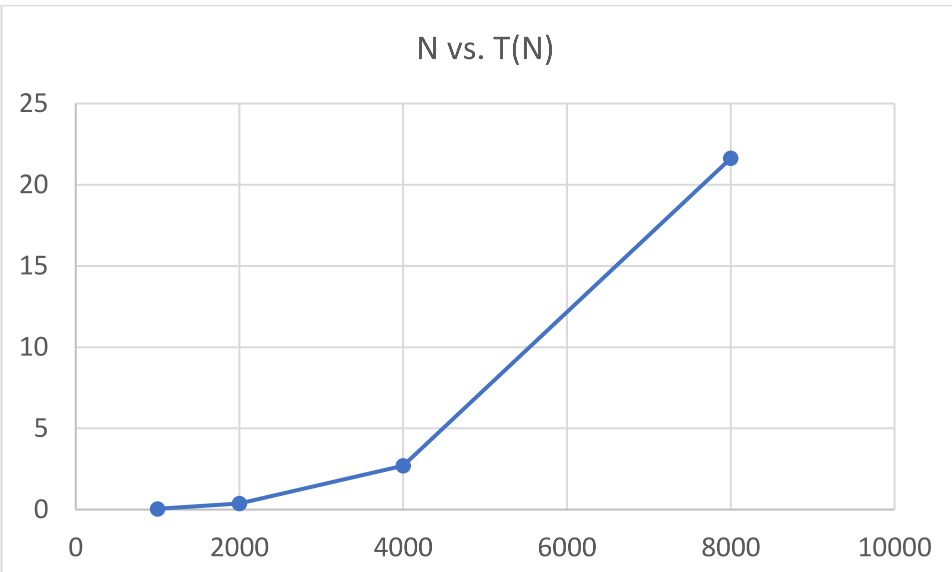
Chork Hieng

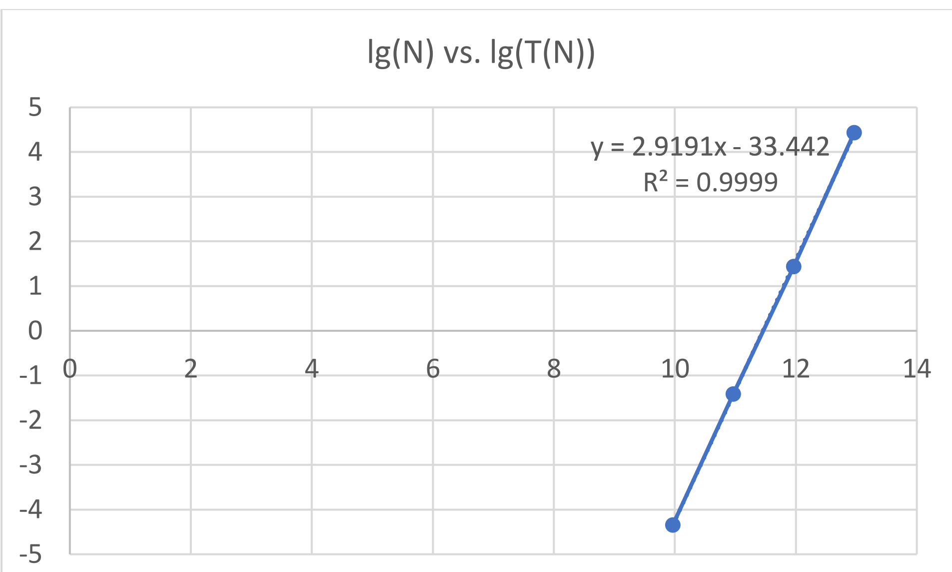
Algorithm Analysis Lab

Timing Tests:

Algorithm 1

N	Run 1	Run 2	Run 3	Run 4	Run 5	Average	lg(N)	lg(T(N))
1000	0.047	0.049	0.051	0.049	0.05	0.0492	9.965784	-4.3452
2000	0.346	0.349	0.346	0.49	0.346	0.3754	10.96578	-1.4135
4000	2.708	2.708	2.711	2.706	2.714	2.7094	11.96578	1.437973
8000	21.814	21.607	21.574	21.563	21.58	21.6276	12.96578	4.434802



