1. Prove that \( f(n) = 60n^2 + 5n + 1 \) is \( \Theta(n^2) \).

**Proof**

\[
60n^2 + 5n + 1 \leq 60n^2 + 5n^2 + n^2 = 66n^2 \quad \text{for } n \geq 1
\]

Take \( C_1 = 66 \)

\[
f(n) = 60n^2 + 5n + 1 \text{ is } O(n^2)
\]

Since

\[
60n^2 + 5n + 1 \geq 60n^2 \quad \text{for } n \geq 1
\]

If we take \( C_2 = 60 \)

\[
f(n) = 60n^2 + 5n + 1 \text{ is } \Omega(n^2)
\]

Since \( f(n) \) is \( O(n^2) \) and \( f(n) \) is \( \Omega(n^2) \), therefore \( f(n) \) is \( \Theta(n^2) \).

2. Prove that \( f(n) = a_kn^k + a_{k-1}n^{k-1} + \ldots + a_1n + a_0 \) is \( \Theta(n^k) \).

**Proof**

Let \( C = a_k + a_{k-1} + \ldots + a_1 + a_0 \).

Then

\[
a_kn^k + a_{k-1}n^{k-1} + \ldots + a_1n + a_0 \leq a_kn^k + a_{k-1}n^{k-1} + \ldots + a_1n + a_0 \leq (a_k + a_{k-1} + \ldots + a_1 + a_0)n^k = Cn^k
\]

Therefore,

\[
f(n) = a_kn^k + a_{k-1}n^{k-1} + \ldots + a_1n + a_0 \text{ is } O(n^k)
\]

Since

\[
a_kn^k + a_{k-1}n^{k-1} + \ldots + a_1n + a_0 \geq a_kn^k
\]

\[
f(n) = a_kn^k + a_{k-1}n^{k-1} + \ldots + a_1n + a_0 \text{ is } \Omega(n^k)
\]

Thus

\[
f(n) = a_kn^k + a_{k-1}n^{k-1} + \ldots + a_1n + a_0 \text{ is } \Theta(n^k)
\]
3. Prove that \( f(n) = 1^k + 2^k + \ldots + n^k \) is \( \Theta \left(n^{k+1}\right)\)

**Proof**

If \( k \) is a positive integer and replace each integer 1, 2, \ldots, \( n \) by \( n \)
We have

\[
1^k + 2^k + \ldots + n^k \leq n^k + n^k + \ldots + n^k = n \cdot n^k = n^{k+1}
\]

for \( n \geq 1 \);

Hence

\( f(n) = 1^k + 2^k + \ldots + n^k \) is \( O \left(n^{k+1}\right)\)

We can obtain the lower bound, throwing away the first half of the terms

\[
1^k + 2^k + \ldots + n^k \geq \left\lfloor \frac{n}{2} \right\rfloor^k + \ldots + \left(n-1\right)^k + n^k
\]

\[
\geq \left\lfloor n/2 \right\rfloor^k + \ldots + \left\lfloor n/2 \right\rfloor^k + \left\lfloor n/2 \right\rfloor^k
\]

\[
= \left\lfloor (n+1)/2 \right\rfloor \left\lfloor n/2 \right\rfloor^k \geq (n/2)(n/2)^k = n^{k+1}/2^{k+1}
\]

So we can conclude that

\( f(n) = 1^k + 2^k + \ldots + n^k \) is \( \Omega \left(n^{k+1}\right)\).

