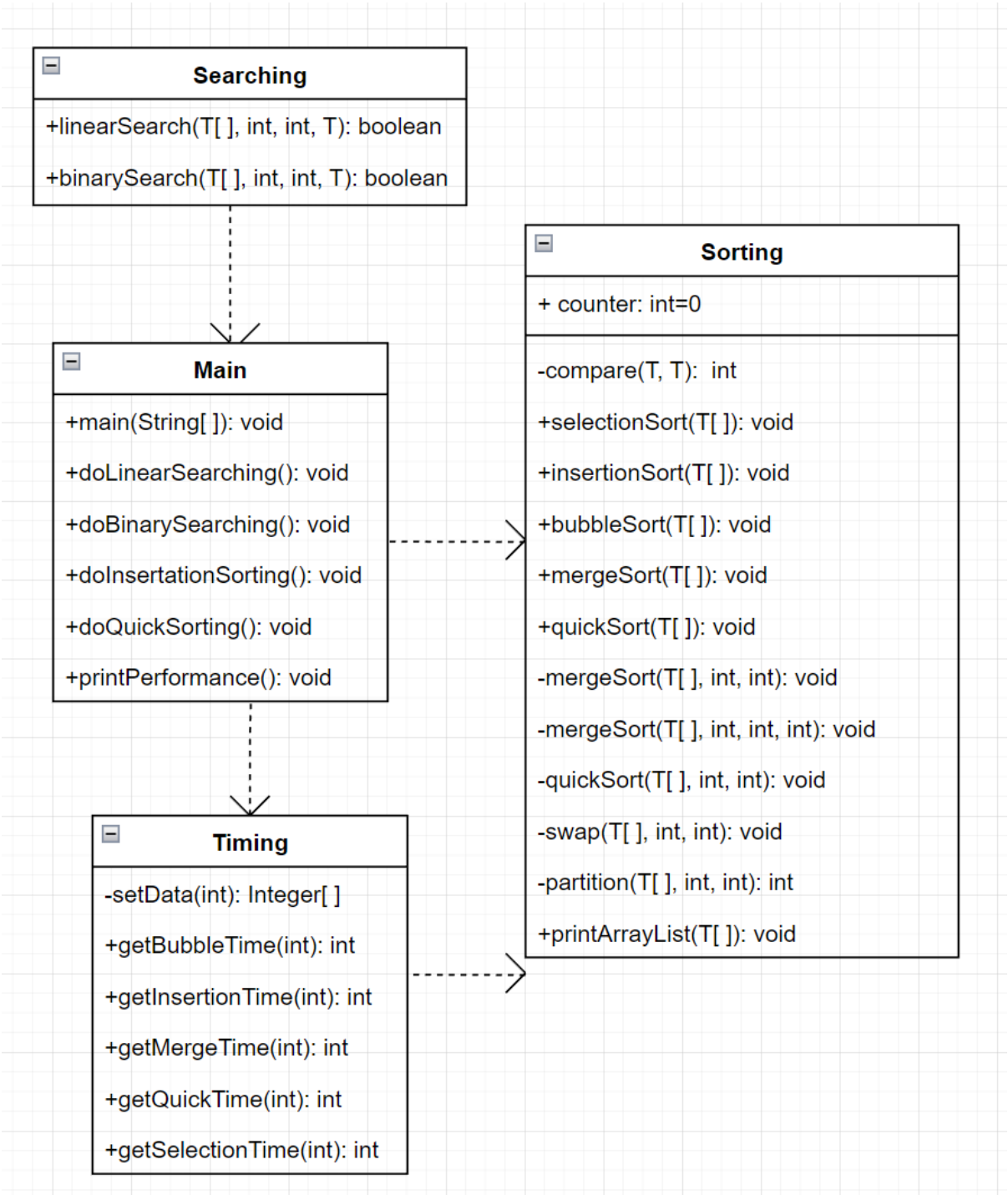
- **Design solution/UML diagrams:**



- **Testing:**

1) **Linear Searching (worth of 1p)**

The searched list contains items from 0 to 9.

You search some of them, the application says the given value was found.

You search a value not in a list, the application says the given value was not found.

