Program Structures & Algorithms
Fall 2021
Assignment No. 2

- **Task (List down the tasks performed in the Assignment)**

- Part1

  Implement three methods of a class called Timer. The function to be timed, hereinafter the "target" function, is the Consumer function fRun (or just f) passed in to one or other of the constructors. The generic type T is that of the input to the target function. The first parameter to the first run method signature is the parameter that will, in turn, be passed to target function. In the second signature, supplier will be invoked each time to get a t which is passed to the other run method. The second parameter to the run function (m) is the number of times the target function will be called. The return value from run is the average number of milliseconds taken for each run of the target function.

```
public <T, U> double repeat(int n, Supplier<T> supplier, Function<T, U> function, UnaryOperator<T> preFunction, Consumer<U>
    logger.trace("repeat: with " + n + " runs");
    // TO BE IMPLEMENTED: note that the timer is running when this method is called and should still be running when it ret
    //return 0;
    for(int i = 0; i < n; i++) {
        pause();
        T t = supplier.get();
        if(preFunction!=null) {
            t = preFunction.apply(t);
        }
        resume();
        U u = function.apply(t);
        pauseAndLap();
        if(postFunction!=null) {
            postFunction.accept(u);
        }
        resume();
    }
    pause();
    return meanLapTime();
}
```

- Part2

  Implement InsertionSort (in the InsertionSort class) by simply looking up the insertion code used by Arrays.sort.

```
public void sort(X[] xs, int from, int to) {
    final Helper<X> helper = getHelper();

    // TO BE IMPLEMENTED
    for(int i = from + 1; i < from + to; i++){
        for(int j = i; j > from && helper.compare(xs, j - 1, j) > 0; j-- ){
            helper.swap(xs, j - 1, j);
        }
    }
```

```
Markers Properties Servers Data Source Explorer Snippets Problems Console JUnit
<terminated> InsertionSortTest [JUnit] C:\JDK\bin\javaw.exe (2021年9月22日 下午7:28:42)
2021-09-22 19:28:42 DEBUG Config - Config.get(helper, instrument) = true
2021-09-22 19:28:42 DEBUG Config - Config.get(helper, seed) = 0
2021-09-22 19:28:42 DEBUG Config - Config.get(instrumenting, copies) = true
2021-09-22 19:28:42 DEBUG Config - Config.get(instrumenting, swaps) = true
2021-09-22 19:28:42 DEBUG Config - Config.get(instrumenting, compares) = true
2021-09-22 19:28:42 DEBUG Config - Config.get(instrumenting, inversions) = 1
2021-09-22 19:28:42 DEBUG Config - Config.get(instrumenting, fixes) = true
2021-09-22 19:28:42 DEBUG Config - Config.get(instrumenting, hits) = true
2021-09-22 19:28:42 DEBUG Config - Config.get(helper, cutoff) =
Helper for InsertionSort with 4 elements
StatPack {hits: 9,684; copies: 0; inversions: 2,421; swaps: 2,421; fixes: 2,421; compares: 2,519}
StatPack {hits: 19,800; copies: 0; inversions: 4,950; swaps: 4,950; fixes: 4,950; compares: 4,950}
```

- Part3

Implement a main program (or you could do it via your own unit tests) to actually run the following benchmarks: measure the running times of this sort, using four different initial array ordering situations: random, ordered, partially-ordered and reverse-ordered. I suggest that your arrays to be sorted are of type Integer. Use the doubling method for choosing n and test for at least five values of n. Draw any conclusions from your observations regarding the order of growth.

