1. (정렬: 자바 클래스 활용) 다음은 배열 내 자료를 정렬하기 위해 Arrays.sort()를 활용하는 코드 예이다. 현재 JDK 구현에서는, 기본자료형(예: int) 배열 n에 대해 Arrays.sort(n) 메소드는 퀵정렬의 한 유형인 Dual-Pivot QuickSort로 동작하며, 참조자료형 배열 m에 대해 Arrays.sort(m) 메소드는 합병정렬의 한 유형인 TimSort로 동작한다.

```
public class Test {
	public static void main(String[] args) {
		int	n[]={9,1,3,7,3,4,2,1,5};
		Arrays.sort(n);
		System.out.println(Arrays.toString(n));
	}
}
```

2. (**실습**: 거품정렬, bubble sort) 다음은 정수 배열을 거품정렬하는 코드 예이다. 아래 코드를 입력하고 실행하면서 거품정렬 구현법을 익히시오. (Reference: https://en.wikipedia.org/wiki/Bubble_sort, CC-BY-SA)

   - **실습 #1**: 다음 힌트를 참고하여 bubbleSort 함수를 좀 더 효율적으로 개선하시오. 힌트: 외부 for 루프는 각 반복마다 정렬되지 않은 자료 한 개의 정렬된 위치(입력 배열의 오른쪽 끝부터 시작하여 1씩 감소된 위치)를 결정하므로, 이전 반복을 통해 기정렬된 자료들은 이후 반복 시 재검사할 필요가 없다.
   - **실습 #2**: 아래 bubbleSort 함수를 좀 더 효율적으로 개선하시오. 힌트: 외부 for 루프의 특정 반복 시 교환이 전혀 발생하지 않았다면 자료는 이미 정렬된 상태이므로 더 이상의 반복은 필요 없다.
   - **실습 #3**: 배열 n과 n의 크기보다 작은 정수 k에 대해 kLargest(n, k);를 호출하면 배열 n 내 가장 큰 k개 정수들이 배열 n의 인덱스 n.length-k 위치부터 n.length-1 위치까지 저장된다고 할 때, private static void kLargest(int v[], int k)를 새롭게 정의하시오.

```
public class Test {
	public static void main(String[] args) {
		int	v[]={5,8,1,9,3,5,1,5};
		bubbleSort(v);
		System.out.println(Arrays.toString(v));
	}
	private static void bubbleSort(int[] v) {
		for (int i = 0; i < v.length; i++) {
			for (int j = 1; j < v.length; j++) {
				if(v[j-1]>v[j]) { // 인접 자료 간 순서 맞지 않으면 교환
					int	temp=v[j];
					v[j]=v[j-1];
					v[j-1]=temp;
				}
			}
		}
	}
}
```

3. (**실습**: 선택정렬, selection sort) 다음은 정수 배열을 선택정렬하는 코드 예이다. 아래 코드를 입력하고 실행하면서 선택정렬 구현법을 익히시오. (Reference: https://en.wikipedia.org/wiki/Selection_sort, CC-BY-SA)

- 선택정렬은 정렬되지 않은 리스트 L 내 최소값 위치를 찾아 L의 첫 위치 자료와 교환한 후 L의 첫 위치를 오른쪽으로 하나 이동하는 작업을 반복함으로써 정렬을 수행한다.
- **실습 #1**: 선택정렬에서 첫 위치 자료가 최소값인 경우에는 교환이 불필요하다. 아래 selectionSort 함수에 이를 반영하시오.
- **실습 #2**: 배열 n과 n의 크기보다 작은 정수 k에 대해 kSmallest(n, k);를 호출하면 배열 n 내 가장 작은 k개 정수들이 배열 n의 인덱스 0 위치부터 k-1 위치까지 저장된다고 할 때, private static void kSmallest(int v[], int k)를 새롭게 정의하시오.

```
public class Test {
	public static void main(String[] args) {
		int	v[]={5,8,1,9,3,5,1,5};
		selectionSort(v);
		System.out.println(Arrays.toString(v));
	}
	private static void selectionSort(int[] v) {
		for (int i = 0; i < v.length-1; i++) {
			int	minPos=i;
			for (int j = i+1; j < v.length; j++) {
				if(v[minPos]>v[j]) minPos=j;
			}
			int	x=v[i]; v[i]=v[minPos]; v[minPos]=x;
		}
	}
}
```