Hence

\( f(n) = 1^k + 2^k + \ldots + n^k \) is \( \Theta \left(n^{k+1}\right)\)

4. Prove that \( f(n) = \lg n! \) is \( \Theta \left(n \lg n\right) \).

**Proof**

Since \( n! = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1 \) and \( \lg a \cdot b = \lg a + \lg b \)

\( \lg n! = \lg n + \lg (n-1) + \ldots + \lg 2 + \lg 1 \)

\( \lg n + \lg (n-1) + \ldots + \lg 2 + \lg 1 \leq \lg n + \lg n + \ldots + \lg n + \lg n = n \lg n \)

We can conclude that

\( \lg n! \) is \( O \left(n \lg n\right) \).

\[
\lg n + \lg (n-1) + \ldots + \lg 2 + \lg 1 \geq \lg n + \lg (n-1) + \ldots + \lg \left\lfloor n/2 \right\rfloor
\]

\[
\geq \lg \left\lfloor n/2 \right\rfloor + \lg \left\lfloor n/2 \right\rfloor + \ldots + \lg \left\lfloor n/2 \right\rfloor
\]

\[
\geq \left(\left\lfloor (n+1)/2 \right\rfloor \lg \left\lfloor n/2 \right\rfloor \right) \geq (n/2) \lg (n/2)
\]
By mathematical induction we can show that if \( n \geq 4 \),

\[(n/2) \log (n/2) \geq (n \log n) / 4.
\]

In the last two inequalities,

\[\log n + \log (n-1) + \ldots + \log 2 + \log 1 \geq (n \log n) / 4 \quad \text{for } n \geq 4.
\]

Therefore,

\[\log n! \text{ is } \Omega (n \log n).
\]

So finally it follows that

\[\log n! \text{ is } \Theta (n \log n).
\]

5. Assuming that \( f_1 (n) \) is \( O (g_1 (n)) \) and \( f_2 (n) \) is \( O (g_2 (n)) \), prove the following statements:

a. \( f_1 (n) + f_2 (n) \) is \( O (\max (g_1 (n), g_2 (n))) \).

b. If a number \( k \) can be determined such that for all \( n > k \), \( g_1 (n) \leq g_2 (n) \), then

\[O (g_1(n)) + O(g_2(n)) \text{ is } O(g_2(n)).
\]

c. \( f_1(n) \ast f_2(n) \) is \( O(g_1(n) \ast g_2(n)) \) (rule of product).

d. \( O(cg(n)) \) is \( O(g(n)) \).

e. \( c \) is \( O(1) \).

Proof

In the following answers, these two definitions are used:

\( f_1(n) \) is \( O(g_1(n)) \) if there exist positive numbers \( c_1 \) and \( N_1 \) such that \( f_1(n) \leq c_1 g_1(n) \) for all \( n > N_1 \);

\( f_2(n) \) is \( O(g_2(n)) \) if there exist positive numbers \( c_2 \) and \( N_2 \) such that \( f_2(n) \leq c_2 g_2(n) \) for all \( n > N_2 \);

(a) From the above definitions we have

\[f_1(n) \leq c_1 \max (g_1(n), g_2(n)) \text{ for all } n \geq \max (N_1, N_2)
\]

\[f_2(n) \leq c_2 \max (g_1(n), g_2(n)) \text{ for all } n \geq \max (N_1, N_2)
\]

which implies that

\[f_1(n) + f_2(n) \leq (c_1 + c_2) \max (g_1(n), g_2(n)) \text{ for all } n \geq \max (N_1, N_2)
\]

Hence for \( c_3 = c_1 + c_2 \) and \( N_3 = \max (N_1, N_2) \),

\[f_1(n) + f_2(n) \leq c_3 \max (g_1(n), g_2(n)) \text{ for all } n \geq N_3
\]

that is \( f_1(n) + f_2(n) \text{ is } O(\max (g_1(n), g_2(n)).
\]
(b) If \( g_1(n) \leq g_2(n), \) then for \( c = \max(c_1, c_2), \)
\[
  cg_1(n) \leq cg_2(n),
\]
\[
  cg_1(n) + cg_2(n) \leq 2cg_2(n)
\]
which implies that \( O(g_1(n)) + O(g_2(n)) = O(g_2(n)). \)

(c) The rule of product, \( f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n)) \)
is true since \( f_1(n) \cdot f_2(n) \leq c_1 c_2 g_1(n) \cdot g_2(n) \) for all \( n \geq \max(N_1, N_2) \)

