Scope: DPV Chapter 1 and Sections 2.1, 2.2, 2.3, 2.4, 2.5, and 3.1.

Problem 1 For each of the following pairs of functions f and g, state whether $f(n) \in \Theta(g(n))$, and if not, then whether $f(n) \in O(g(n))$ or $f(n) \in \Omega(g(n))$.

(Example: For $f(n) = 2n$ and $g(n) = n$ the answer would be $f(n) \in \Theta(g(n))$.)

	$f(n) =$	$g(n) =$	**Your answer**
(w)	3^{n-1}	2^n	
(q)	$\frac{1}{15}n^2 + 97^{98^{99}} \cdot n$	n^2	
(s)	n^2	$\frac{n(n+1)(n+2)}{(n+3)}$	
(y)	$n^{\log_2 n}$	$2^{2(\log_2 n)}$	
(u)	$n^5 + 12n$	$\frac{2^n}{n^3+6}$	

Problem 2 In each of the following, ❶ state whether the statement is true or false ***and*** ❷ justify your answer.

(g) $n^2 2^n \in \Omega(n 2^{2n})$?

(i) $n^2 \in O\left(\frac{(n+1)^4}{n}\right)$?

Problem 3 Prove by induction that, for each $n \geq 0$, $n^3 + 2n$ is divisible by 3.

Problem 4 For each of the following give a tight $\Theta(\cdot)$ bound on the number of times the `z=z+1` statement is executed and justify your answer.

a.
```
j = 1;
while (j<=n) {
  j = j*2;
  z = z+1;
}
```

b.
```
j = 1;
while (j<=n) {
  j = j+j+j;
  z = z+1;
}
```

c.
```
for(k=0; k<=n; k++)
  for(j=n; j>=k;j--)
    z = z+1
```

d.
```
for(k=n; k>=0; k--)
  for(j=0; j<=k;j++)
    z= z+1
```

Problem 5 *[This problem is based on DPV Exercise 2.4.]* For each of the following state the recurrence for the algorithm and solve it via the Master Recurrence Theorem. (See the Math Facts page for the theorem.)

(a) Algorithm W solves problems of size n by dividing them into nine subproblems of size $n/2$, recursively solving them, and then combining the solutions in $\Theta(n^2)$ time.

(b) Algorithm X solves problems of size n by dividing them into five subproblems of size $n/2$, recursively solving them, and then combining the solutions in $\Theta(n)$ time.

(c) Algorithm Y solves problems of size n by dividing them into 7 problems of size $n/3$, recursively solving them, and then combining the solutions in $\Theta(n^2)$ time.

(d) Algorithm Z solves problems of size n by dividing them into nine subproblems of size $n/3$, recursively solving them, and then combining the solutions in $\Theta(n^2)$ time.

Problem 6 *[This problem is based on DPV Exercise 2.23(a).]* A value v is said to be the *majority value* of a list $[x_1, \ldots, x_k]$ when **more than half** of the elements of the list equal v, i.e., $\#(\{\, i : x_i = v \,\}) > k/2$; a value that is **not** a majority value is called a *minority value*. Given a list, we want to decide if a list has a majority value, and if so, we want to find this value. We assume we can do equality tests in constant time.

(a) Given a value y and a list $[x_1, \ldots, x_k]$, show how we can test whether y is the majority value of $[x_1, \ldots, x_k]$ in $\Theta(k)$ time. *(Hint: Yes, this is really, really easy.)*

(b) Suppose we split the list $[x_1, \ldots, x_k]$ into two parts $[x_1, \ldots, x_{\lfloor k/2 \rfloor}]$ and $[x_{1+\lfloor k/2 \rfloor}, \ldots, x_k]$. Suppose v is a minority value in both sublists. Explain why v must be a minority value in the full list. (If you want, you may assume k is even.)

(c) Suppose v is a majority value in $[x_1, \ldots, x_k]$. Use part b to explain why v must be a majority value in at least one of $[x_1, \ldots, x_{\lfloor k/2 \rfloor}]$ and $[x_{1+\lfloor k/2 \rfloor}, \ldots, x_k]$.

