Lab2 Running Time Survey

YAO ZHAO

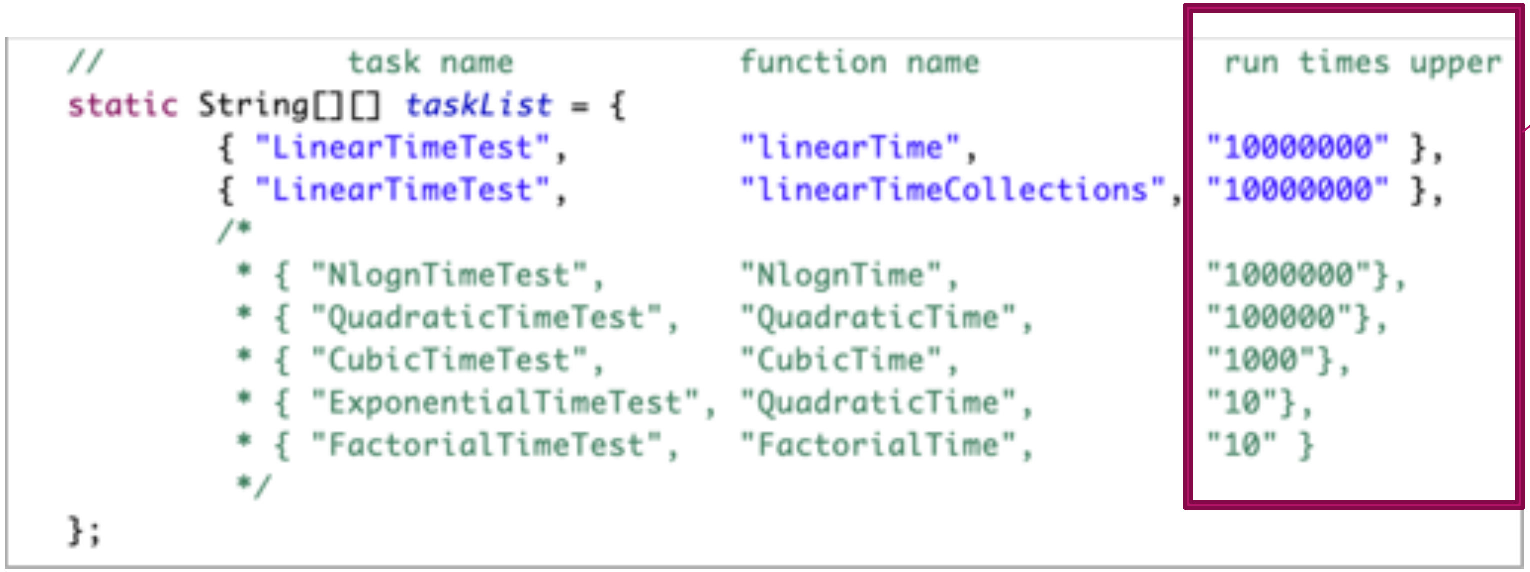
- This week, let’s do a running time survey.
- A simple frame for you to do the running time survey of different algorithms on inputs of increasing size.

`RunningTimeSurvey.java`

How to use?