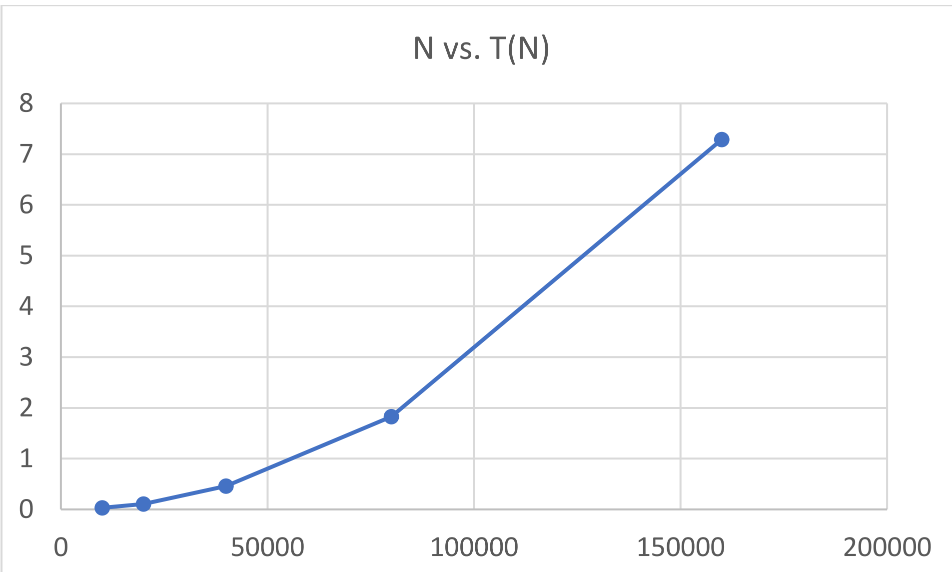


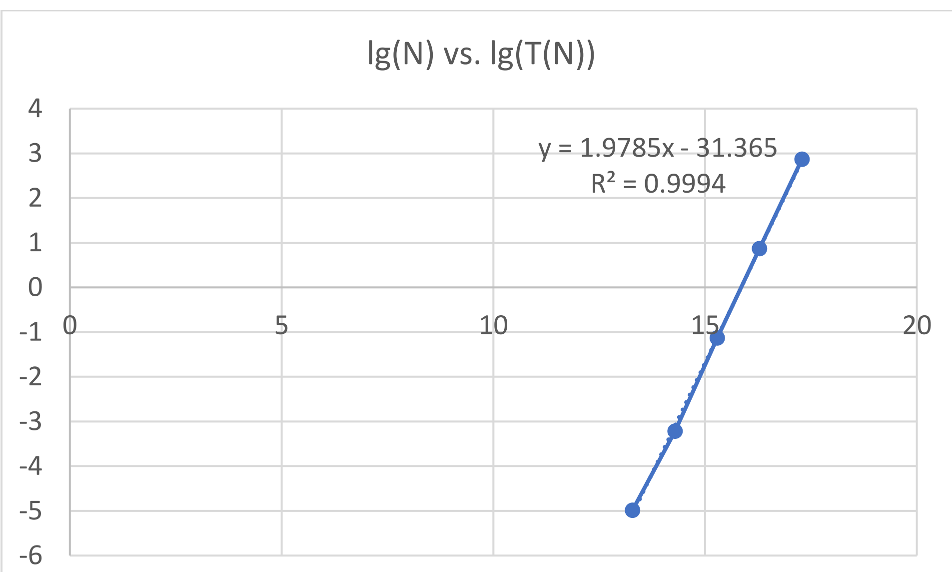
Chork Hieng

Algorithm Analysis Lab

Algorithm 2

N	Run 1	Run 2	Run 3	Run 4	Run 5	Average	lg(N)	lg(T(N))
10000	0.031	0.032	0.032	0.032	0.031	0.0316	13.28771	-4.98393
20000	0.107	0.108	0.106	0.108	0.109	0.1076	14.28771	-3.21625
40000	0.457	0.459	0.458	0.457	0.458	0.4578	15.28771	-1.12721
80000	1.827	1.823	1.825	1.826	1.83	1.8262	16.28771	0.868845
160000	7.282	7.277	7.274	7.28	7.339	7.2904	17.28771	2.865998