(d) Use parts a and b to give a divide-and-conquer algorithm for the majority value problem. In the case that the list ***fails*** to have a majority value, the algorithm should return the value of the first element of the list.

(e) Use the master method to give a tight $O(\,\cdot\,)$-time bound on the algorithm's run time.

Problem 7 Suppose $G = (V, E)$ is a directed graph. The *reverse* of G is the graph $G^R = (V, E^R)$ where $E^R = \{\, (b, a) \mid (a, b) \in E \,\}$.

(a) Give an algorithm for computing G^R from G where graphs are represented via ***adjacency lists***. State and justify the $O(\cdot)$-runtime of your algorithm.

(b) Give an algorithm for computing G^R from G where graphs are represented via ***adjacency matrices***. State and justify the $O(\cdot)$-runtime of your algorithm.

Some Reference Math Facts

a. $a^m \cdot a^n = a^{m+n}$.
b. $a^{m \cdot n} = (a^m)^n = (a^n)^m$.
c. $a^n \cdot b^n = (a \cdot b)^n$.
d. $\log_a a^n = n$.
e. $a^{\log_a n} = n$.
f. $c \cdot \log_a b = \log_a b^c$.
g. $\log_a(b \cdot c) = (\log_a b) + (\log_a c)$.
h. $\log_a x = (\log_a b) \cdot (\log_b x)$.
i. $a^{\log_b c} = c^{\log_b a}$.
j. $\sum_{k=1}^{n} k = \frac{n \cdot (n+1)}{2}$.
k. $\sum_{k=0}^{n} a^k = \frac{a^{n+1}-1}{a-1}$.
l. $\sum_{k=1}^{\infty} 2^{-k} = 1$.
m. $\sum_{k=1}^{n} k^2 = \frac{n \cdot (n+1) \cdot (2n+1)}{6}$.
n. $\sum_{k=1}^{\infty} k \cdot 2^{-k} = 2$.
o. For $a > 1$ and $b, c > 0$:

$$\lim_{n\to\infty} \frac{(\log_a n)^b}{n^c} = 0.$$

$$\lim_{n\to\infty} \frac{n^c}{(\log_a n)^b} = \infty.$$

p. For $a > 1$ and $b, c > 0$:

$$\lim_{n\to\infty} \frac{n^b}{a^{c \cdot n}} = 0.$$

$$\lim_{n\to\infty} \frac{a^{c \cdot n}}{n^b} = \infty.$$

q. If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$, then:

	$c = 0$	$0 < c < \infty$	$c = \infty$
$f(n) \in O(g(n))$	True	True	False
$f(n) \in \Theta(g(n))$	False	True	False
$f(n) \in \Omega(g(n))$	False	True	True

r. *The Master Recurrence Theorem:* Suppose $T(n) = a \cdot T(n/b) + O(n^d)$ where $a, b \geq 1$ and $d \geq 0$. Then:

- **i)** $T(n) \in O(n^d)$, if $d > \log_b a$.
- **ii)** $T(n) \in O(n^d \log n)$, if $d = \log_b a$.
- **iii)** $T(n) \in O(n^{\log_b a})$, if $d < \log_b a$.

s. *Some log tables (to three decimal places)*

n	$log_2 n$	$log_3 n$	$log_5 n$
1	0.000	0.000	0.000
2	1.000	0.631	0.431
3	1.585	1.000	0.683
4	2.000	1.262	0.861
5	2.322	1.465	1.000
6	2.585	1.631	1.113
7	2.807	1.771	1.209
8	3.000	1.893	1.292
9	3.170	2.000	1.365

t. For any integer $n > 0$ (e.g., 3), n divides 0.

