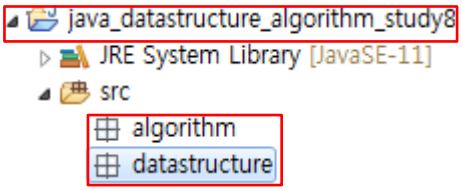
1. 프로젝트 만들기



2. 알고리즘 수업 목록

- 최대값(Maximum Value) 찾기
- 버블(Bubble) 정렬 알고리즘
- 최소값(Minimum Value) 찾기
- 선택(Selection) 정렬 알고리즘
- 삽입(Insert) 정렬(Sorting) 알고리즘
- 퀵(Quick) 정렬(Sorting) 알고리즘

3. 최대값 찾기

가. 배열명: arrays[]

나. 인덱스(요소 지시자): i

다. 시작배열

60	50	**95**	80	70

라. 방법

- arrays[i]와 arrays[i+1]과 비교
- 조건 arrays[i] > arrays[i+1]이 만족하면 교환
- 마지막 숫자까지 계속 반복
- 최종적으로 arrays[length -1]이 최대값이 됨

마. 진행과정

i값 변화

60	**50**	95	80	70
↓				
50	**60**	95	80	70

50	**60**	**95**	80	70
↓				
50	**60**	**95**	80	70

50	60	**95**	**80**	70
↓				
50	60	**80**	**95**	70

50	60	80	**95**	**70**
↓				
50	60	80	**70**	**95**

바. 최종배열

50	60	80	70	**95**

사. 소스

```
public class TestMaxValue {
```

```
    public static void main(String[] args) {
            int [] scores = { 60, 50, 95, 80, 70 };
            int maxValue = max (scores);
            System.out.println("Max Value = " + maxValue);
    }
    public static int max(int [] arrays) {
            // Maximum initialization value is 0
            for (int i = 0; i < arrays.length - 1; i++) {
                    if (arrays[i] > arrays[i + 1]) { // swap
                            int temp = arrays[i];
                            arrays[i] = arrays[i + 1];
                            arrays[i + 1] = temp;
                    }
            }
            int maxValue = arrays[arrays.length - 1];
            return maxValue;
    }
}
```

Result:

```
Max Value = 95
```

4. 버블 정렬 알고리즘

가. 배열명: arrays[]

나. 인덱스(요소 지시자): i

다. 시작배열

60	**50**	95	80	70

라. 방법

- arrays[i]와 arrays[i+1] 비교
- 만약, arrays[i]가 arrays[i+1]보다 크다면 양쪽 값을 교환
- 정렬이 완료될 때까지 반복

마. 진행과정 **첫번째**

50	**60**	95	80	70

50	**60**	**95**	80	70

50	**60**	**95**	80	70

50	60	**95**	**80**	70

50	60	**80**	**95**	70

50	60	80	**95**	**70**

50	60	80	**70**	**95**

첫번째 최대값

두번째

50	**60**	80	70	95

50	**60**	80	70	95

50	**60**	**80**	70	95

50	**60**	**80**	70	95

50	60	**80**	**70**	95

↓

50	60	**70**	**80**	95

50	60	70	**80**	**95**

↓

50	60	70	**80** 두번재 최대값	**95**

바. 최종배열

50	60	70	80	95

사. 소스

```
public class TestBubbleSort {
    public static void main(String[] args) {
            int [] scores = { 60, 50, 95, 80, 70 };
            sort (scores);
            for (int i = 0; i < scores.length ; i++) {
                    System.out .print(scores[i] + "," );
            }
    }
    public static void sort(int [] arrays) {
            for (int i = 0; i < arrays.length - 1; i++) {
                    boolean isSwap = false ;
                    for (int j = 0; j < arrays.length - i - 1; j++) {
                            if (arrays[j] > arrays[j + 1]) {//swap
                                    int temp = arrays[j];
                                    arrays[j] = arrays[j + 1];
                                    arrays[j + 1] = temp;
                                    isSwap = true ;
                            }
                    }
                    if (!isSwap) //No swap so terminate sorting
                    {
                            break ;
                    }
            }
    }
}
```

Result:

50,60,70,80,95,

5. 최소값 찾기

가. 배열명: arrays[]

나. 인덱스(요소 지시자): i, j

다. 시작 배열

60	80	95	**50**	70

라. 방법

- minIndex=0, j=1을 초기값으로 지정
- arrays[minIndex]와 arrays[j]를 비교
- 만약 arrays[minIndex]가 arrays[j]보다 크다면,
- minIndex에 j를 대입, 기존 j값은 1 증가시킴
- 반복이 끝나면, arrays[minIndex]가 최소값이 됨

