arrays. You can also modify the two test instances.

# Project - D

## 1. The code and logic of the program

1. `bucketSort`
2. `bucketSort2D`

## 2. Bucket Sort

**Definition:**

- Set up an array of m buckets (**created to put elements into them**) where each bucket (linked list) is responsible for one possible key value
- place each item in the correct bucket
- concatenate all of the lists from the buckets to get the final sorted order (start with the bucket that stores the lowest key and add the others in order)

### 2.1 The methods of merge sort

#### 2.1.1 Method 1 (`bucketSort`)

```
public static void bucketSort(int[] arr){
    // find the minimum value
    int min = arr[0];
    for(int i = 0; i < arr.length;i++){
        if(min > arr[i]){
            min = arr[i];
        }
    }
    // find the maximum value
    int max = arr[0];
    for(int i = 0; i < arr.length;i++){
        if(max<arr[i]){
            max = arr[i];
        }
    }
    // create a bucket to put the element in (In this program I set
the interval to 1)
```

```
    int[] bucket = new int[max-min+1];
    int[] temArr = new int[arr.length];
    // put the number in each bucket
    for(int i = 0; i < arr.length; i++){
        bucket[arr[i]-min]++;
    }
    // take the element out and join them to get the sorted array
    int j = 0;
    for (int i = 0; i < bucket.length; i++){
        if(bucket[i]!=0){
            for(int a = 0; a < bucket[i]; a++){
                temArr[j] = i + min;
                j++;
            }
        }
    }
    // store the sorted array in arr
    for(int i = 0; i < arr.length;i++){
        arr[i] = temArr[i];
    }
}
```

In the method:

1. Find the maximum value and minimum in the `arr`
2. Create a new array (bucket) which size is $max - min + 1$ to put the element in
3. Use for loop to put the elements in the bucket
4. Use a nest loop to take the element out and join them to get the sorted array
5. Store the sorted array in `arr`

### 2.1.2 Method 2 (`bucketSort2D`)

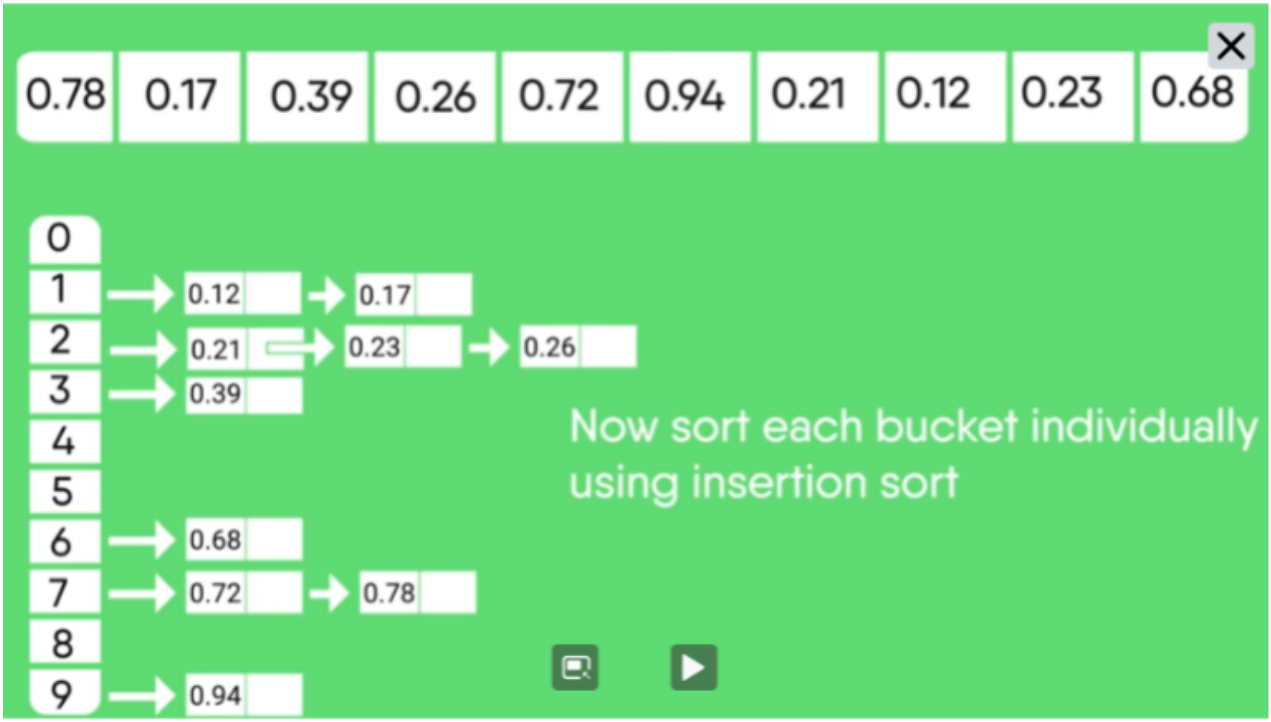
```
public static void bucketSort2D(int[][] arr){
    for (int i = 0; i < arr.length; i++){
        Pd.bucketSort(arr[i]);
    }
}
```

This method use a for loop to use `bucketSort` to sort each lows in the 2D array.