Answers

Answers for Problem 1

	$f(n) =$	$g(n) =$	**Your answer**	**Why? *(You didn't need to give thie part)***
(q)	$\frac{1}{15}n^2 + 97^{98^{99}} \cdot n$	n^2	Θ	$\lim_{n\to\infty} \frac{\frac{1}{15}n^2 + 97^{98^{99}} \cdot n}{n^2} = \lim_{n\to\infty} \left(\frac{1}{15} + \frac{97^{98^{99}}}{n^2}\right) = \frac{1}{15}$.
(s)	n^2	$\frac{n(n+1)(n+2)}{n+3}$	Θ	$\lim_{n\to\infty} \frac{n^2 \cdot (n+3)}{n(n+1)(n+2)} = \lim_{n\to\infty} \frac{n^3+3n^2}{n^3+3n^2+2n} = 1$.
(u)	$n^5 + 12n$	$\frac{2^n}{n^3+6}$	O	$\lim_{n\to\infty} \frac{(n^5+12n)\cdot(n^3+6)}{2^n} = 0$ by Math Fact p.
(w)	3^{n-1}	2^n	Ω	$\lim_{n\to\infty} \frac{3^{n-1}}{2^n} = \frac{1}{2}\lim_{n\to\infty} \cdot(\frac{3}{2})^{n-1} = \infty$.
(y)	$n^{\log_2 n}$	$2^{2(\log_2 n)}$	Ω	$\cdot \lim_{n\to\infty} \frac{n^{\log_2 n}}{2^{2(\log_2 n)}} = \lim_{n\to\infty} \frac{n^{\log_2 n}}{4^{\log_2 n}} = \lim_{n\to\infty} (\frac{n}{4})^{\log_2 n} = \infty$.

Answers for Problem 2.

(g) False, because: $\lim_{n\to\infty} \frac{n^2 2^n}{n 2^{2n}} = \lim_{n\to\infty} \frac{n}{2^n} = 0$ by Math Fact p.

(i) True, because: $\lim_{n\to\infty} \frac{n^2 \cdot n}{(n+1)^4} = \lim_{n\to\infty} \frac{n^3}{n^4+4n^3+6n^2+4n+1} = 0$.

An answer to Problem 3

Base case: $n = 0$. Then $0^3 + 2 \cdot 0 = 0$ which is divisible by 3.
Induction step: Suppose $3|(n^3+2n)$. We need to show that $3|((n+1)^3+2(n+1))$. We note

$$(n+1)^3 + 2(n+1) \;=\; n^3 + 3n^2 + 3n + 1 + 2n + 2 \;=\; (n^3+2n) + 3n^2 + 3n + 3.$$

By the induction hypothesis, $3|(n^3+2n)$ and by inspection $3|(3n^2+3n+3)$.
Therefore, $3|((n+1)^3+2(n+1))$ as required.

Answers for Problem 4.

(a) Since j doubles every time through the loop, at the beginning of the i-th iteration, the value of j is 2^i. The loop exits with the first i with $2^i > n$, i.e., $i > \log_2 n$. Therefore, z increases by $\Theta(\log n)$.

(b) Since j triples every time through the loop, at the beginning of the i-th iteration, the value of j is 3^i. The loop exits with the first i with $3^i > n$, i.e., $i > \log_3 n$. Therefore, z increases by $\Theta(\log n)$.

(c) The inner loop goes through $n-k+1$ many iterations. So the inner loop increases z by $n-k+1$. In the outer loop, k goes from 0 to n in steps of 1. Therefore, z is increased by $\sum_{k=0}^{n} n-k+1 = (n+1)+n+\cdots+2+1 = \frac{1}{2}(n+1)(n+2)$ by Math Fact j. So $z \in \Theta(n^2)$.

(d) The inner loop goes through $k+1$ many iterations. So the inner loop increases z by $k+1$. In the outer loop, k goes from n to 0 in steps of -1. Therefore, z is increased by $\sum_{k=0}^{n} n-k+1 = (n+1)+n+\cdots+2+1 = \frac{1}{2}(n+1)(n+2)$ by Math Fact j. So $z \in \Theta(n^2)$.

Answers for Problem 5.

(a) For this case $a = 9$, $b = 2$, and $d = 2$. So $\log_b a = \log_2 9 \approx 3.17 > 2 = d$. Therefore, by the Master Recurrence Theorem, the algorithm runs in $\Theta(n^{\log_2 9})$ time.