Example:

Choice: 1

value: 1 ----------> Found

value: 10 ----------> Not found

```
Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:  1
In the list are values 0, ..., 9; which value woudld you like to search with linear search? 3

Found

Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:  1
In the list are values 0, ..., 9; which value woudld you like to search with linear search? 10

Not found

Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:
```

2) Binary Searching (worth of 2p)

The searched list contains items from 0 to 9.

You search some of them, the application says the given value was found.

You search a value not in a list, the application says the given value was not found.

Example:

Choice: 2

value: 1 ----------> Found

value: 10 ----------> Not found

```
Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:  2
In the list are values 0, ..., 9; which value woudld you like to search with binary search? 5

Found

Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:  2
In the list are values 0, ..., 9; which value woudld you like to search with binary search? 11

Not found

Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:
```

3) $O(n^2)$ Type of Sorting (worth of 3p)

It produces a random set of ten integers (-99~99) and gives that set to a sort algorithm having $O(n^2)$ performance requirement.

Here I choose insertion sort.

Data is output before and after sorting.

```
Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:  3

Data set before insertion sorting:
92 68 -6 11 78 70 0 -17 -39 11

Data set after insertion sorting:
-39 -17 -6 0 11 11 68 70 78 92

Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:  3

Data set before insertion sorting:
-17 -90 -86 3 43 -99 23 44 38 -97

Data set after insertion sorting:
-99 -97 -90 -86 -17 3 23 38 43 44

Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:
```

4) O(n*log(n)) Type of Sorting (worth of 4p)

It produces a random set of ten integers (-99~99) and gives that set to a sort algorithm having O(n*log(n)) performance requirement.

Here I choose quick sort.

Data is output before and after sorting.

```
Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:  4

Data set before quicksort:
13 -23 -95 -65 26 59 23 75 -89 32

Data set after quicksort:
-95 -89 -65 -23 13 23 26 32 59 75

Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:  4

Data set before quicksort:
-60 -39 56 56 -49 -72 92 33 -89 42

Data set after quicksort:
-89 -72 -60 -49 -39 33 42 56 56 92

Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:
```

5) Sorting performance (worth of 5p)

It sorts items of sizes 1000, 2000, ⋯ , 10 000 (in random order) and monitor how many milliseconds went to sorting this data set and how many comparisons between two data items of all sorted data were done during sorting.

Two sorting algorithms of type O(n2)**(bubbleSort, seletionSort,insertionSort)** are monitored and two of type O(n*log(n))**(mergeSort, quickSort)** are handled here.

```
Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:  5
                                1000      2000      3000      4000      5000      6000      7000      8000      9000      10000
bubbleSort,random,comparisons   499500    1999000   4498500   7998000   12497500  17997000  24496500  31996000  40495500  49995000
bubbleSort,random,ms            16        12        41        55        85        127       164       228       278       345
insertionSort,random,comparisons 249980   991132    2282470   4057104   6195666   8984106   12442104  16013174  20601366  24834086
insertionSort,random,ms         4         17        8         9         13        19        26        34        43        93
mergeSort,random,comparisons    8693      19439     30915     42823     55218     67745     80653     93717     107007    120422
mergeSort,random,ms             2         9         10        12        34        10        17        14        25        57
quickSort,random,comparisons    15560     33638     55769     78346     100194    120975    145222    168523    196215    218348
quickSort,random,ms             0         1         1         1         0         1         1         1         1         1
selectionSort,random,comparisons 499500   1999000   4498500   7998000   12497500  17997000  24496500  31996000  40495500  49995000
selectionSort,random,ms         6         4         8         13        21        29        40        54        67        84

Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:  5
                                1000      2000      3000      4000      5000      6000      7000      8000      9000      10000
bubbleSort,random,comparisons   499500    1999000   4498500   7998000   12497500  17997000  24496500  31996000  40495500  49995000
bubbleSort,random,ms            3         13        23        45        71        103       143       188       270       294
insertionSort,random,comparisons 251694   1002014   2260001   3992669   6281242   9022621   12196626  16087077  20472112  25243361
insertionSort,random,ms         1         2         5         9         13        19        28        34        43        54
mergeSort,random,comparisons    8680      19396     30883     42871     55138     67827     80731     93645     106940    120436
mergeSort,random,ms             1         5         13        22        23        11        15        12        23        30
quickSort,random,comparisons    15172     34625     56233     78856     97105     119092    141961    173372    199826    213184
quickSort,random,ms             0         0         0         0         1         1         0         1         1         1
selectionSort,random,comparisons 499500   1999000   4498500   7998000   12497500  17997000  24496500  31996000  40495500  49995000
selectionSort,random,ms         1         4         7         14        20        30        40        51        66        83

Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:
```

6) Other requirements

The menu is to be able to rerun any function as many times as wanted and only after choosing Quit you exit the application.