### 2.2 What I've learned

I learned how to use bucket sort, as well as a number of other buck sorts implemented with other sorting methods by querying data.

for example in the class:




### 2.3 Problems and bugs encountered and solutions

1. I'm just using a simple bucket sort for this program without recursion or other sort methods, so the downside is that it only works with **small amounts of data and small ranges of data.**

## 3. Test Part & How to run the code?

### 3.1 Test Part

```
class testToProjectD{
    public static void main(String[] args) {
        int[] testCase1 = new int[]{31,33,27,15,42,11,40,5,19,21};
        // testCase1 print
        System.out.print("The initial testCase1 is: ");
        for (int i = 0; i < testCase1.length; i++){
```

```
            System.out.print(testCase1[i] + " ");
        }
        System.out.println();

        int[][] testCase2 = new int[][]{
                {98,34,100,36,44,64,3,99,59},
                {20,88,55,91,14,58,25,29,44},
                {66,62,4,65,49,71,71,24,12},
                {14,3,58,23,12,66,11,45,36},
                {55,64,35,24,85,73,33,85,46},
                {94,76,23,36,57,26,8,92,17},
                {85,68,52,34,53,93,4,37,34},
                {70,9,15,42,31,16,72,61,62},
                {11,38,34,21,81,9,45,68,11},
                {20,83,27,6,69,26,5,31,8},
                {74,97,11,60,1,68,14,27,46}};
        // testCase2 print
        System.out.println("The initial testCase2 is: ");
        for(int i = 0; i < testCase2.length; i++){
            for(int j = 0; j < testCase2[i].length;j++){
                System.out.print(testCase2[i][j] + " ");
            }
            System.out.println();
        }
        System.out.println();

        // testCase1 sorted by bucket print
        Pd.bucketSort(testCase1);
        System.out.print("The testCase1 sorted by bucket is: ");
        for(int i = 0; i < testCase1.length; i++){
            System.out.print(testCase1[i]+" ");
        }
        System.out.println();

        // testCase2 sorted by bucketSort2D print
        System.out.println("The testCase2 sorted by bucketSort2D
is: ");
        Pd.bucketSort2D(testCase2);
        for(int i = 0; i < testCase2.length; i++){
            for(int j = 0; j < testCase2[i].length;j++){
                System.out.print(testCase2[i][j] + " ");
            }
            System.out.println();
```

```
            }
        }
    }
```

**Two examples** are included in the test, one is **one-dimensional array** and the other is **two-dimensional array**, and all the results can be printed out.

The following figure shows the output:

```
The initial testCase1 is: 31 33 27 15 42 11 40 5 19 21
The initial testCase2 is:
98 34 100 36 44 64 3 99 59
20 88 55 91 14 58 25 29 44
66 62 4 65 49 71 71 24 12
14 3 58 23 12 66 11 45 36
55 64 35 24 85 73 33 85 46
94 76 23 36 57 26 8 92 17
85 68 52 34 53 93 4 37 34
70 9 15 42 31 16 72 61 62
11 38 34 21 81 9 45 68 11
20 83 27 6 69 26 5 31 8
74 97 11 60 1 68 14 27 46
```

```
The testCase1 sorted by bucket is: 5 11 15 19 21 27 31 33 40 42
The testCase2 sorted by bucketSort2D is:
3 34 36 44 59 64 98 99 100
14 20 25 29 44 55 58 88 91
4 12 24 49 62 65 66 71 71
3 11 12 14 23 36 45 58 66
24 33 35 46 55 64 73 85 85
8 17 23 26 36 57 76 92 94
4 34 34 37 52 53 68 85 93
9 15 16 31 42 61 62 70 72
9 11 11 21 34 38 45 68 81
5 6 8 20 26 27 31 69 83
1 11 14 27 46 60 68 74 97
```

### 3.2 How to run the code?

This program **already has two examples** in the main function, `testCase1` and `testCase2`, which can be **run directly** to get sorted arrays. You can also modify the two test instances.