4. (삽입정렬, insertion sort) 다음은 정수 배열을 삽입정렬하는 코드 예이다. 아래 코드를 입력하고 실행하면서 삽입정렬 구현법을 익히시오. (Reference: https://en.wikipedia.org/wiki/Insertion_sort, CC-BY-SA)

- 삽입정렬은 정렬되지 않은 리스트에서 추출된 자료를 정렬된 리스트 내 올바른 위치에 삽입하는 과정을 반복함으로써 정렬을 수행한다.
- 아래 코드에서는 정렬되지 않은 각 자료의 삽입 위치를 찾기 위해 정렬된 자료의 끝 위치 자료부터 크기를 비교하면서 교환하는 작업을 반복한다.
- 최초 배열이 거의 정렬되어 있거나 배열의 크기가 크지 않은 경우 효율적 정렬 방법이다.

```
public class Test {
	public static void main(String[] args) {
		int	v[]={5,8,1,9,3,5,1,5};
		insertionSort(v);
		System.out.println(Arrays.toString(v));
	}
	private static void insertionSort(int[] v) {
		for (int i = 1; i < v.length; i++) {
			for (int j = i; j>0 && v[j-1]>v[j]; j--) {
				int	x=v[j]; v[j]=v[j-1]; v[j-1]=x;
			}
		}
	}
}
```

5. (**실습**: 정렬된 두 배열을 합병) 아래 mergeArray()는 두 정렬된 배열 n, m을 하나의 정렬된 배열 v에 전체적으로 정렬을 유지하면서 합병하는 메소드이다. 이 코드가 정상 동작하도록 아래 구현 방법을 참고하여 mergeArray() 메소드를 완성하시오.

- mergeArray(int n[], int m[]) 구현방법: 두 배열 n, m 전체를 담을 수 있는 새로운 배열 v를 만든다. n[0]과 m[0]을 비교한다. n[0]<m[0]인 경우 n[0]을 v[0]에 대입한다. n[0]은 처리되었으므로 n[1]과 m[0]을 비교한다. n[1]>m[0]인 경우 m[0]을 v[1]에 대입한다. n[1], m[1]을 비교한다. n[1]>m[1]인 경우 m[1]을 v[2]에 대입한다. 이후 유사한 방식으로 n[1], m[2]를 비교한다.
- 실행결과: [1, 2, 3, 4, 5, 6, 7, 7, 8, 10]

```
public class Test {
	public static void main(String[] args) {
		int	n[]={1,3,5,7,8};
		int	m[]={2,4,6,7,10};
		int	v[]=mergeArray(n, m);
		System.out.println(Arrays.toString(v));
	}
	private static int[] mergeArray(int[] n, int[] m) {

	}
}
```

6. (한 배열 내 정렬된 두 부분 배열들을 합병) 다음은 하나의 배열 내 정렬된 부분 배열들을 합병하여 전체 정렬된 배열로 변환하는 코드이다. 아래 코드에서 합병 전 v의 두 부분 배열 v[0..4], v[5..9]은 [1,3,5,7,8], [2,4,6,7,10]으로 각각 정렬된 상태이나 v 전체는 정렬되어 있지 않다.

- 실행결과: [1, 2, 3, 4, 5, 6, 7, 7, 8, 10]

```
public class Test {
	public static void main(String[] args) {
		int	v[]={1,3,5,7,8,2,4,6,7,10};
		mergeArray(v, 0, 4, 5, 9); // v의 부분 배열들 v[0..4], v[5..9]이 정렬된 상태임
		System.out.println(Arrays.toString(v));
	}
	private static void mergeArray(int[] v, int left1, int right1, int left2, int right2) {
		int	w[]=new int[right2-left1+1];
		int	i=left1, j=left2, k=0;
		while(i<=right1 && j<=right2) w[k++]=(v[i]<v[j])? v[i++] : v[j++];
		while(i<=right1) w[k++]=v[i++];
		while(j<=right2) w[k++]=v[j++];
		System.arraycopy(w, 0, v, left1, w.length);
	}
}
```

