Running Time Survey

YAO ZHAO

Running Time Survey

- This week, let's do a running time survey.
- You will be given a simple frame 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

```
//              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",   "ExponentialTime",         "10"},
         * { "FactorialTimeTest",     "FactorialTime",           "10"}
         */
};
```

You can change the number according to your computer configuration and your time calculation.

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 ← a1
for i = 2 to n {
    if (ai > max)
        max ← ai
}
```

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

ExponentialTimeTest

- Given n bits, enumerate all possible Number.

FactorialTimeTest

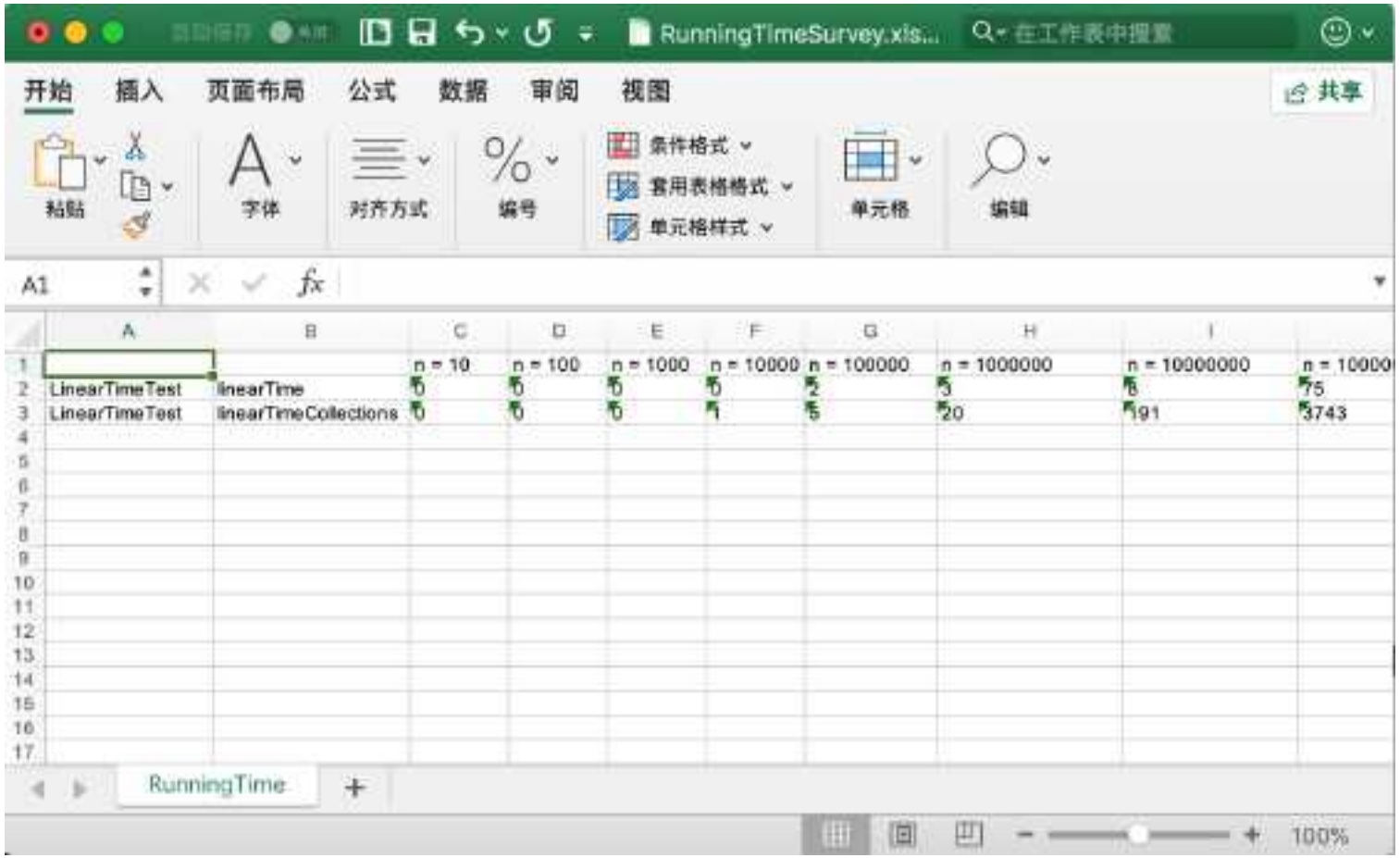
- Bruce force to compute factorial n
- Use addition instead of multiplication

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
    }
}
```

Running result of the frame:



Result example:

		n = 10	n = 100	n = 1000	n = 10000	n = 100000	n = 1000000	n = 10000000
LinearTime	linearTime	0	0	0	0	15	0	0
LinearTime	linearTime(	0	0	0	0	0	38	142
Nlogn Time	NlognTime	0	0	0	8	10	111	
QuadraticT	QuadraticT	0	0	131	684	46994		
CubicTime	CubicTime	0	0	216				

		n = 10	n = 11	n = 12	n = 13	n = 14	n = 15	n = 16	n = 17	n = 18	n = 19	n = 20	n = 21	n = 22	n = 23	n = 24	n = 25	n = 26	n = 27	n = 28	n = 29
Exponentia	Exponentia	0	0	0	0	0	0	0	0	16	0	0	0	22	32	50	150	192	602	670	2385
FactorialTi	FactorialTi	113	1300	22989																	

Submission

Java:
Please submit two files, **RunningTimeSurvey.java** and RunningTimeSurvey.xls, zip these two files into a **.zip** file named "**Name-StudentID-Lab0**" (e.g., **Zhangsan-12011000-Lab0.zip**).

C:
Please submit three files, **running_time_survey.c**, **running_time_survey.h** and **c_result.csv**, zip these three files into a **.zip** file named "**Name-StudentID-Lab0**" (e.g., **Zhangsan-12011000-Lab0.zip**).