- You should register your tasks and methods in the taskList

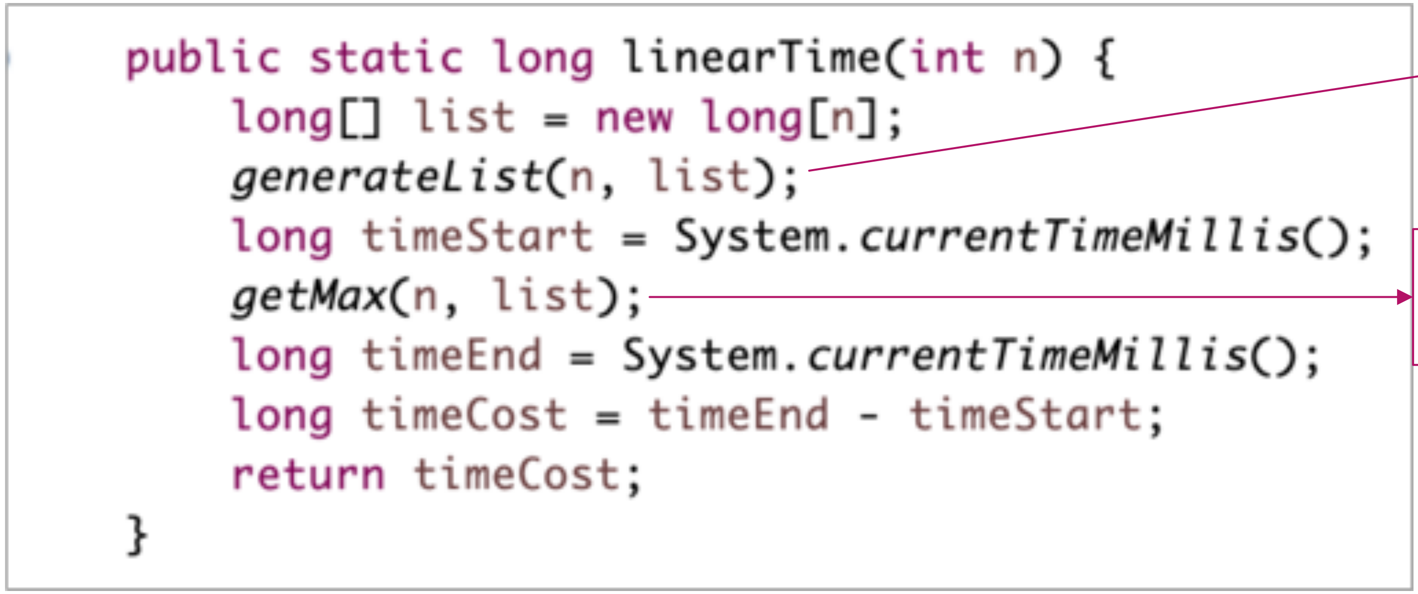
You can change the number according your computer configuration



```
//              task name              function name               run times upper
static String[][] taskList = {
        { "LinearTimeTest",            "linearTime",               "10000000" },
        { "LinearTimeTest",            "linearTimeCollections",    "10000000" },
        /*
         * { "NlognTimeTest",          "NlognTime",                "1000000"},
         * { "QuadraticTimeTest",      "QuadraticTime",            "100000"},
         * { "CubicTimeTest",          "CubicTime",                "1000"},
         * { "ExponentialTimeTest",    "QuadraticTime",            "10"},
         * { "FactorialTimeTest",      "FactorialTime",            "10" }
         */
};
```

LinearTimeTest

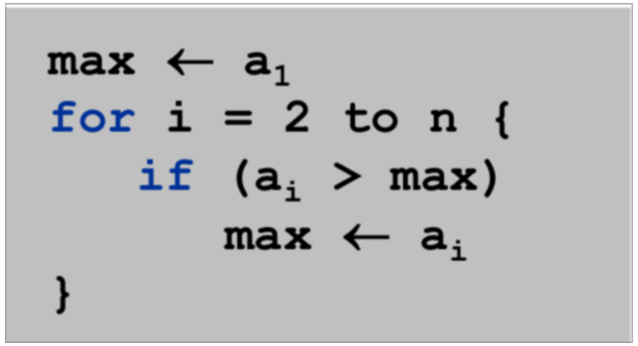
Since “linearTime” is registered for “LinearTimeTest”, you should define a function named linearTime, which looks like the following code:



```
public static long linearTime(int n) {
    long[] list = new long[n];
    generateList(n, list);
    long timeStart = System.currentTimeMillis();
    getMax(n, list);
    long timeEnd = System.currentTimeMillis();
    long timeCost = timeEnd - timeStart;
    return timeCost;
}
```

You can first write a function to generate data for your following algorithm.

Implements a Linear algorithm, for example, computing the maximum.



```
max ← $a_1$
for i = 2 to n {
    if ($a_i$ > max)
        max ← $a_i$
}
```

You can also choose other linear time algorithms.