7. (합병정렬, merge sort) 다음은 정수 배열을 합병정렬하는 코드 예이다. 아래 코드를 입력하고 실행하면서 합병정렬 구현법을 익히시오. (Reference: https://en.wikipedia.org/wiki/Merge_sort, CC-BY-SA)

```
public class Test {
	public static void main(String[] args) {
		int	v[]={8,5,9,1,5,3,5,1};
		mergeSort(v, 0, v.length-1);
		System.out.println(Arrays.toString(v));
	}
	private static void mergeSort(int[] v, int left, int right) {
		if(left==right) return;
		int	m=(left+right)/2;
		mergeSort(v, left, m);
		mergeSort(v, m+1, right);
		mergeArray(v, left, m, m+1, right);
	}
	private static void mergeArray(int[] v, int left1, int right1, int left2, int right2) {
		int	w[]=new int[right2-left1+1];
		int	i=left1, j=left2, k=0;
		while(i<=right1 && j<=right2) w[k++]=(v[i]<v[j])? v[i++] : v[j++];
		while(i<=right1) w[k++]=v[i++];
		while(j<=right2) w[k++]=v[j++];
		System.arraycopy(w, 0, v, left1, w.length);
		System.out.println(Arrays.toString(v));
	}
}
```

8. (**실습**: 정수 값 정렬, 최소/최대값 인지, 중복데이터) 정수 배열 n에 중복 허용 0 ~ 100 범위의 임의 값들이 저장되어 있다. 아래 코드는 함수 sort()를 호출하여 배열 n을 정렬한 후 출력한다고 한다. 비교 기반 정렬을 사용하지 않으면서 배열 n을 정렬하는 함수 sort()를 아래 구현방법을 참고하여 완성하시오. 이 sort() 함수의 시간복잡도는 얼마인가?

   - 구현 방법: 배열 n에 저장된 정수 값들의 범위를 0 ~ 100으로 가정하고, 0 ~ 100의 각 값에 대응하는 인덱스를 갖는 크기 101의 새로운 배열 count를 생성하고 0으로 초기화한다. 배열 n을 순차탐색하면서 배열 내 각 값 v에 대해 count[v]의 값을 1씩 증가시킨다. 이후 count[v]에는 배열 n 내에서 발견되는 값 v의 총 출현횟수가 저장되어 있다. 이후 배열 count를 순차탐색하면서 0보다 큰 값을 갖는 count[v]에 대해 count[v]에 저장된 횟수만큼 값 v를 배열 n에 순차 저장한다.

   배열 n => [4, 1, 4, 4, 1]
   배열 count => [0, 2, 0, 0, 3] => count[4]의 값 3은 배열 n에서 4가 3회 출현했음을 의미
   배열 n에 1을 2회, 4를 3회 순차 저장 => [1, 1, 4, 4, 4]

```
public class Test {
	public static void main(String[] args) {
		int	n[]={87,95,53,77,100,95,14};
		sort(n);
		System.out.println(Arrays.toString(n));
	}
	private static void sort(int[] n) {

	}
}
```

9. (**실습**: 리스트 정렬 여부 판단) 다음 코드는 배열 n 내 정수들이 오름차순 정렬된 경우 true를 그렇지 않은 경우 false를 출력한다고 한다. 아래 코드를 완성하시오.

   - 구현 방법: 배열 내 각 인접 값들(예: <n[0],n[1]>, <n[1], n[2]>, ...)이 정렬(예: n[0]<=n[1]?)되어 있는지 검사한다.

```
public class Test {
	public static void main(String[] args) {
		int	n[]={1,2,3,4,5};
		System.out.println(sorted(n));
	}
	private static boolean sorted(int[] n) {

	}
}
```

References

- C로 쓴 자료구조론 (Fundamentals of Data Structures in C, Horowitz et al.). 이석호 역. 사이텍미디어. 1993.
- 쉽게 배우는 알고리즘: 관계 중심의 사고법. 문병로. 한빛아카데미. 2013.
- C언어로 쉽게 풀어 쓴 자료구조. 천인국 외 2인. 생능출판사. 2017.
- 김윤명. (2008). 뇌를 자극하는 Java 프로그래밍. 한빛미디어.
- 남궁성. 자바의 정석. 도우출판.
- 김윤명. (2010). 뇌를 자극하는 JSP & Servlet. 한빛미디어.