Mastering Data Structures and Algorithms
Homework 1

Yiying Peng (Christine)

January 29, 2019

Problem 1.1

Order the following functions by growth rate: N, $\sqrt{N}$, $N^{1.5}$, N^2, $N \log N$, $N \log \log N$, $N \log^2 N$, $N \log(N^2)$, $\frac{2}{N}$, 2^N, $2^{N/2}$, 37, $N^2 \log N$, N^3. Indicate which functions grow at the same rate.

Sol. In terms of Big-Oh notation, the functions can be ordered as follows.

$$\begin{aligned}\frac{2}{N} &< 37 < N \log \log N < N \log N = N \log(N^2) \\ &< N \log^2 N < N^{1.5} < N^2 < N^2 \log N < N^3 < 2^{N/2} < 2^N\end{aligned}$$

□

Problem 1.2

For each of the following six program fragments:

a. Give an analysis of the running time (Big-Oh will do).

b. Implement the code in the language of your choice, and give the running time for several values of N.

c. Compare your analysis with the actual running times.

(1)
```
sum = 0;
for( i = 0; i < n; ++i )
    ++sum;
```

(2)
```
sum = 0;
for( i = 0; i < n; ++i )
    for( j = 0; j < n; ++j )
        ++sum;
```

(3)
```
sum = 0;
for( i = 0; i < n; ++i )
    for( j = 0; j < n * n; ++j )
        ++sum;
```

(4)
```
sum = 0;
for( i = 0; i < n; ++i )
    for( j = 0; j < i; ++j )
        ++sum;
```

(5)
```
sum = 0;
for( i = 0; i < n; ++i )
    for( j = 0; j < i * i; ++j )
        for( k = 0; k < j; ++k )
            ++sum;
```

(6)
```
sum = 0;
for( i = 1; i < n; ++i )
    for( j = 1; j < i * i; ++j )
        if( j % i == 0 )
            for( k = 0; k < j; ++k )
                ++sum;
```

Sol. The answer to three sub-problems for each of six program fragments are listed as follows.

(1) a. $O(n)$

b.

N	Elapsed Time	Ratio
8	0.000003	
16	0.000004	1.4381
32	0.000006	1.6689
64	0.000011	1.8492
128	0.000021	1.8906
256	0.000045	2.1283
512	0.000083	1.8565
1024	0.000175	2.1040
2048	0.000342	1.9574
4096	0.000750	2.1943
8192	0.001612	2.1490
16384	0.002903	1.8011
32768	0.004810	1.6571
65536	0.009743	2.0254
131072	0.017759	1.8228
262144	0.035810	2.0165
524288	0.071837	2.0060
1048576	0.144966	2.0180

c. The running time for this algorithm is $O(n)$. At each new row we have doubled the input size, so we would expect the experimental growth rate to approach 2 , which it does.

(2) a. $O(n^2)$

b.

N	Elapsed Time	Ratio
8	0.000012	
16	0.000038	3.1431
32	0.000140	3.7116
64	0.000526	3.7633
128	0.002140	4.0648
256	0.008538	3.9894
512	0.034538	4.0452
1024	0.137729	3.9878

c. The running time for this algorithm is $O(n^2)$. At each new row we have doubled the input size, so we would expect the experimental growth rate to approach 2^2 , which it does.

(3) a. $O(n^3)$

b.

N	Elapsed Time	Ratio
8	0.000072	
16	0.000516	7.1304
32	0.004387	8.4967
64	0.034894	7.9543
128	0.282396	8.0930

c. The running time for this algorithm is $O(n^3)$. At each new row we have doubled the input size, so we would expect the experimental growth rate to approach 2^3 , which it does.

(4) a. $O(n^2)$

b.

N	Elapsed Time	Ratio
8	0.000008	
16	0.000022	2.9175
32	0.000074	3.3678
64	0.000269	3.6468
128	0.001134	4.2133
256	0.004339	3.8272
512	0.017111	3.9434
1024	0.068113	3.9805

c. The running time for this algorithm is $O(n^2)$. At each new row we have doubled the input size, so we would expect the experimental growth rate to approach 2^2 , which it does.

(5) a. $O(n^5)$

b.

N	Elapsed Time	Ratio
8	0.000378	
16	0.012082	31.9388
32	0.398985	33.0220

c. The running time for this algorithm is $O(n^5)$. At each new row we have doubled the input size, so we would expect the experimental growth rate to approach 2^5 , which it does.

(6) a. $O(n^4)$

b.

N	Elapsed Time	Ratio
8	0.000092	
16	0.001228	13.3491
32	0.017747	14.4536
64	0.279686	15.7593

c. The running time for this algorithm is $O(n^4)$. At each new row we have doubled the input size, so we would expect the experimental growth rate to approach 2^4 , which it does.

□

Problem 1.3

An algorithm takes 0.5 ms for input size 100. How long will it take for input size 500 if the running time is the following (assume low-order terms are negligible)?

a. linear

b. $O(N \log N)$

c. quadratic

d. cubic

Sol. The corresponding answers are as below.

a. linear: 0.5 ms $\times \frac{500}{100} = 2.5$ ms

b. $O(N \log N)$: 0.5 ms $\times \frac{500 \log 500}{100 \log 100} \approx 3.37$ ms

c. quadratic: 0.5 ms $\times \frac{500^2}{100^2} = 12.5$ ms

d. cubic: 0.5 ms $\times \frac{500^3}{100^3} = 62.5$ ms

□

Problem 1.4

Give an efficient algorithm to determine if there exists an integer i such that $A_i = i$ in an array of integers $A_1 < A_2 < A_3 < \cdots < A_N$. What is the running time of your algorithm?

Sol. Since it is an ascending order array, we can use binary search to find the integer. A snapshot of Python implementation is as following.

```
def binary_search(input_array, value):
    l = 0
    r = len(input_array) - 1
    while l < r:
        m = (l + r) / 2
        if input_array[m] < value:
            l = m + 1
        elif input_array[m] > value:
            r = m
        else:
            return m
    return -1
```

It is noticeable that this algorithm each time could eliminate half of the remaining elements, thus the overall time complexity is $O(\log n)$. □

Problem 1.5

Programs A and B are analyzed and found to have worst-case running times no greater than $150N \log_2 N$ and N^2, respectively. Answer the following questions, if possible:

a. Which program has the better guarantee on the running time for large values of N ($N > 10,000$)?

b. Which program has the better guarantee on the running time for small values of N ($N < 100$)?

c. Which program will run faster on average for $N = 1,000$?

d. Is it possible that program B will run faster than program A on all possible inputs?

Sol. The corresponding answers are as below.

a. Program A.
If $N = 100000$. We can get the time complexity of program A is around 172693881, which is faster than programs B's 10000000000.

b. Program B.
If $N = 10$. We can get the time complexity of program A is 3453, which is slower than program B's 100.

c. Program B.
When $N = 1000$. We can get the time complexity of program A is 1036163, which is slower than program B's 1000000.

d. No.
Program B only run faster than program A in smaller inputs.

□