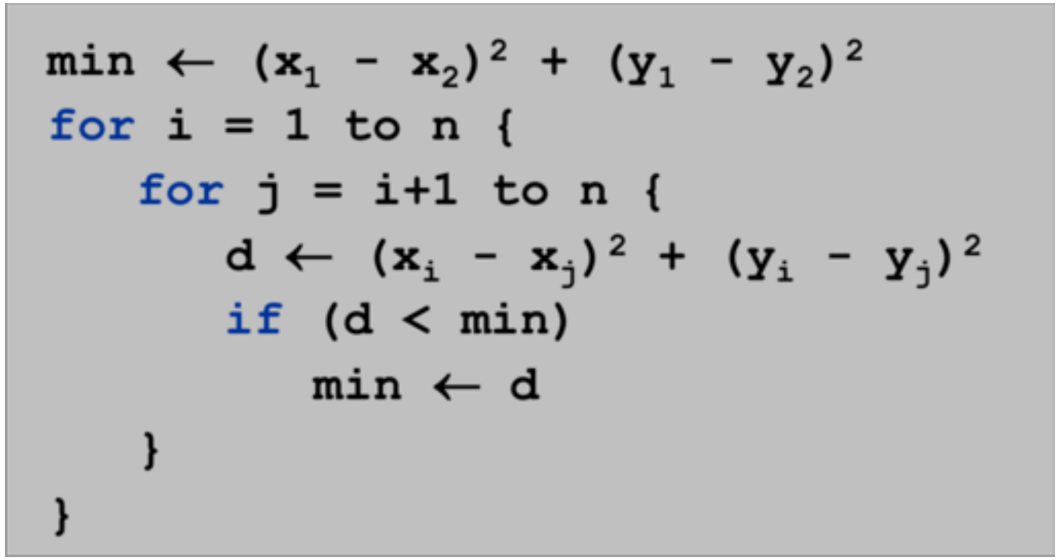
O(n log n) TimeTest

- You should register a new task named “NlognTimeTest”.
- You should register a function named “NlognTime”, the input parameter should be int, the return type should be long.
- You should generate your test data for your algorithm.
- You should implement your algorithm which running time is required, for example, heap sort.

```
public static long NlognTime(int n) {
    //TODO:generate you test input data here
    long timeStart = System.currentTimeMillis();
    //TODO: write a algorithm
    long timeEnd = System.currentTimeMillis();
    long timeCost = timeEnd - timeStart;
    return timeCost;
}
```

QuadraticTimeTest

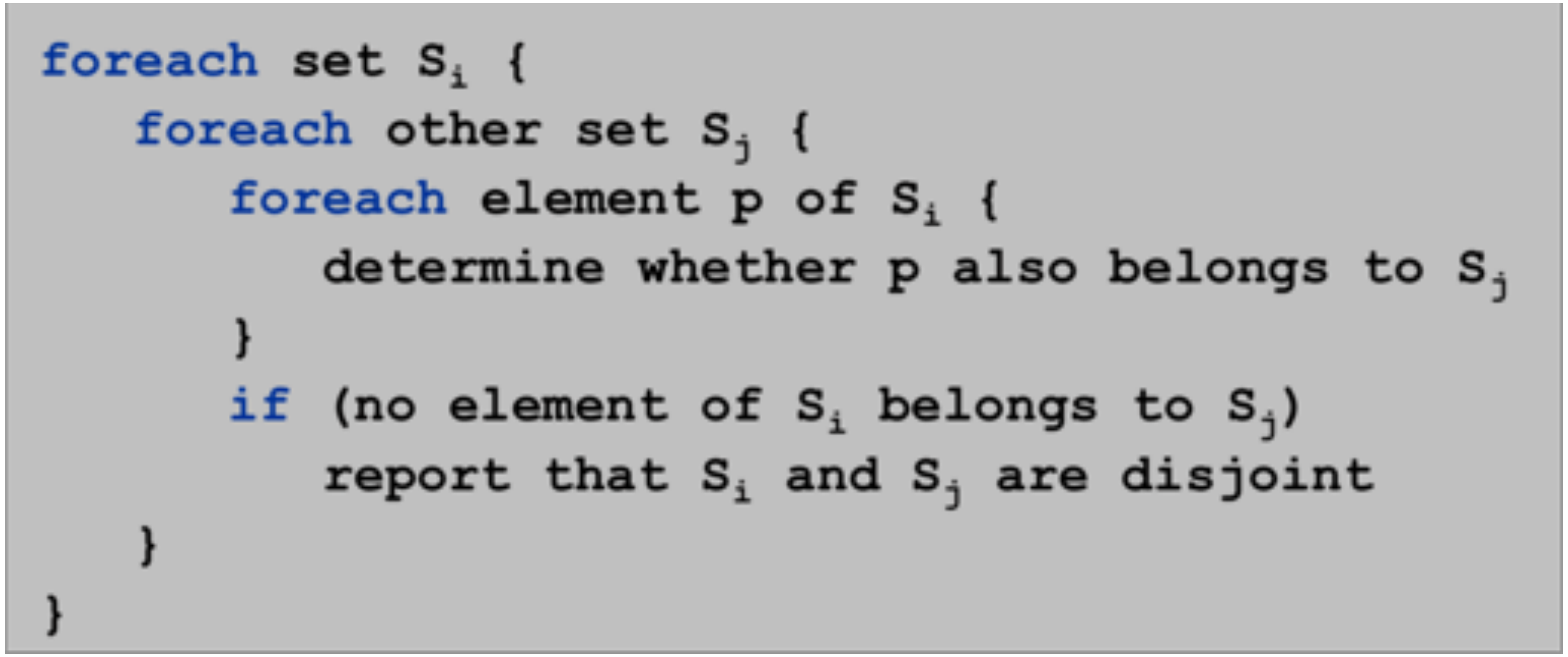
- Optional:
- Closest pair of points. Given a list of n points in the plane (x1, y1), ..., (xn, yn), find the pair that is closest.
- O(n2) solution. Try all pairs of points.



```
min ← $(x_1 - x_2)^2 + (y_1 - y_2)^2$
for i = 1 to n {
    for j = i+1 to n {
        d ← $(x_i - x_j)^2 + (y_i - y_j)^2$
        if (d < min)
            min ← d
    }
}
```

