Complexity: Problems for Week 4
with Model Solutions

Exercise 1 *Given the running times of programs/algorithms, evaluate the corresponding complexities.*

(a) The running time of my program, on an argument of size n, is $3n^2+9n+8$ seconds. Is this $O(n^2)$? Is it $O(n)$? Is it $O(n^3)$?

(b) The running time of my program, on an argument of size n, is 5^n seconds for $n < 1000$, and $3n^2 + 9n + 8$ seconds for $n \geqslant 1000$. Is this $O(n^2)$? Is it $O(n)$? Is it $O(n^3)$?

(c) On an argument of size n, I first run a program whose running time is in $O(n^2)$, and then run a program whose running time is in $O(n^3)$. Show that the total running time is in $O(n^3)$.

(d) Suppose you have two algorithms to solve a given problem. The first algorithm has a running time of $3n^2 + 2n + 33$ while the second algorithm has a running time of $2^n - 5n + 5$. Which one will you prefer and why?

Solution 1

(a) Let's see if the complexity is $O(n^2)$. Write $T(n)$ for the running time. Provided $n \geq 1$, we can write:

$$\begin{aligned}\frac{T(n)}{n^2} &= \frac{3n^2+9n+8}{n^2}\\ &= 3+\frac{9}{n}+\frac{8}{n^2}\\ &\leqslant 3+9+8 \quad (for\ n=1)\\ &\leqslant 20\end{aligned}$$

So taking M to be 1 and C to be 20, we have shown that $T(n)$ is $O(n^2)$.

Let's see if the complexity is $O(n)$. Provided $n \geq 1$, we can write:

$$\begin{aligned}\frac{f(n)}{g(n)} &= \frac{3n^2+9n+8}{n}\\ &= 3n+9+\frac{8}{n}\\ &\geqslant 3n\end{aligned}$$

This doesn't have an upper bound, so $f(n)$ is not $O(n)$

Let's see if the complexity is $O(n^3)$. Provided $n \geq 1$, we can write:

$$\begin{aligned}\frac{f(n)}{g(n)} &= \frac{3n^2+9n+8}{n^3} \\ &= \frac{3}{n}+\frac{9}{n^2}+\frac{8}{n^3} \\ &\leqslant 3+9+8 \quad (for\ n=1) \\ &\leqslant 20\end{aligned}$$

So this is in $O(n^3)$. In fact, anything in $O(n^2)$ is also in $O(n^3)$.

(b) *For $n \geq 1000$, we have $\frac{T(n)}{n^2} \leqslant 20$, so taking M to be 1000 and C to be 20, we have shown that $T(n)$ is $O(n^2)$ as in part (a) above. And just as in part (a), it is not $O(n)$ and it is $O(n^3)$.*

By the way, for $n = 999$, the time taken is

$$T(999) = 5^{999} \approx 1.86 \times 10^{698}$$

seconds, which is prohibitive.

(c) *Let the running time of the first program be $f(n)$. Since it is $O(n^2)$, there are numbers M and C such that for all $n \geqslant M$ we have $f(n) \leqslant Cn^2$.*

Let the running time of the first program be $g(n)$. Since it is $O(n^3)$, there are numbers N and D such that for all $n \geqslant N$ we have $g(n) \leqslant Dn^3$.

So for any $n \geqslant \max(M, N)$, we have

$$\begin{aligned} f(n) &\leqslant Cn^2 \\ \text{and } g(n) &\leqslant Dn^3 \\ \text{so } f(n)+g(n) &\leqslant Cn^2+Dn^3 \\ &\leqslant Cn^3+Dn^3 \\ &= (C+D)n^3\end{aligned}$$

So $f(n) + g(n)$ is $O(n^3)$.

(d) *For sufficiently large inputs, the first algorithm is faster.*

The running time of first algorithm is $3n^2 + 2n + 33$, which corresponds to a complexity of $O(n^2)$, and the running time of the second algorithm is $2^n - 5n + 5$, having a complexity of $O(2^n)$. We can easily see that the complexity of first algorithm is polynomial and the second one is exponential. As explained in the handout, by the binomial theorem, an exponential function is larger than any polynomial.

