EXERCISE 1: Sorting

The leftmost column contains an array of 24 integers to be sorted; the rightmost column contains the integers in sorted order; the other columns are the contents of the array at some intermediate step during one of the five sorting algorithms listed below. Match each algorithm by writing its number in the box under the corresponding column. Use each number once.

63	44	11	19	11	81	11
21	21	19	21	19	79	19
19	19	21	32	21	63	21
32	32	25	45	29	60	25
45	45	29	60	31	71	29
60	60	31	63	32	29	31
31	31	32	11	44	48	32
79	25	44	31	45	45	44
48	48	45	48	48	52	45
11	11	48	71	60	50	48
71	50	50	79	63	67	50
88	52	52	88	71	21	52
29	29	63	29	79	25	60
99	63	99	99	88	31	63
89	89	89	89	89	19	67
44	99	79	44	99	44	71
86	86	86	86	86	11	79
52	88	88	52	52	32	81
92	92	92	92	92	86	86
50	71	71	50	50	88	88
25	79	60	25	25	89	89
67	67	67	67	67	92	92
93	93	93	93	93	93	93
81	81	81	81	81	99	99
0						6

(0) Original array.

(1) Selection sort.

(2) Insertion Sort.

(3) Mergesort.

(4) Heapsort.

(5) Quicksort (*standard, no shuffle*) .

(6) Sorted.

EXERCISE 2: Hashing & **Binary Search Trees**

Consider the following Symbol Table operation:

> `equals(Object other)`**:** Check if this symbol table has exactly the *same keys* as `other`, regardless of the order of the keys and the values they map to.

For each of the following implementations, give the order of growth of the best- and worst-case running times in symbol tables containing n key-value pairs each. Briefly describe the algorithm of each operation.

	Algorithm	Best case	Worst case
Binary Search Tree			
Red-Black Binary Search Tree			
Hash Table with Chaining (assuming uniform hashing)			
Hash Table with Chaining (*not* assuming uniform hashing)			

EXERCISE 3: Data Structure & **Algorithm Design**

A Point is an object consisting of an *x*- and a *y*-coordinate. Your goal is to maintain a collection of Points that supports the following operations:

- ***Add*** a Point to the collection.
- ***Return*** the Point with the ***lowest x***-coordinate.
- ***Return*** the Point with the ***lowest y***-coordinate.
- ***Delete*** the Point with the ***lowest x***-coordinate.
- ***Delete*** the Point with the ***lowest y***-coordinate.

If there are multiple Points with the same *x*- or *y*-coordinate, you may choose among them arbitrarily.

Performance Requirements:

- Any sequence of n invocations of the supported operations (in any order), starting from an empty collection, must complete in time proportional to $n \log n$ in the worst case.
- Returning the Point with the lowest *x*- or *y*-coordinate must take $constant$ time.

You may make any standard technical assumptions that we have seen in this course.

(a) Describe the ***data structures*** you would use. Specifically, for any new data structures you need, write the class declaration. For any data structures from lectures/textbook that you would use, succinctly describe how you would use them and what modifications are needed (if any).

(b) Give a concise English description of your algorithm for ***adding*** a Point to the collection. Feel free to use some pseudocode if you think it will improve clarity.

(c) Give a concise English description of your algorithm for ***returning*** the Point with the lowest *x*- or *y*-coordinate. Feel free to use some pseudocode if you think it will improve clarity.

(d) Give a concise English description of your algorithm for ***deleting*** the Point with the lowest *x*- or *y*-coordinate. Feel free to use some pseudocode if you think it will improve clarity.