마. 진행 과정

minIndex=0 j=1

60	**80**	95	50	70

↓

60	**80**	95	50	70

minIndex=0 j=2

60	80	**95**	50	70

↓

60	80	**95**	50	70

minIndex=0 j=3

60	80	95	**50**	70

↓

60	80	95	50	70

minIndex=3 j=4

60	80	95	**50**	**70**

↓ minIndex=3

60	80	95	**50**	70

바. 최종 배열

60	80	95	**50**	70

사. 소스

```
public class TestMinValue {
    public static void main(String[] args) {
            int [] scores = { 60, 80, 95, 50, 70 };
            int minValue = min (scores);
            System.out .println("Min Value = " + minValue);
    }
    public static int min(int [] arrays) {
            int minIndex = 0;// the index of the minimum
            for (int j = 1; j < arrays.length ; j++) {
                    if (arrays[minIndex] > arrays[j]) {
                            minIndex = j;
                    }
            }
            return arrays[minIndex];
    }
}
```

Result:

```
Min Value = 50
```

6. 선택 정렬 알고리즘

가. 배열명: arrays[]

나. 인덱스(요소 지시자): i, j

다. 추가 변수: minIndex

라. 방법

- 정렬되지 않은 부분에서 최소 값 요소를 반복적으로 찾아 배열을 정렬
- 최소값으로 찾아진 요소의 인덱스 값을 시작위치와 교환

마. 시작배열

60	80	95	50	70

바. 진행과정

1) 첫 번째 정렬

60 **minIndex=0** i=0	**80** j=1	95	50	70
60 **minIndex=0** i=0	80	**95** j=2	50	70
60 **minIndex=0** i=0	80	95	**50** j=3	70
↓				
60 i=0	80	95	**50** **minIndex=3** j=3	70
60 i=0	80	95	**50** **minIndex=3** j=3	**70** j=4
60 i=0	80	95	**50** **minIndex=3**	70

i!=minIndex && 60 > 50이면, 교환

2) 두 번째 정렬(i = 1)

3) 세 번째 정렬(i = 2)

4) 네 번째 정렬(i = 3)

사. 최종배열

50	60	70	80	95

사. 소스

```
public class TestSelectSort {
    public static void main(String[] args) {
            int [] scores = { 60, 80, 95, 50, 70 };
            sort (scores);
            for (int score : scores) {
                    System.out .print(score + "," );
            }
    }
    public static void sort(int [] arrays) {
            for (int i = 0; i < arrays.length - 1; i++) {
                    int minIndex = i; // the index of the selected minimum
                    for (int j = i + 1; j < arrays.length ; j++) {
                            if (arrays[minIndex] > arrays[j]) {
                                    minIndex = j;
                            }
                    }
                    if (i!= minIndex) //minimum arrays[i] is swaped with the
            arrays[minIndex]
                    {
                            int temp = arrays[i];
                            arrays[i] = arrays[minIndex];
                            arrays[minIndex] = temp;
                    }
            }
    }
}
```

Result:

50,60,70,80,95,

7. 삽입 정렬 알고리즘

가. 정렬 대상

80	70	60	50	95

나. 정렬 이미지

- 첫 번째 정렬

80	**70**	60	50	95

↓

70	**80**	60	50	95

- 두 번째 정렬 이미 정렬된 것 정렬되지 않은 것

70	**80**	**60**	50	95

70 > 60 80 > 60

↓

60	**70**	**80**	50	95

- 세 번째 정렬 이미 정렬된 것 정렬되지 않은 것

60	**70**	**80**	**50**	95

이미 정렬된 것 정렬되지 않은 것

50	**60**	**70**	**80**	95

60 > 50 80 > 50

- 네 번째 정렬 정렬된 것 정렬되지 않은 것

50	**60**	**70**	**80**	**95**

80 > 90

50	**60**	**70**	**80**	**95**

80 > 90

다. 소스 코드

```
public class TestInsertSort {
        public static void main(String[] args) {
                int [] scores = { 80, 70, 60, 50, 95};
                sort (scores);
                for (int score : scores) {
                        System.out .print(score + "," );
                }
        }

        public static void sort(int [] arrays) {
                for (int i = 1; i < arrays.length ; i++) {
                        //Take unsorted new elements
                        int insertElement = arrays[i];
                        int insertPosition = i;
                        for (int j = insertPosition - 1; j >= 0; j--) {
                                // insertElement is shifted to the right
                                if (insertElement < arrays[j]) {
                                        arrays[j + 1] = arrays[j];
                                        insertPosition--;
                                }else {
                                        break ;
                                }
                        }
                        //Insert the new element
                        arrays[insertPosition] = insertElement;
                }
        }
}
```

Result:

50,60,70,80,95,

8. 퀵 정렬 알고리즘

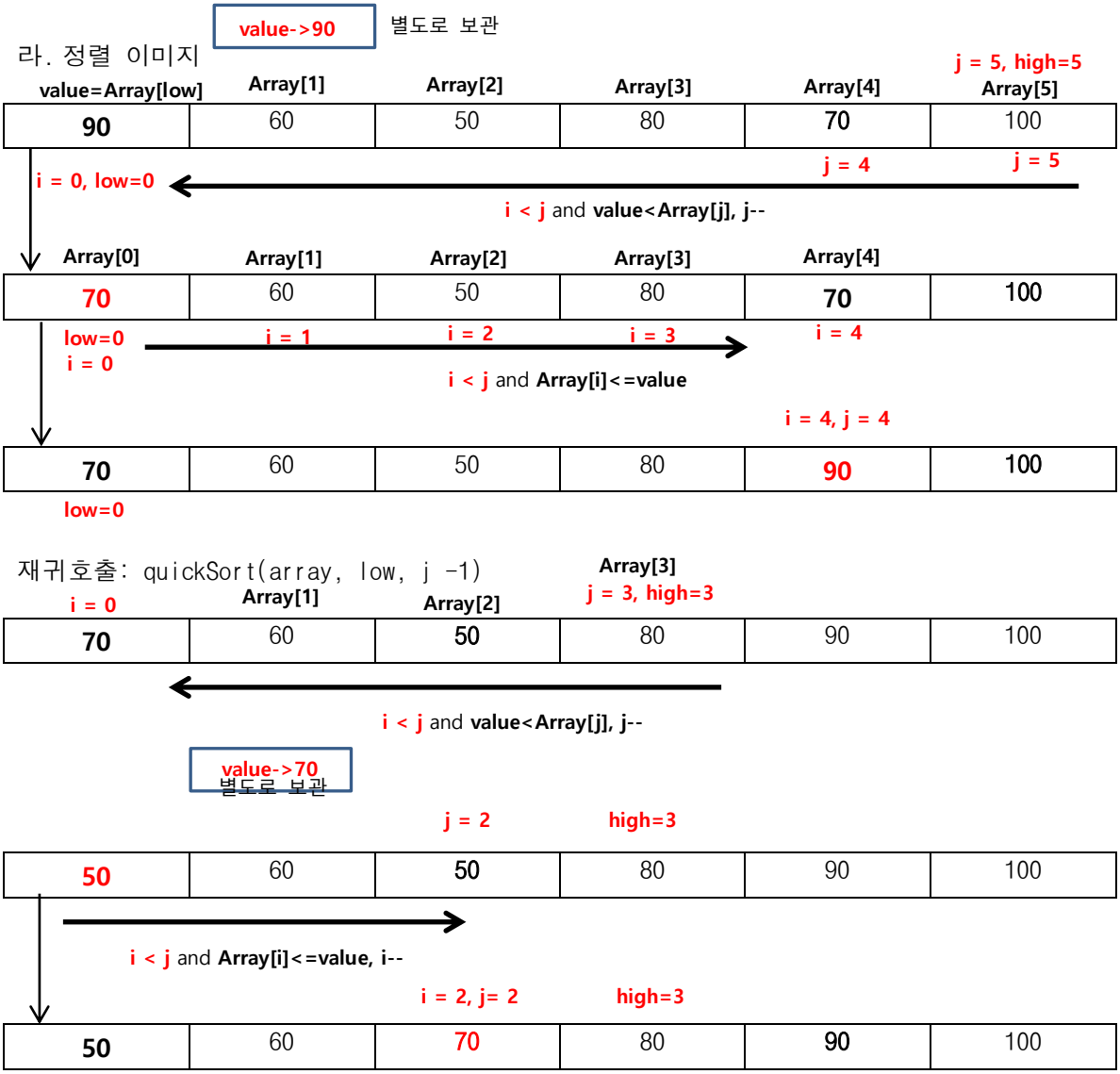
가. 핵심: 큰 배열을 두 개의 더 작은 배열로 나눔.

나. 배열명: Array[]

다. 대상 배열

90	60	50	80	70	100

라. 정렬 이미지



재귀호출: quickSort(array, low, j -1)

마. 소스 코드

```
public class TestQuickSort {
    private static void quickSort(int [] array, int low, int high) {
        if (low > high) {
            return;
        }
        int i = low;
        int j = high;
        int threshold = array[low];
        // Alternately scanned from both ends of the list
        while (i < j) {
            // Find the first position less than value from right to left
            while (i < j && array[j] > threshold) {
                    j--;
            } //Replace the low with a smaller number than the value
            if (i < j)
                    array[i++] = array[j];
                    // Find the first position greater than threshold from left to right
            while (i < j && array[i] <= threshold) {
                    i++;
            } //Replace the high with a number larger than the value
            if (i < j)
                    array[j--] = array[i];
        }
        array[i] = threshold;
        quickSort (array, low, i - 1); // left quickSort
        quickSort (array, i + 1, high); // right quickSort
    }

    public static void quickSort(int [] array) {
            if (array.length > 0) {
                    quickSort (array, 0, array.length - 1);
            }
    }

    public static void main(String[] args) {
            int [] scores = { 90, 60, 50, 80, 70, 100 };
            quickSort (scores);
            for (int i = 0; i < scores.length ; i++) {
                    System.out .print(scores[i] + "," );
            }
    }
}
```