Exercise 2 *The following program operates on an array of characters that are all* `a` *or* `b`*.*

```
void f (char[] p) {
  elapse(1 second);
  for (nat i = 0; i<p.length(), i++) {
    if (p[i]=='a'){
      elapse(1 second);
    } else {
      elapse(2 seconds);
    }
    elapse (1 second);
  }
}
```

What is the average time taken to process an array of length 4*, assuming that the character in position* i *(starting from 0) has probability* 2^{-i} *of being* `a`*, and that the characters are independent? Also, what would be the worst case?*

Solution 2

Here is a table of running times.

Array contents	*Probability*			*Time*	*Contribution*
`aaaa`	$1 \times \frac{1}{2} \times \frac{1}{4} \times \frac{1}{8}$	$=$	$\frac{1}{64}$	$9s$	$\frac{9}{64}s$
`aaab`	$1 \times \frac{1}{2} \times \frac{1}{4} \times \frac{7}{8}$	$=$	$\frac{7}{64}$	$10s$	$\frac{70}{64}s$
`aaba`	$1 \times \frac{1}{2} \times \frac{3}{4} \times \frac{1}{8}$	$=$	$\frac{3}{64}$	$10s$	$\frac{30}{64}s$
`aabb`	$1 \times \frac{1}{2} \times \frac{3}{4} \times \frac{7}{8}$	$=$	$\frac{21}{64}$	$11s$	$\frac{231}{64}s$
`abaa`	$1 \times \frac{1}{2} \times \frac{1}{4} \times \frac{1}{8}$	$=$	$\frac{1}{64}$	$10s$	$\frac{10}{64}s$
`abab`	$1 \times \frac{1}{2} \times \frac{1}{4} \times \frac{7}{8}$	$=$	$\frac{7}{64}$	$11s$	$\frac{77}{64}s$
`abba`	$1 \times \frac{1}{2} \times \frac{3}{4} \times \frac{1}{8}$	$=$	$\frac{3}{64}$	$11s$	$\frac{33}{64}s$
`abbb`	$1 \times \frac{1}{2} \times \frac{3}{4} \times \frac{7}{8}$	$=$	$\frac{21}{64}$	$12s$	$\frac{252}{64}s$
`baaa`	$0 \times \frac{1}{2} \times \frac{1}{4} \times \frac{1}{8}$	$=$	0	$10s$	0
`baab`	$0 \times \frac{1}{2} \times \frac{1}{4} \times \frac{7}{8}$	$=$	0	$11s$	0
`baba`	$0 \times \frac{1}{2} \times \frac{3}{4} \times \frac{1}{8}$	$=$	0	$11s$	0
`babb`	$0 \times \frac{1}{2} \times \frac{3}{4} \times \frac{7}{8}$	$=$	0	$12s$	0
`bbaa`	$0 \times \frac{1}{2} \times \frac{1}{4} \times \frac{1}{8}$	$=$	0	$11s$	0
`bbab`	$0 \times \frac{1}{2} \times \frac{1}{4} \times \frac{7}{8}$	$=$	0	$12s$	0
`bbba`	$0 \times \frac{1}{2} \times \frac{3}{4} \times \frac{1}{8}$	$=$	0	$12s$	0
`bbbb`	$0 \times \frac{1}{2} \times \frac{3}{4} \times \frac{4}{8}$	$=$	0	$13s$	0
Worst case: `bbbb`				$13s$	
Average case					$\frac{712}{64}s = 11\frac{1}{8}s$

Note that the input `bbbb` *is possible, despite having probability 0 of occurring.*

Exercise 3 *(a) My program takes 2^{2^n} steps on every input of size $n < 100000$, and $5n^3 + 3n + 8$ steps on every input of size $n \geqslant 100000$. Show that the running time is in $O(n^3)$.*

(b) Show that if $f \in O(g)$ and $g \in O(h)$ then $f \in O(h)$.