CubicTimeTest

- Optional:
- Set disjointness. Given n sets S1, ..., Sn each of which is a subset of 1, 2, ..., n, is there some pair of these which are disjoint?
  O(n3) solution: For each pairs of sets, determine if they are disjoint.



```
foreach set $S_i$ {
    foreach other set $S_j$ {
        foreach element p of $S_i$ {
            determine whether p also belongs to $S_j$
        }
        if (no element of $S_i$ belongs to $S_j$)
            report that $S_i$ and $S_j$ are disjoint
    }
}
```

ExponentialTimeTest

- Given n bits, enumerate all possible Number.

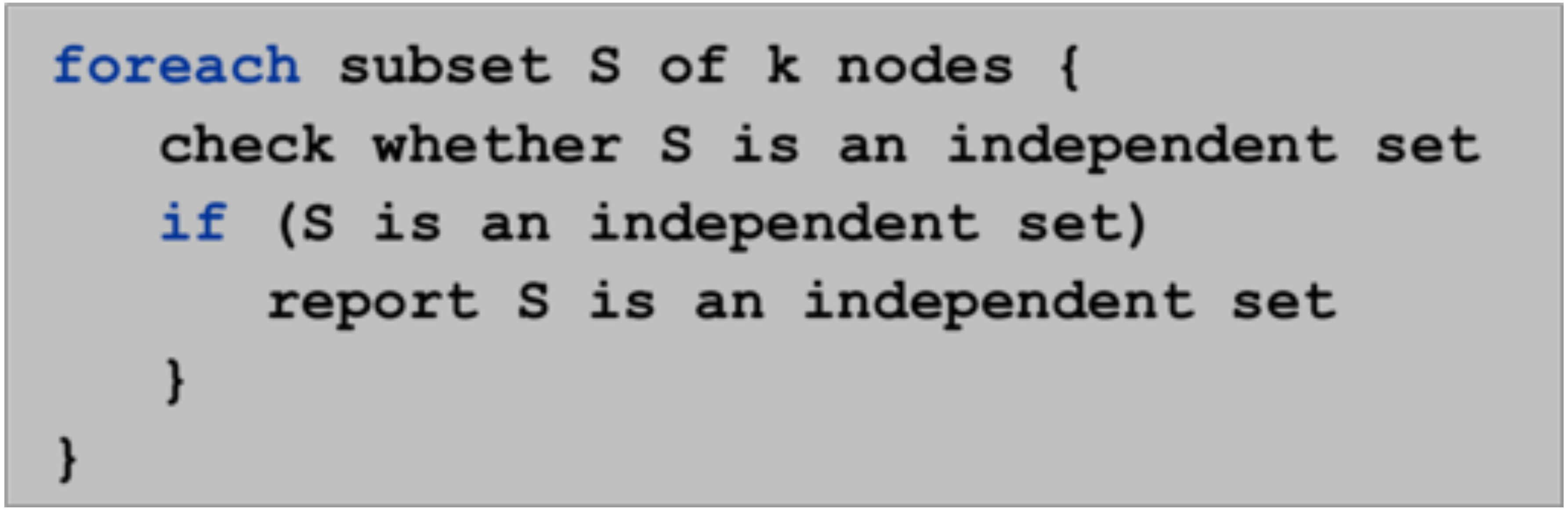
FactorialTimeTest

- Bruce force to compute factorial n

```
Factorial(n) {
    if (n == 1) return 1;
    else {
         sum <- 0;
         for (i = 1 to n) {
            sum <-sum + Factorial(n - 1);
         }
         return sum;
    }
}
```