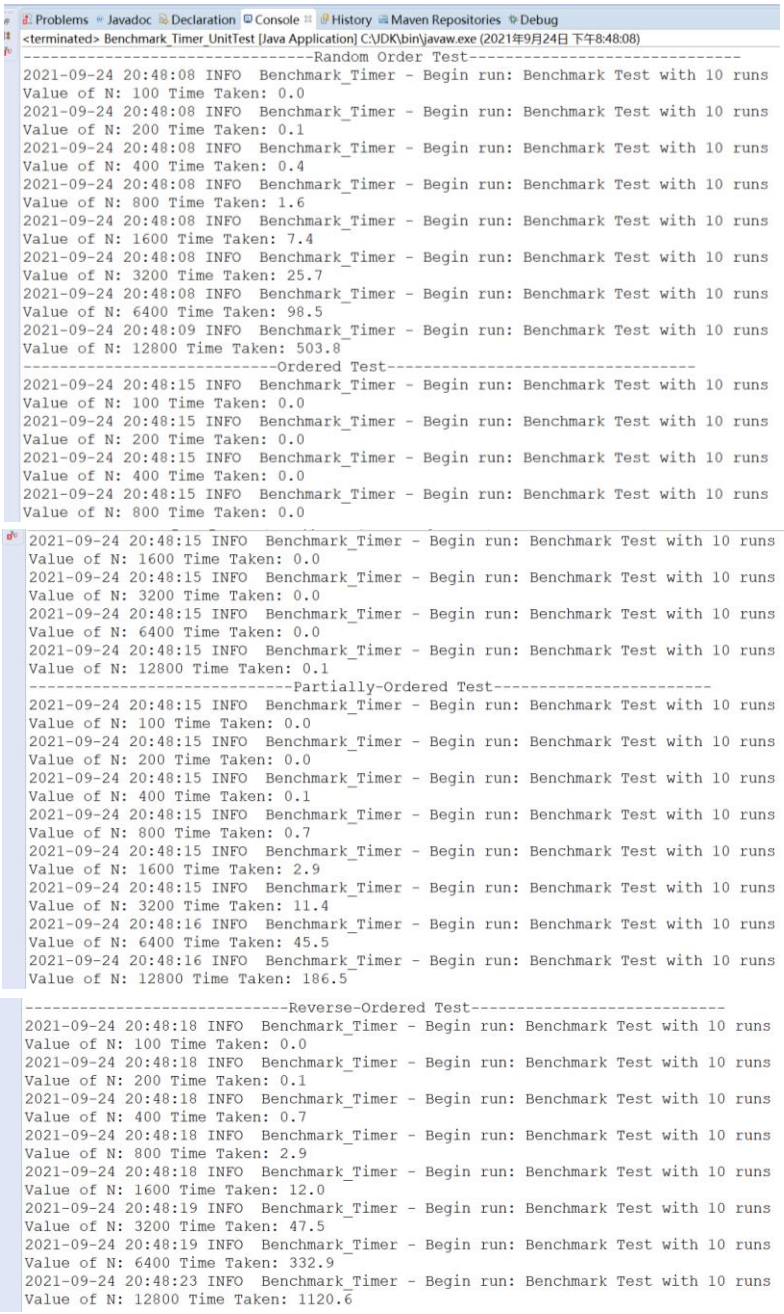
```
    public static void main(String[] args) {
        InsertionSort ins_sort = new InsertionSort();
        Benchmark_Timer<Integer[]> bTimer = new Benchmark_Timer<>("Benchmark Test", null, (x) -> ir

        System.out.println("------------------------------Random Order Test----------------------
        //random array
        for(int i = 100; i < 20000; i = i*2) {
            int j = i;
            Supplier<Integer[]> supplier = new Supplier<Integer[]>() {
                @Override
                public Integer[] get() {
                    Random random = new Random();
                    Integer[] arr = new Integer[j];
                    for(int k = 0; k < j; k++) {
                        arr[k] = random.nextInt(j);
                    }
                    return arr;
                }
            };
            double time = bTimer.runFromSupplier(supplier, 10);
            System.out.println("Value of N: " + i + " Time Taken: " + time);
        }
for(int i = 100; i < 20000; i = i*2) {
    int j = i;
    Supplier<Integer[]> supplier = new Supplier<Integer[]>() {
        @Override
        public Integer[] get() {
            Random random = new Random();
            Integer[] arr = new Integer[j];
            for(int k = 0; k < j; k++) {
                arr[k] = random.nextInt(j);
            }
            Arrays.sort(arr);
            int rearrange = (int) (0.5*j);
            for(int i = 0; i < rearrange; i++) {
                int index = random.nextInt(j);
                arr[index] = random.nextInt(j);
            }
            return arr;
        }
    };
    double time = bTimer.runFromSupplier(supplier, 10);
    System.out.println("Value of N: " + i + " Time Taken: " + time);
for(int i = 100; i < 20000; i = i*2) {
    int j = i;
    Supplier<Integer[]> supplier = new Supplier<Integer[]>() {
        @Override
        public Integer[] get() {
            Random random = new Random();
            Integer[] arr = new Integer[j];
            for(int k = 0; k < j; k++) {
                arr[k] = random.nextInt(j);
            }
            Arrays.sort(arr);
            return arr;
        }
    };
    double time = bTimer.runFromSupplier(supplier, 10);
    System.out.println("Value of N: " + i + " Time Taken: " + time);
}
    for(int i = 100; i < 20000; i = i*2) {
        int j = i;
        Supplier<Integer[]> supplier = new Supplier<Integer[]>() {
            @Override
            public Integer[] get() {
                Random random = new Random();
                Integer[] arr = new Integer[j];
                for(int k = 0; k < j; k++) {
                    arr[k] = random.nextInt(j);
                }
                Arrays.sort(arr, Collections.reverseOrder());
                return arr;
            }
        };
        double time = bTimer.runFromSupplier(supplier, 10);
        System.out.println("Value of N: " + i + " Time Taken: " + time);
    }
}
```