(b) For this case $a = 5$, $b = 2$, and $d = 1$. So $\log_b a = \log_2 5 = 2.322 > 1 = d$. Therefore, by the Master Recurrence Theorem, the algorithm runs in $\Theta(n^{\log_2 5})$ time.

(c) For this case $a = 7$, $b = 3$, and $d = 2$. So $\log_b a = \log_3 7 = 1.893 < 2 = d$. Therefore, by the Master Recurrence Theorem, the algorithm runs in $\Theta(n^2)$ time.

(d) For this case $a = 9$, $b = 3$, and $d = 2$. So $\log_b a = \log_3 9 = 2 = d$. Therefore, by the Master Recurrence Theorem, the algorithm runs in $\Theta(n^2 \log n)$ time.

Answers for Problem 6.

(a) Scan the list and count how many times y occurs. Then: y is the majority value if and only if the count is $> k/2$. Clearly this test takes $\Theta(k)$ time.

(b) Suppose the number of y's in $[x_1, \ldots, x_{\lfloor k/2 \rfloor}]$ is $\leq \frac{1}{2}\lfloor k/2 \rfloor$ and the number of y's in $[x_{1+\lfloor k/2 \rfloor}, \ldots, x_k]$ is $\leq \frac{1}{2}(k - \lfloor k/2 \rfloor)$. So the number of y's in $[x_1, \ldots, x_k]$ must be $\leq \frac{1}{2}\lfloor \frac{k}{2} \rfloor + \frac{1}{2}(k - \lfloor \frac{k}{2} \rfloor)) = \frac{1}{2}k$.

(c) By the previous part, if y is the majority in $[x_1, \ldots, x_k]$, it must also be the majority in at least one of $[x_1, \ldots, x_{\lfloor k/2 \rfloor}]$ and $[x_{1+\lfloor k/2 \rfloor}, \ldots, x_k]$.

(d) Here is the algorithm:

```
procedure majority([x_i, ..., x_k])
    if k = 1 then return x_1
    mid ← ⌊k/2⌋
    m_left ← majority([x_1, ..., x_mid]
    m_right ← majority([x_{mid+1}, ..., x_k])
    if m_left occurs > k/2 many times in [x_1, ..., x_k]
       then return m_left
    else if m_right occurs > k/2 many times in [x_1, ..., x_k]
       then return m_right
    else return x_1
```

(e) On a list of length $k > 1$ there is a recursion on a list of length $\lfloor k/2 \rfloor$ and another on a list of length $\lceil k/2 \rceil$. Besides the recursions, the work done in a call is $\Theta(k)$-time. So the recurrence is: $T(k) = 2T(k/2) + \Theta(k)$. Hence, $a = b = 2$, and $d = 1 = \log_2 2 = \log_b a$. Therefore, $T(k) \in \Theta(k \log k)$.

Answers for Problem 7.

Part (a). Just scan down G's adjacency lists and build G^R's adjacency lists as we go. Here is the algorithm. We assume $V = \{1, \ldots, n\}$.

```
function rev(adj_G[1..n])
  for i ← 1 to n do
    add_G[i] ← an empty list                    (⋆)
  for i ← 1 to n do
    for each u in adj_G[i] do
      add i to adj_R[u]                         (‡)
  return adj_R
```

Runtime analysis: The line marked-$(\star)$ is executed once per vertex and the line marked-$(\ddagger)$ is executed once per edge. Hence, the total time is $\Theta(|V| + |E|)$.

Part (b). Just take the transpose of G's adjacency matrix. Here is the code. We assume $V = \{1, \ldots, n\}$.

```
function rev(A_G[1..n, 1..n])
  A_R ← a new n × n element array
  for i ← 1 to n do
    for j ← 1 to n do A_R[i, j] ← A_G[j, i]
  return A_R
```

Initializing A_R and the nested-for loops take $\Theta(n \times n) = \Theta(|V|^2)$ time.