(d) \( O(cg(n)) = O(g(n)) \) means that any function \( f \) which is \( O(cg) \) is also \( O(g) \). Function \( f \)
is \( O(cg) \) if there are two constants \( c_1 \) and \( N \) so that \( f(n) \leq c_1 cg(n) \) for all \( n \geq N \); in this

(e) A constant \( c \) is \( O(1) \) if there exist positive numbers \( c_1 \) and \( N \) such that \( c \leq c_1, 1 \) for all \( n \geq N \); that is, the constant function \( c \) is independent of \( n \), and we can simply set \( c_1 = c \).

6. Assuming that \( f_1(n) = O(g_1(n)) \) and \( f_2(n) = O(g_2(n)) \). Find counter examples refute
the following statement: \( f_1(n) - f_2(n) = O(g_1(n) - g_2(n)) \).

Proof

Let \( f_1(n) = a_1 n \) and \( f_2(n) = a_2 n \), then \( f_1(n) \) and \( f_2(n) \) are \( O(n) \).
But \( f_1(n) - f_2(n) = (a_1 - a_2) n \) is not \( O(n - n) = O(0) \).
Hence \( f_1(n) - f_2(n) \) is not \( O(g_1(n) - g_2(n)) \).

7. Find the complexity of the function used to find the \( k \)th smallest integer in an
unordered array of integers

```c
int selectkth(int a[], int k, int n) {
    int i, j, mini, tmp;
    for (i = 0; i < k; i++) {
        mini = i;
        for (j = i+1; j < n; j++)
            if (a[j]<a[mini])
                mini = j;
        tmp = a[i];
        a[i] = a[mini];
        a[mini] = tmp;
    }
    return a[k-1];
}
```

The complexity of \( \text{selectkth}() \) is
\[
(n-1) + (n-2) + \ldots + (n-k) = (2n - k - 1) k / 2 = O(n^2)
\]
8. Determine the complexity of the following implementations of the algorithms for adding, multiplying, and transposing $n \times n$ matrices:

```plaintext
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    a[i][j] = b[i][j] + c[i][j];

for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    for (k = 0; k < n; k++)
      a[i][j] += b[i][k] * c[k][j];

for (i = 0; i < n - 1; i++)
  for (j = i+1; j < n; j++) {
    tmp = a[i][j];
    a[i][j] = a[j][i];
    a[j][i] = tmp;
  }
```

Answer:

The algorithm for adding matrices requires $n^2$ assignments. Note that the counter $i$ for the inner loop does not depend on the counter $j$ for the outer loop and both of them take values $0, \ldots, n - 1$.

All three counters, $i$, $j$ and $k$, in the algorithm for matrix multiplication are also independent of each other, hence the complexity of the algorithms is $n^3$.

To transpose a matrix, $\sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 3 = O(n^2)$ assignments are required.

9. Find the computational complexity for the following four loops:

a. for (cnt1 = 0, i = 1; i <= n; i++)
   for (j = 1; j <= n; j++)
     cnt1++;

b. for (cnt2 = 0, i = 1; i <= n; i++)
   for (j = 1; j <= i; j++)
     cnt2++;

c. for (cnt3 = 0, i = 1; i <= n; i *= 2)
   for (j = 1; j <= n; j++)
     cnt3++;
d. for (cnt4 = 0, i = 1; i <= n; i *= 2) 
   for (j = 1; j <= i; j++)
     cnt4++;

Answer:

(a) The auto-increment \( cnt++ \) is executed exactly \( n^2 \) times.

(b) \( \sum_{i=1}^{n} i = O(n^2) \);

(c) \( \sum_{i=1}^{\log n} i = \Theta(n \log n) \)

(d) \( \sum_{i=1}^{\log n} 2^i = \Theta(n) \)

10. Find the average case complexity of sequential search in an array if the probability of accessing the last cell equals \( \frac{1}{2} \) the probability of the next to last cell equals \( \frac{1}{4} \), and the probability of locating a number in any of the remaining cells is the same and equal to \( \frac{1}{4(n-2)} \).