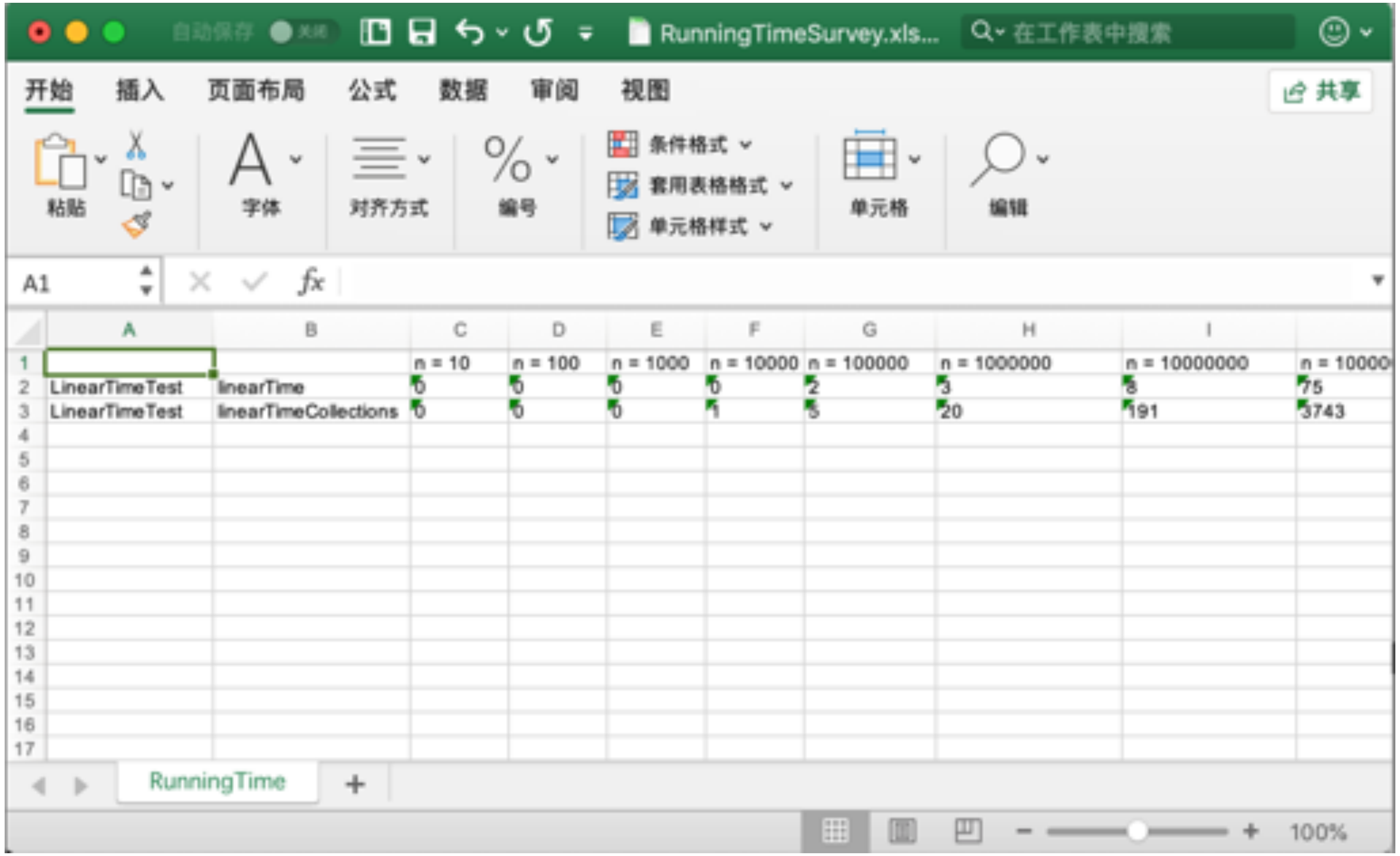
Optional: KPolynomialTimeTest

- Independent set of size k. Given a graph, are there k nodes such that no two are joined by an edge?
- O(nk) solution. Enumerate all subsets of k nodes.



```
foreach subset S of k nodes {
    check whether S is an independent set
    if (S is an independent set)
        report S is an independent set
    }
}
```

A sample run:



	A	B	C	D	E	F	G	H	I	
1			n = 10	n = 100	n = 1000	n = 10000	n = 100000	n = 1000000	n = 10000000	n = 10000
2	LinearTimeTest	linearTime	0	0	0	0	2	3	8	75
3	LinearTimeTest	linearTimeCollections	0	0	0	1	5	20	191	3743