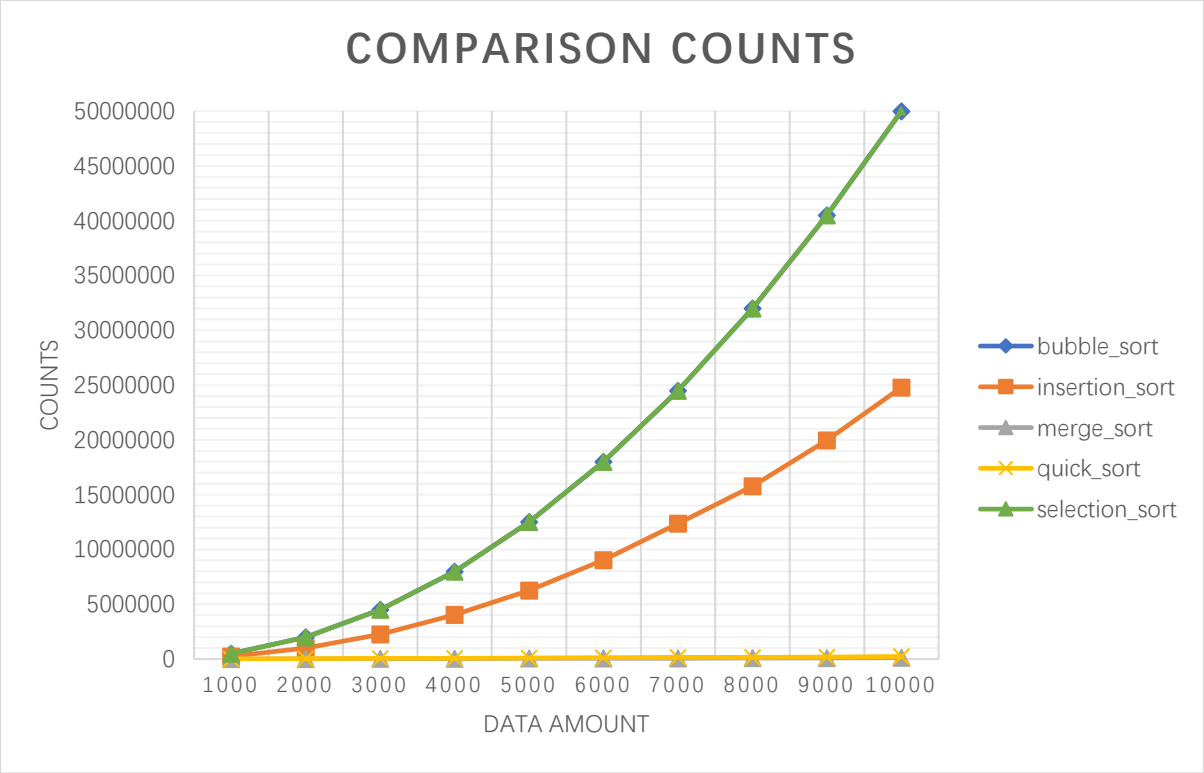
The consequent runs must not interfere each other.