- **Relationship Conclusion: (For ex : z = a * b)**

  In ordered cases, the array is ordered, the insertion sort runs in $\mathbf{O}(n)$ times.
  The order of run time of insertion sort for different order situation in arrays are:
  Ordered < Partially Ordered < Randomly Ordered < Reverse Ordered

- **Evidence to support the conclusion:**

1. **Output (Snapshot of Code output in the terminal)**

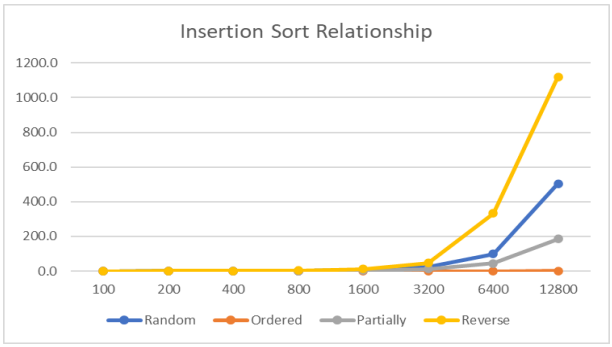


```
Problems  Javadoc  Declaration  Console  History  Maven Repositories  Debug
<terminated> Benchmark_Timer_UnitTest [Java Application] C:\JDK\bin\javaw.exe (2021年9月24日 下午8:48:08)
--------------------------------Random Order Test----------------------------
2021-09-24 20:48:08 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 100 Time Taken: 0.0
2021-09-24 20:48:08 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 200 Time Taken: 0.1
2021-09-24 20:48:08 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 400 Time Taken: 0.4
2021-09-24 20:48:08 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 800 Time Taken: 1.6
2021-09-24 20:48:08 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 1600 Time Taken: 7.4
2021-09-24 20:48:08 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 3200 Time Taken: 25.7
2021-09-24 20:48:08 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 6400 Time Taken: 98.5
2021-09-24 20:48:09 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 12800 Time Taken: 503.8
---------------------------Ordered Test---------------------------------
2021-09-24 20:48:15 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 100 Time Taken: 0.0
2021-09-24 20:48:15 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 200 Time Taken: 0.0
2021-09-24 20:48:15 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 400 Time Taken: 0.0
2021-09-24 20:48:15 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 800 Time Taken: 0.0
2021-09-24 20:48:15 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 1600 Time Taken: 0.0
2021-09-24 20:48:15 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 3200 Time Taken: 0.0
2021-09-24 20:48:15 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 6400 Time Taken: 0.0
2021-09-24 20:48:15 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 12800 Time Taken: 0.1
---------------------------Partially-Ordered Test-----------------------
2021-09-24 20:48:15 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 100 Time Taken: 0.0
2021-09-24 20:48:15 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 200 Time Taken: 0.0
2021-09-24 20:48:15 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 400 Time Taken: 0.1
2021-09-24 20:48:15 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 800 Time Taken: 0.7
2021-09-24 20:48:15 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 1600 Time Taken: 2.9
2021-09-24 20:48:15 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 3200 Time Taken: 11.4
2021-09-24 20:48:16 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 6400 Time Taken: 45.5
2021-09-24 20:48:16 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 12800 Time Taken: 186.5
---------------------------Reverse-Ordered Test--------------------------
2021-09-24 20:48:18 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 100 Time Taken: 0.0
2021-09-24 20:48:18 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 200 Time Taken: 0.1
2021-09-24 20:48:18 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 400 Time Taken: 0.7
2021-09-24 20:48:18 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 800 Time Taken: 2.9
2021-09-24 20:48:18 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 1600 Time Taken: 12.0
2021-09-24 20:48:19 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 3200 Time Taken: 47.5
2021-09-24 20:48:19 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 6400 Time Taken: 332.9
2021-09-24 20:48:23 INFO  Benchmark_Timer - Begin run: Benchmark Test with 10 runs
Value of N: 12800 Time Taken: 1120.6
```