(c) Show that $2^n \in O(n!)$

Solution 3

(a) Let the running time be $f(n)$. We want to show that there is M and C such that for all $n \geqslant M$ we have $f(n)/(n^3) \leqslant C$. Put $M = 100000$ and $C = 16$. For $n \geqslant M$ we have:

$$\begin{aligned} f(n) &= 5n^3 + 3n + 8 \\ &\leqslant 5n^3 + 3n^3 + 8n^3 \\ &= Cn^3. \end{aligned}$$

For $n \geqslant 100000$ the running time is $5n^3 + 3n + 8 \leqslant 5n^3 + 3n^3 + 8n^3 = 16n^3$. Therefore, the running time is in the complexity class of $O(n^3)$.

(b) We know that:

- *there are M and C such that for all $n \geqslant M$ we have $f(n) \leqslant C \times g(n)$*
- *there are M' and C' such that for all $n \geqslant M'$ we have $g(n) \leqslant C' \times h(n)$.*

We want to show that there are M'' and C'' such that for all $n \geqslant M''$ we have $f(n) \leqslant C'' \times h(n)$.

Let M'' be the maximum of M and M', and let C'' be $C \times C'$. Then for $n \geqslant M''$ we have

$$\begin{aligned} & f(n) \leqslant C \times g(n) \\ & g(n) \leqslant C' \times h(n) \\ \textit{So}\quad & f(n) \leqslant C \times g(n) \\ & \leqslant C \times (C' \times h(n)) \\ & \leqslant (C \times C') \times h(n) \\ & = C'' \times h(n) \end{aligned}$$

(c) By the following argument:

$$\frac{2^n}{n!} = \frac{2}{1} \times \frac{2}{2} \times \cdots \times \frac{2}{n} \leqslant 2$$

Exercise 4 *(a) A sorting method with Big-O complexity $O(n \log n)$ spends exactly 1 millisecond to sort 1000 data items. Assuming that time $T(n)$ of sorting n items is directly proportional to $n \log n$, that is, $T(n) = Cn \log n$, derive a formula for $T(n)$, given the time $T(N)$ for sorting $N = 1000$ items, and estimate how long this method will take to sort $n = 1000000$ items.*

(b) One of the two software packages, A or B, should be chosen to process very big databases, containing up to 10^{16} records. Average processing time of the package A is $T_A(n) = 0.1n \log_2 n$ microseconds, and the average processing time of the package B is $T_B(n) = 6n$ microseconds.
Which algorithm has better performance in Big-O sense? Work out the exact conditions when these packages outperform each other.

Solution 4

(a) We are given that $N = 1000$ and the processing time for N items is $T(N) = CN \log N = 1ms$
We are required to find out the processing time $T(n) = Cn \log n$ for $n = 1000000$ items.

As we know that the constant factor C will remain the same, we can compare the above two formulas i.e. $C = \frac{T(N)}{N \log N}$ and $C = \frac{T(n)}{n \log n}$ giving us $T(n) = T(N) \cdot \frac{n \log n}{N \log N}$.

We know that the ratio of the logarithms of the same base is independent of the base, hence, any appropriate base can be used in the above formula (say, a base of 10). Therefore, for $n = 1000000$ the estimated time is:

$$T(1000000) = T(1000) \cdot \frac{1000000 \log_{10} 1000000}{1000 \log_{10} 1000}$$

$$= 1 \cdot \frac{1000000 \times 6}{1000 \times 3} = 2000ms$$

(b) In the Big-O sense, the algorithm B of complexity $O(n)$ is better than A of complexity $O(n \log n)$. The package B has better performance in case of very large values of n, as its linear i.e. the package B begins to outperform A when $T_A(n) \geq T_B(n)$, that is, when $0.1n \log_2 n \geq 6n$. This inequality reduces to $0.1 \log_2 n \geq 6$, or $n \geq 2^{60} \approx 10^{18}$. Thus for processing up to 10^{16} data items, the package of choice is A.