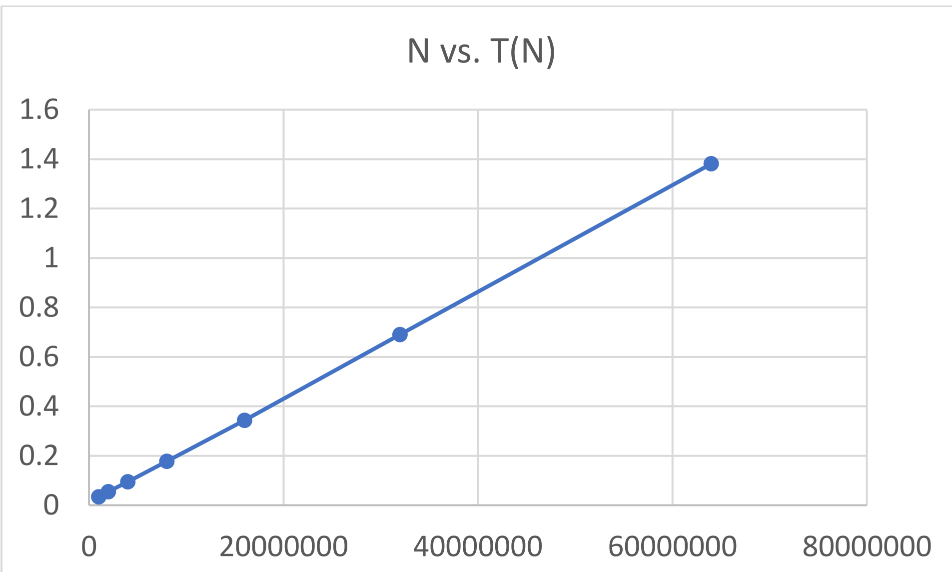


Chork Hieng

Algorithm Analysis Lab

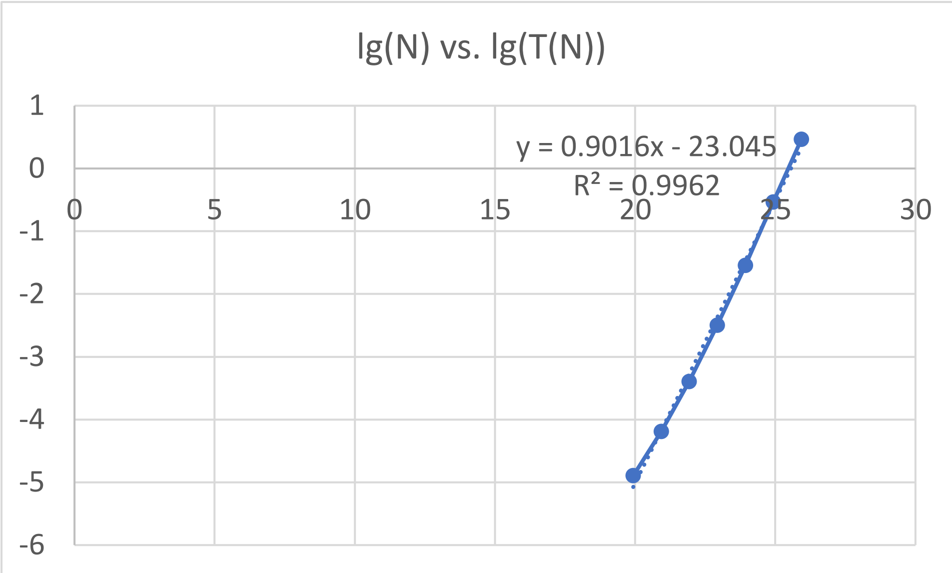
Algorithm 3

N	Run 1	Run 2	Run 3	Run 4	Run 5	Average	lg(N)	lg(T(N))
1000000	0.033	0.034	0.033	0.034	0.034	0.0336	19.93157	-4.89539
2000000	0.054	0.055	0.055	0.055	0.055	0.0548	20.93157	-4.18968
4000000	0.096	0.096	0.093	0.095	0.095	0.095	21.93157	-3.39593
8000000	0.178	0.181	0.175	0.173	0.179	0.1772	22.93157	-2.49655
16000000	0.347	0.344	0.344	0.343	0.338	0.3432	23.93157	-1.54288
32000000	0.692	0.689	0.693	0.695	0.681	0.69	24.93157	-0.53533
64000000	1.381	1.382	1.378	1.378	1.386	1.381	25.93157	0.465713