2. Graphical Representation (Observations from experiments should be tabulated and analyzed by plotting graphs(usually in excel) to arrive on the relationship conclusion)

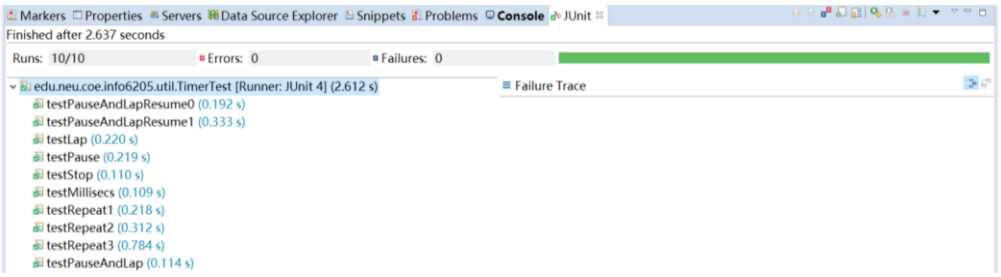
I put all the output into the Excel, then put the running time and eight different values of N to drawn in a broken line diagram, from which can clearly see the speed of these four sorts in running time.

Random	Ordered	Partially	Reverse	N
0.0	0.0	0.0	0.0	100
0.1	0.0	0.0	0.1	200
0.4	0.0	0.1	0.7	400
1.6	0.0	0.7	2.9	800
7.4	0.0	2.9	12.0	1600
25.7	0.0	11.4	47.5	3200
98.5	0.0	45.5	332.9	6400
503.8	0.1	186.5	1120.6	12800

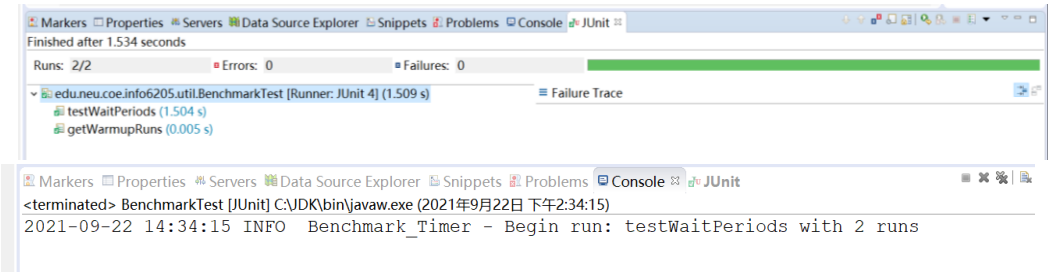


- **Unit tests result:(Snapshot of successful unit test run)**

- TimerTest



- BenchmarkTest



- InsertionSortTest