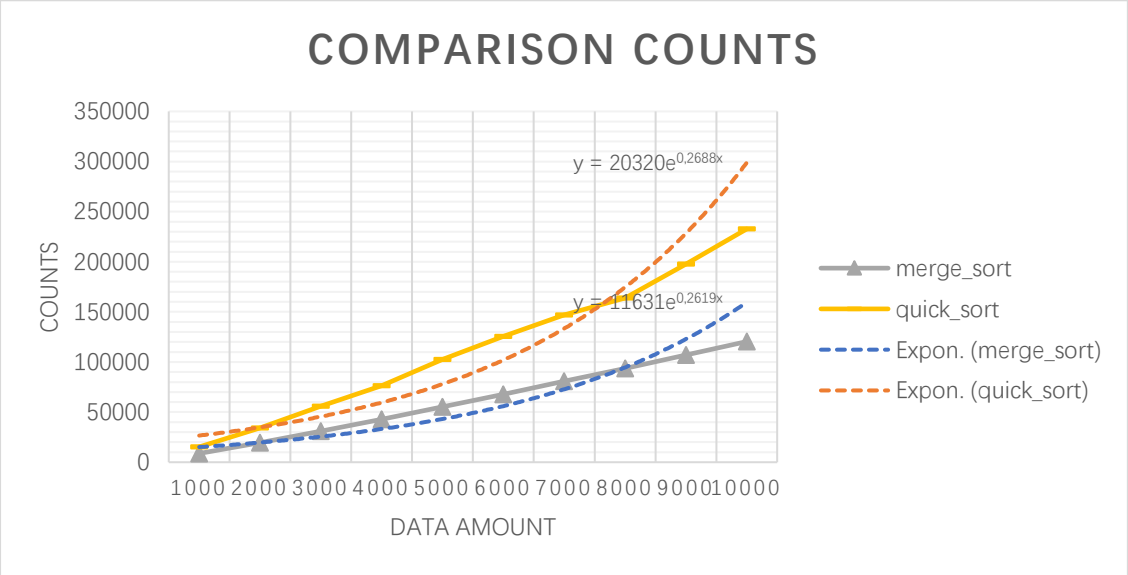
```
Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:  A
Invalid input! Please try again...

Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:  6
Invalid input! Please try again...

Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:  q
Quiting...

Process finished with exit code 0
```

```
Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:  x
Invalid input! Please try again...

Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:  oji
Invalid input! Please try again...

Menu of Searching and Sorting Testbed.
1)  Linear searching
2)  Binary searching
3)  O(n^2) type of sorting
4)  O(n*log(n)) type of sorting
5)  Sorting performance

q/Q) Quit

Your choice:  Q
Quiting...

Process finished with exit code 0
```

- Excel performance diagram:


Time in milliseconds
Milliseconds
350
300
250
200
150
100
50
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Data amount
bubble_sort
insertion_sort
merge_sort
quick_sort
selection_sort


COMPARISON COUNTS
COUNTS
50000000
45000000
40000000
35000000
30000000
25000000
20000000
15000000
10000000
5000000
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
DATA AMOUNT
bubble_sort
insertion_sort
merge_sort
quick_sort
selection_sort



- **Work hour bookkeeping:**

Date	Hours	Doing
16th November	20min	Getting familiar with project requirements.
20th November	5min 15min 1h	Created project Get menu done using while loop and switch case. Learning searching and sorting on slide09.
21st November	5min 30min 40min 1h 30min 10min 1h	Copied Searching.java and Sorting.java to my project. Get function 1&2 done using ternary operator. Get random data set done, added void printArrayList() to Sorting.java, then get function 3&4 done. Working on function 5, get comparison bubbleSort and selectionSort right using System.currentTimeMillis(), but get stuck on the others. Optimizing project structure by separating some code blocks as dependent method. Formatting menu structure using \n. Get function 5 printing formatted, using printf and multidimensional array, after \t failed.
22nd November	1h 1.5h 1h	Trying to get function 5 right, modify code back and forth, resulted in nothing. Watching recording. Learning the difference between each sorting algorithm online.
26th Novemeber	1h	Get elapsed time monitor done by creating a method of some code portion
28th Novemeber	1h	Get comparison counts monitor done: created a class holding the increment the counter variable(public static) for each sorting algorithm and compareTo method, then use that method in corresponding sorting algorithm.
29th November	1.5h	Watched recording, used one counter instead of five ones for each sorting algorithm by resetting the counter to 0 in each algorithm, Optimizing project structure
30th Novermeber	2h 1h	Documented the report. Made an Excel chart of milliseconds/comparison counts as a function of data

		set size.
12th December	30min	Checking all requirements are meet and submit to Moodle.

- **How to test my program:**

Step 1: Download the file JingjingYang_5p.zip from Moodle, go ahead and unzip that to your working directory.

Step 2: Launch IntelliJ IDEA (or another Java development toolkit you are using), configure project structure SDK 17. From the main menu, select File | Open, and then navigate to the unzipped folder searchNSort. Specify whether you want to open the project in a new window or close the current project and reuse the existing window.

Step 3: Test the program by running Main.java.