Chork Hieng

Algorithm Analysis Lab

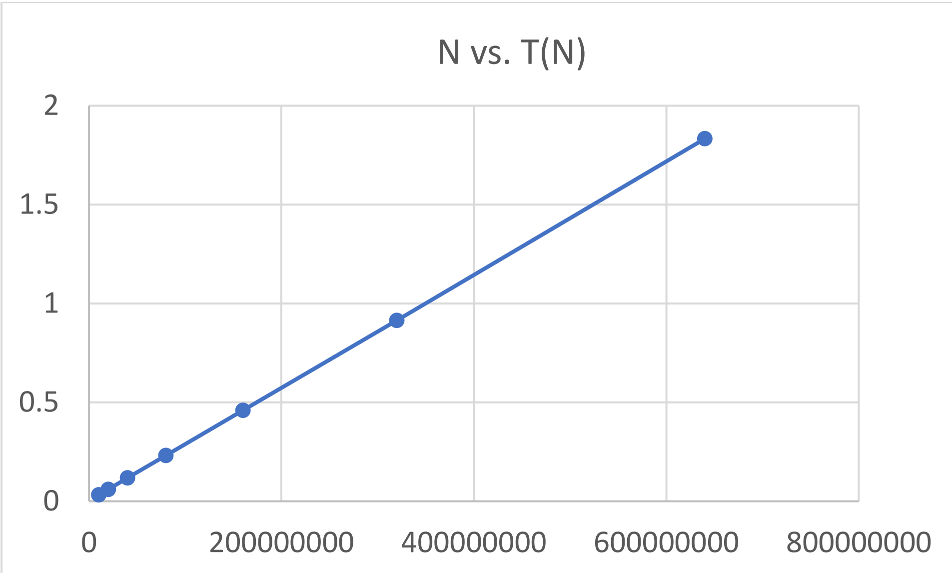


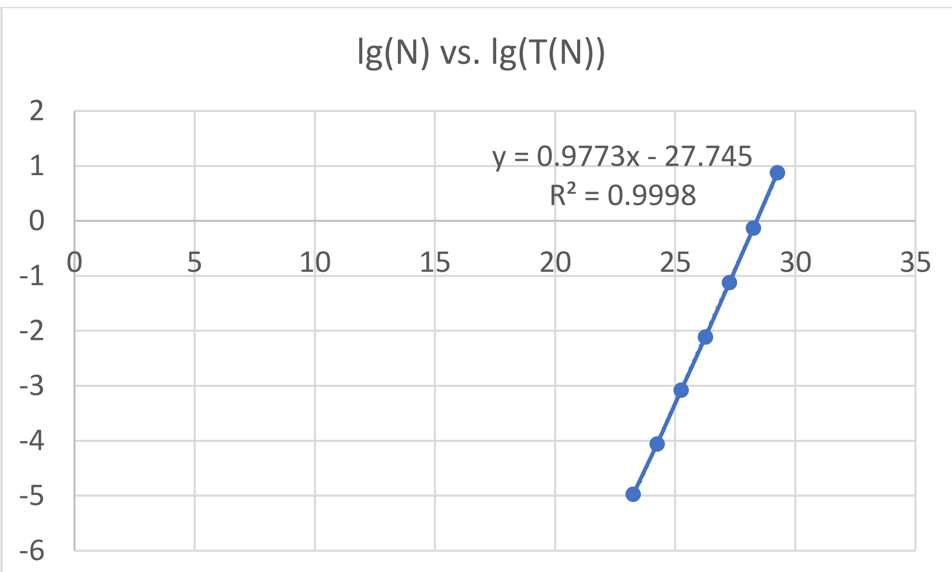
Algorithm 4

N	Run 1	Run 2	Run 3	Run 4	Run 5	Average	lg(N)	lg(T(N))
10000000	0.032	0.032	0.032	0.031	0.032	0.0318	23.2535	-4.97483
20000000	0.061	0.06	0.06	0.06	0.059	0.06	24.2535	-4.05889
40000000	0.117	0.117	0.118	0.12	0.118	0.118	25.2535	-3.08314
80000000	0.231	0.23	0.231	0.231	0.231	0.2308	26.2535	-2.11528
160000000	0.458	0.458	0.459	0.461	0.461	0.4594	27.2535	-1.12218
320000000	0.914	0.913	0.913	0.913	0.915	0.9136	28.2535	-0.13037
640000000	1.823	1.823	1.826	1.821	1.872	1.833	29.2535	0.874207

