16. We will count comparisons of elements in the list to z. (This ignores comparisons of subscript, but since we are only interested in a bit-O analysis, no harm is done.) Furthermore, we will assume that the number of elements still being examined (those subscripts whose values are from $i$ to $j$) is a factor of 4. Therefore after $i$ elements still being examined, the number of elements will be divided by 4. Thus, the number of elements in the list will be divided by 4 when $i = j$. The latter step terminates the algorithm when $i = j$, as desired. Therefore, the number of comparisons needed is $O(n)$. Therefore, the number of comparisons needed is $O(n^3)$. The latter step terminates the algorithm when $i = j$, as desired. Therefore, the number of comparisons needed is $O(n^3)$. Therefore, the number of comparisons needed is $O(n^3)$.

18. The algorithm requires 2 comparisons to find the next element in the list, then $n - 1$ comparisons to find the least element among the remaining elements, and one more, thus the total number of comparisons is $n - 1 + (n - 2) + \cdots + 3 + 2 + 1 - 1 = n(n - 1)/2$, which is $O(n^2)$.

20. The algorithm we gave is clearly of linear time complexity, i.e., $O(n)$, since we were able to keep updating the value of the minimum element, rather than recomputing it each time. This applies in all cases, not just the worst case.

23. There is a formula length of work on each turn, leading to a total of $2n$ comparisons. Thus the complexity is $O(n^2)$.

24. It takes $n - 1$ comparisons to find the least element in the list. Then $n - 1$ comparisons to find the least element among the remaining elements, and one more, thus the total number of comparisons is $n - 1 + (n - 2) + \cdots + 3 + 2 + 1 - 1 = n(n - 1)/2$, which is $O(n^2)$.

26. Each iteration (determining whether we can move $a$ to $a$) of a given location takes a bounded amount of time, and these are at most $n$ iterations, since each iteration decreases the number of elements remaining. Therefore, there are $O(n)$ comparisons.

28. The bubble sort algorithm works on a list of length $n$, and $(n^2)/2 = O(n^2)$ comparisons for a list of length $2n$. Therefore the number of comparisons goes up by a factor of 4.

29. a) The bubble sort algorithm works on a list of length $n$, and $(n^2)/2 = O(n^2)$ comparisons for a list of length $2n$. Therefore the number of comparisons increases by a factor of 4.

b) The analysis is the same as for bubble sort.

c) The analysis is the same as for bubble sort.

d) The linear least-squares algorithm works on a list of length $n$, where $C$ is a constant. Therefore, we have $C + 2n \log_2 n + 3 = O(n \log n)$ -- the linear complexity of the algorithm.

32. a) $\gamma(1) = \gamma(0) = 0$.

b) $\gamma(1) = \gamma(0) = 0$.

33. a) $\gamma(z) = \gamma(0) = 0$.

b) $\gamma(z) = \gamma(0) = 0$.

34. a) $\gamma(z) = \gamma(0) = 0$.

b) $\gamma(z) = \gamma(0) = 0$.

35. a) $\gamma(z) = \gamma(0) = 0$.

b) $\gamma(z) = \gamma(0) = 0$.

36. a) $\gamma(z) = \gamma(0) = 0$.

b) $\gamma(z) = \gamma(0) = 0$.
22. From \( a \equiv b \pmod{m} \) we know that \( b = a + sm \) for some integer \( s \). Multiplying by \( c \) we have \( bc = ac + smc \), which means that \( ac \equiv bc \pmod{mc} \).

24. Write \( n = 2k + 1 \) for some integer \( k \). Then \( n^2 = (2k + 1)^2 = 4k^2 + 4k + 1 = 4(k + 1) + 1 \). Since either \( k \) or \( k + 1 \) is even, \( 4(k + 1) \) is a multiple of 8. Therefore \( n^2 - 1 \) is a multiple of 8, so \( n^2 \equiv 1 \pmod{8} \).

26. In each case we need to compute \( k \mod{101} \) by dividing by 101 and finding the remainders. This can be done with a calculator that keeps 13 digits of accuracy internally. Just divide the number by 101, subtract off the integer part of the answer, and multiply the fraction that remains by 101. The result will be almost exactly an integer, and that integer is the answer.

   a) 58  b) 60  c) 52  d) 3

28. We just calculate using the formula. We are given \( x_0 = 3 \). Then \( x_1 = (4 \cdot 3 + 1) \mod{7} = 13 \mod{7} = 6; \)
\( x_2 = (4 \cdot 6 + 1) \mod{7} = 25 \mod{7} = 4; \) \( x_3 = (4 \cdot 4 + 1) \mod{7} = 17 \mod{7} = 3 \). At this point, the sequence must continue to repeat 3, 6, 4, 3, 6, 4, ... forever.

30. We assume that the input to this procedure consists of a modulus \( m \geq 2 \), a multiplier \( (a) \), an increment \( (c) \), a seed \( (x_0) \), and the number \( (n) \) of pseudorandom numbers desired. The output will be the sequence \( \{x_i\} \).

    procedure pseudorandom\( (m, a, c, x_0, n) \) : nonnegative integers
    for \( i := 1 \) to \( n \)
    \( x_i := (ax_{i-1} + c) \mod{m} \)

32. We just need to “subtract 3” from each letter. For example, E goes down to B, and B goes down to Y.

   a) BLUE JEANS  b) TEST TODAY  c) EAT DIM SUM

34. We know that \( 1 \cdot 0 + 2 \cdot 3 + 3 \cdot 2 + 4 \cdot 1 + 5 \cdot 2 + 6 \cdot 3 + 7 \cdot Q + 8 \cdot 0 + 9 \cdot 7 + 10 \cdot 2 \equiv 0 \pmod{11} \). This simplifies to \( 127 + 7Q \equiv 0 \pmod{11} \). We subtract 127 from both sides and simplify to \( 7Q \equiv 5 \pmod{11} \), since \( -127 = -11 \cdot 11 + 4 \). It is now a simple matter to use trial and error (or the methods to be introduced in Section 3.7) to find that \( Q = 7 \) (since \( 49 \equiv 5 \pmod{11} \)).