Exercise 5 *Compute the time complexity (with respect to N) of the following functions. Give an informal justification. (Complexity proof is not required.)*

(a) The function A() is doing some processing on a string:

```
void A(String str){
  nat N = str.length();
  for(nat i = 0; i < N; i = i+1){
    // p seconds elapse
  }
  for(nat i = 0; i < N; i = i+1){
    for(nat j = 0; j < N; j = j+1){
      // q seconds elapse
    }
  }
  for(nat i = 0; i < N; i = i+1){
    for(nat j = 0; j < N; j = j+1){
      // r seconds elapse
    }
  }
}
```

(b) The function B() is doing some processing on a string:

```
void B(String str){
  nat N = str.length();
  for(nat j = 2 * N; j > 0; j = j-1){
    for(nat i = N; i > 0; i = i/2){
      // p seconds elapse
    }
  }
}
```

(c) The function C() is doing some processing on a string:

```
void C(String str){
  nat N = str.length();
  nat i = 1000;
  nat k = 0;
  // p seconds elapse
  while(i > 1){
    for(nat j = 1; j < N*N; j = j+1){
      // q seconds elapse
      if (j < N)
        k += 1; // r seconds elapse
    }
    i = i - 1; // s seconds elapse
  }
}
```

(d) The function D() is processing the number N, using recursion:

```
nat D(nat N){
  if (N == 1){
    return 1; // p seconds elapse
  }
  else{
    D(N-1);  // q seconds elapse
    D(N-1);  // q seconds elapse
  }
}
```

Solution 5

(a) The complexity of function A() is : $O(N^2)$
***Justification**: The function contains a single loop and two nested loops, which contribute quadratic time. Therefore, the time-complexity is* $O(N^2)$ *as this indicates the maximum required time for this function.*

(b) The complexity of function B() is : $O(N \log N)$
***Justification**: There are two nested loops in this function. The outer loop runs* $2N$ *times and the inner loop runs* $\log N$ *times. Therefore the overall complexity is* $O(N \log N)$*. The constant multiplier* 2 *is ignored in Big-O notation.*

(c) The complexity of function C() is : $O(N^2)$
***Justification**: The outer while loop runs for* 1000 *times and is independent of* N*. The inner for loop runs for* N^2 *times, which gives us a complexity of* $O(N^2)$*.*

(d) The complexity of function D() is : $O(2^N)$
***Justification**: The function contains 2 recursive calls for each of its calls. It creates a binary tree-like call structure i.e. 1 call leads to 2 calls, which in turn lead to 4 calls and so on. Therefore the time-complexity is exponential.*

Exercise 6 *Callum writes a program that operates on an array of* `a`*'s and* `b`*'s. The time taken is* $5A^2 + 2B^3$*, where* A *is the number of* `a`*'s and* B *the number of* `b`*'s. If* n *is the length of the array, show that, in the worst case, the time taken is* $O(n^3)$*.*

Solution 6
Since $A \leqslant n$ *and* $B \leqslant n$*, the time taken is in all cases* $5A^2 + 2B^3 \leqslant 5n^3 + 2n^3 = 7n^3$*.*

***Note**: Some students suggested an alternative method: for* $n \geqslant 3$*, the worst case is an array of* `b`*'s, which has running time* $2n^3$*. However, I (Paul) wasn't convinced that this is indeed the worst case, until Achim Jung sent me the following nice proof. The function* $f(x) = 5(n-x)^2 + 2x^3$ *has second derivative* $10 + 12x$*, which is positive for all* $x \geqslant 0$*, hence* $f(x)$ *itself is convex for* $x \geqslant 0$*. Therefore it assumes its maxima at the endpoints of any interval we want to consider, in our case the interval from* 0 *to* n*, in other words, we only need to consider* $x = 0$ *and* $x = n$*. Since* $5n^2 < 2n^3$ *for* $n \geqslant 3$*, we are done.*