Chork Hieng

Algorithm Analysis Lab





Analyzing Codes:

Algorithm 1:

```
public static int maxSubsequence1(int[] a) {
    int maxSum = 0;
    for (int i = 0; i < a.length; i++) {
        for (int j = i + 1; j <= a.length; j++) {
            int sum = 0;
```

```
                for (int k = i; k < j; k++)
                    sum += a[k];
                if (sum > maxSum)
                    maxSum = sum;
            }
        }
        return maxSum;
    }
```

The outer loop executes N times: from 0 to (N – 1)

The first inner loop also executes N times: from 1 to N, and then increment j by 1

The second inner loop starts from i to j:

N {1 + 2 + 3 + …. + (N – 2) + (N – 1) + N} = (N^2) (N + 1)/2

⇨ f(N) = (N^2) (N + 1)/2

= (N^3)/2 + (N^2)/2

Tilde approximation: g(N) = (N^3)/2

The order of growth: N^3

As N grows larger, the time of executions increases very quickly.

Algorithm 2:

```
    public static int maxSubsequence2(int[] a) {
        int maxSum = 0;
        for (int i = 0; i < a.length; i++) {
            int sum = 0;
            for (int j = i; j < a.length; j++) {
                sum += a[j];
                if (sum > maxSum)
                    maxSum = sum;
            }
        }
        return maxSum;
    }
```

The outer loop executes N times: from 0 to N – 1

The inner loop: from i to N:

- ⇨ N + (N – 1) + (N – 2) + ……………… + 3 + 2 + 1 = N(N + 1)/2

Tilde approximation: g(N) = (N^2)/2

Order of growth: N^2

N^2 grows very fast but runs much faster comparing to N^3 when N becomes very large.

Algorithm 3:

```java
    public static int maxSubsequence3(int[] a, int lo, int hi) {

        // Base case: a 1-element range.
        if (hi - lo == 1)
            return Math.max(a[lo], 0);

        int mid = lo + (hi - lo) / 2;
        int maxLeft = maxSubsequence3(a, lo, mid);
        int maxRight = maxSubsequence3(a, mid, hi);

        int maxLeftBorder = 0;
        int leftBorder = 0;
        for (int i = mid; i > lo;) {
            leftBorder += a[--i];
            if (leftBorder > maxLeftBorder)
                maxLeftBorder = leftBorder;
        }

        int maxRightBorder = 0;
        int rightBorder = 0;
        for (int i = mid; i < hi;) {
            rightBorder += a[i++];
            if (rightBorder > maxRightBorder)
                maxRightBorder = rightBorder;
        }

        return max3(maxLeft, maxRight, maxLeftBorder +
maxRightBorder);
    }
```

```java
public static int maxSubsequence3(int[] a) {
    return maxSubsequence3(a, 0, a.length);
}
```

For the first part: `maxSubsequence3(int[] a, int lo, int hi)`

The first loop runs N/2 times: from (lo + 1) to mid

The second loop runs N/2 times: from mid to (hi – 1)

- ⇨ f(N) = N/2 + N/2 = N

Tilde approximation: g(N) = N

Order of growth: N

For the second part (`maxSubsequence3(int[] a)`)

Using the first part and then splitting to two equal parts

- ⇨ N(1/2 + 1/4 + 1/8 + ...) = N (lg N + 1)
- ⇨ f(N) = N lg N + N

Tilde approximation: g(N) = N lg N

Order of growth: N lg N

Algorithm 4:

```java
public static int maxSubsequence4(int[] a) {
    int maxSum = 0;
    int sum = 0;
    for (int n : a) {
        sum += n;
        if (sum > maxSum)
            maxSum = sum;
        else if (sum < 0)
            sum = 0;
    }
    return maxSum;
}
```

The run time is N:

Function f(N) = N

Chork Hieng

Algorithm Analysis Lab

Tilde approximation: g(N) = N

Order of growth: N