Answer:

\[
\frac{1 + \ldots + (n-2)}{4(n-2)} + \frac{n-1}{4} + \frac{n}{2} = \frac{n-1 + 2(n-1) + 4n}{8} = \frac{7n-3}{8}
\]

IMPORTANT NOTE Study problems of exercises in pages 76-78 of chapter 2.
Exact Analysis Rules

1. We assume an arbitrary time unit.
2. Execution of one of the following operations takes time 1:
   a) assignment operation
   b) single I/O operations
   c) single Boolean operations, numeric comparisons
   d) single arithmetic operations
   e) function return
3. Running time of a selection statement (if, switch) is the time for the condition evaluation + the maximum of the running times for the individual clauses in the selection.
4. Loop execution time is the sum, over the number of times the loop is executed, of the body time + time for the loop check and update operations, + time for the loop setup.
   † Always assume that the loop executes the maximum number of iterations possible
5. Running time of a function call is 1 for setup + the time for any parameter calculations + the time required for the execution of the function body.

Analysis Example 1

Given:

```c
for (i = 0; i < n-1; i++) {
    for (j = 0; j < i; j++)
        array[i][j] = 0;
}
```

So, the total time $T(n)$ is given by:

$$T(n) = 1 + \sum_{i=1}^{n-1} \left( 4 + \sum_{j=1}^{i} 3 + 1 \right) + 1 = \frac{3}{2} n^2 + \frac{7}{2} n - 3$$
Analysis Example 2

### Rule 2a: time 1 before loop

```c
Sum = 0;
for (k = 1; k <= n; k = 2*k) {
    for (j = 1; j <= n; j++) {
        Sum++;
    }
}
```

Rules 4, 2c and 2d: time 3 on each iteration of outer loop. plus one more test

Rules 4 and 2a: time 1 on each iteration of outer loop

### Rule 2a: time 1 on each pass of inner loop

The tricky part is that the outer loop will be executed \( \log(n) \) times assuming \( n \) is a power of 2.

So, the total time \( T(n) \) is given by:

\[
T(n) = 2 + \sum_{k=1}^{\log n} \left( 4 + \sum_{j=1}^{n} 3 + 1 \right) + 1 = 3n \log n + 5 \log n + 3
\]

How does this compare to the previous result?

Analysis Example 3

### Rule 2a: time 1 before loop

```c
Sum = 0;
In >> Value;
while ( In ) {
    if ( Value < 0 ) {
        Sum = -Sum;
        Sum = Sum + Value;
    } else {
        Sum = Sum + Value;
    }
    In >> Value;
}
```

Rules 4, 2c and 2d: time 2, if done

Rules 2ad: time 1 on each pass

So, assuming \( n \) input values are received, the total time \( T(n) \) is given by:

\[
T(n) = 2 + \sum_{k=1}^{n} \left( 3 + \max(4,2) \right) + 1 = 7n + 3
\]
Big-O Example

Take the function obtained in the algorithm analysis example earlier:

\[ T(n) = \frac{3}{2} n^2 + \frac{5}{2} n - 3 \]

Intuitively, one should expect that this function grows similarly to \( n^2 \). To show that, we will shortly prove that:

\[ \frac{5}{2} n < 5n^2 \text{ for all } n \geq 1 \quad \text{and that} \quad -3 < n^2 \text{ for all } n \geq 1 \]

Why? Because then we can argue by substitution (of non-equal quantities):

\[ T(n) = \frac{3}{2} n^2 + \frac{5}{2} n - 3 \leq \frac{3}{2} n^2 + 5n^2 + n^2 = \frac{15}{2} n^2 \text{ for all } n \geq 1 \]

Thus, applying the definition with \( C = 15/2 \) and \( N = 1 \), \( T(n) \) is \( O(n^2) \).