Example. Consider the following code fragment.

```plaintext
for (i = 0; i < n; i++)
    for (j = 0; j < i; j=j+10)
        print ("run time analysis")
```

Give a tight bound on the running time of this code fragment.

Solution. For each value of $i$, the inner loop executes $i/10$ times. Thus the running time of the body of the outer loop is at most $c(i/10)$, for some positive constant $c$. Hence the total running time of the code fragment is given by

$$
\sum_{i=0}^{n-1} c \left\lfloor \frac{i}{10} \right\rfloor \leq \sum_{i=0}^{n-1} c \left( \frac{i}{10} + 1 \right) = \frac{c(n-1)n}{20} + cn \leq 2cn^2 = O(n^2)
$$

We will now show that $\sum_{i=0}^{n-1} c \left\lfloor \frac{i}{10} \right\rfloor = \Omega(n^2)$. Note that

$$
\sum_{i=0}^{n-1} c \left\lfloor \frac{i}{10} \right\rfloor \geq \sum_{i=0}^{n-1} \frac{ci}{10} = c(n-1)n/20
$$

We want to find positive constants $c'$ and $n_0$, such that for all $n \geq n_0$,

$$
\frac{c(n-1)n}{20} \geq c'n^2
$$

This is equivalent to showing that $n(c - 20c') \geq c$. This is true when $c' = c/40$ and $n \geq 2$. Thus, the running time of the code fragment is $\Omega(n^2)$.

Example. Consider the following code fragment.

```plaintext
i = n
while (i >= 10) do
    i = i/3
for j = 1 to n do
    print ("Inner loop")
```

What is an upper-bound on the running time of this code fragment? Is there a matching lower-bound?
**Solution.** The running time of the body of the inner loop is $O(1)$. Thus the running time of the inner loop is at most \(c_1n\), for some positive constant \(c_1\). The body of the outer loop takes at most \(c_2n\) time, for some positive constant \(c_2\) (note that the statement \(i = i/3\) takes \(O(1)\) time). Suppose the algorithm goes through \(t\) iterations of the while loop. At the end of the last iteration of the while loop, the value of \(i\) is \(n/3^t\). We know that the code fragment surely finishes when \(n/3^t \leq 1\), solving which gives us \(t \geq \log_3 n\). This means that the number of iterations of the while loop is at most \(O(\log n)\). Thus the total running time is \(O(n \log n)\).

We will now show that the running time is \(\Omega(n \log n)\). We will lower-bound the number of iterations of the outer loop. Note that when the value of \(i\) is more than 10 (say, \(3^3\)), the outer loop has not terminated. Solving \(n/3^t \geq 3^3\), gives us that \(\log_3 n - 3\) is a lower bound on the number of iterations of the outer loop. For each iteration of the outer loop, the inner loop runs \(n\) times. Thus the total running time is at least \(cn(\log_3 n - 3)\), for some positive constant \(c\). Note that \(cn(\log_3 n - 3) \geq c'n \log n\), when \(c' = c/2\) and \(n \geq 3^6\). Thus the running time is \(\Omega(n \log n)\).

**Example.** Consider the following code fragment.

```plaintext
for i = 0 to n do
    for j = n down to 0 do
        for k = 1 to j-i do
            print (k)
```

What is an upper-bound on the running time of this algorithm? What is the lower bound?

**Solution.** Note that for a fixed value of \(i\) and \(j\), the innermost loop goes through \(\max\{0, j - i\} \leq n\) times. Thus the running time of the above code fragment is \(O(n^3)\).

To find the lower bound on the running time, consider the values of \(i\), such that \(0 \leq i \leq n/4\) and values of \(j\), such that \(3n/4 \leq j \leq n\). Note that for each of the \(n^2/16\) different combinations of \(i\) and \(j\), the innermost loop executes at least \(n/2\) times. Thus the running time is at least

\[(n^2/16)(n/2) = \Omega(n^3)\]

**Example.** Consider the following code fragment.

```plaintext
for i = 1 to n do
    for j = 1 to i*i do
        for k = 1 to j do
            print (k)
```

Give a tight-bound on the running time of this algorithm? We will assume that \(n\) is a power of 2.

**Solution.** Note that the value of \(j\) in the second for-loop is upper bounded by \(n^2\) and the value of \(k\) in the innermost loop is also bounded by \(n^2\). Thus the outermost for-loop iterates for \(n\) times, the second for-loop iterates for at most \(n^2\) times, and the innermost loop iterates for at most \(n^2\) times. Thus the running time of the code fragment is \(O(n^5)\).

We will now argue that the running time of the code fragment is \(\Omega(n^5)\). Consider the following code fragment.
for \( i = n/2 \) to \( n \) do
  for \( j = (n/4) \cdot (n/4) \) to \( (n/2) \cdot (n/2) \) do
    for \( k = 1 \) to \( (n/4) \cdot (n/4) \) do
      print \( (k) \)

Note that the values of \( i, j, k \) in the above code fragment form a subset of the corresponding values in the code fragment in question. Thus the running time of the new code fragment is a lower bound on the running time of the code fragment in question. Thus the running time of the code fragment in question is at least \( n/2 \cdot 3n^2/16 \cdot n^2/16 = \Omega(n^5) \).

Thus the running time of the code fragment in question is \( \Theta(n^5) \).

**Example.** Consider the following code fragment. We will assume that \( n \) is a power of 2.

for \( (i = 1; i <= n; i = 2*i) \) do
  for \( j = 1 \) to \( i \) do
    print \( (j) \)

Give a tight-bound on the running time of this algorithm?

**Solution.** Observe that for \( 0 \leq k \leq \lg n \), in the \( k^{th} \) iteration of the outer loop, the value of \( i = 2^k \). Thus the running time \( T(n) \) of the code fragment can be written as follows.

\[
T(n) = \sum_{k=0}^{\lg n} 2^k \\
= 2^{\lg n + 1} - 1 \\
= 2n - 1 \\
= \Theta(n) \quad (c_1 = 1, c_2 = 2, n_0 = 1)
\]

What is wrong with the following argument that the running time of the above code fragment is \( \Omega(n \log n) \)?

for \( (i = n/128; i <= n; i = 2*i) \) do
  for \( j = 1 \) to \( n/128 \) do
    print \( (j) \)

The outer loop runs \( \Omega(\log n) \) times and the inner loop runs \( \Omega(n) \) times and hence the running time is \( \Omega(n \log n) \).

**Discussion:** Consider a problem \( X \) with an algorithm \( A \).

- Algorithm \( A \) runs in time \( O(n^2) \). This means that the worst case asymptotic running time of algorithm \( A \) is upper-bounded by \( n^2 \). Is this bound tight? That is, is it possible that the run-time analysis of algorithm \( A \) is loose and that one can give a tighter upper-bound on the running time?

- Algorithm \( A \) runs in time \( \Theta(n^2) \). This means that the bound is tight, that is, a better (tighter) bound on the worst case asymptotic running time for algorithm \( A \) is not possible.

- Problem \( X \) takes time \( O(n^2) \). This means that there is an algorithm that solves problem \( X \) on all inputs in time \( O(n^2) \).
• Problem $X$ takes $\Theta(n^{1.5})$. This means that there is an algorithm to solve problem $X$ that takes time $O(n^{1.5})$ and no algorithm can do better.

**Logarithm Facts:** Below are some facts on logarithms that you may find useful.

i. $\log_a b = \frac{1}{\log_b a}$

ii. $\log_a b = \frac{\log_c b}{\log_c a}$

iii. $a^{\log_a b} = b$

iv. $b^{\log_a x} = x^{\log_a b}$