**3.1** For each of the six expressions of Figure 3.1, give the range of values of n for which that expression is most efﬁcient.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **n** |  |  |  |  |  |  |
| **1** | 1 | 2 | 2 | **0** | 20 | 10 |
| **2** | 2 | 4 | 8 | 3.010 | 40 | 20 |
| **3** | 6 | 8 | 18 | 7.156 | 60 | 30 |
| **4** | 24 | 16 | 32 | 12.041 | 80 | 40 |
| **5** | 120 | 32 | 50 | 17.474 | 100 | 50 |
| **6** | 720 | 64 | 72 | 23.344 | 120 | 60 |
| **7** | 5,040 | 128 | 98 | 29.578 | 140 | 70 |
| **8** | 40,320 | 256 | 128 | 36.123 | 160 | 80 |
| **9** | 362,880 | 512 | 168 | 42.940 | 180 | 90 |
| **10** | 3,628,800 | 1,024 | 200 | 50 | 200 | 100 |
| **50** |  |  | 5,000 | 424.742 | 1,000 | 500 |
| **100** |  |  | 20,000 | 1000 | 2,000 | 1,000 |
| **500** |  |  | 500,000 | 6,747.425 | 10,000 | 5,000 |
| **1,000** |  |  | 2,000,000 | 15,000 | 20,000 | 10,000 |
|  |  |  |  |  |  |  |
| **Range** | [2,3] | X | X | [1,4,5,6,…,100] | X |  |

**3.2** Graph the following expressions. For each expression, state the range of values of n for which that expression is the most efﬁcient.

X

[1,..,8]

X

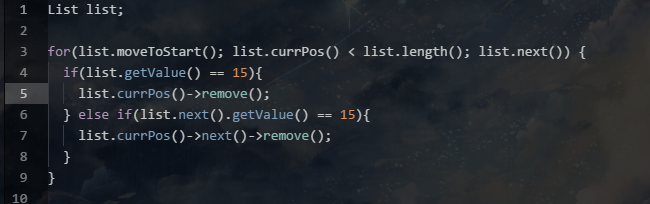
X

[8,)

[1]

X

**4.1** Assume a list has the following conﬁguration: <|2, 23, 15, 5, 9 >. Write a series of C++ statements using the List ADT of Figure4.1 to delete the element with value 15.



**4.2** Show the list conﬁguration resulting from each series of list operations using the List ADT of Figure 4.1. Assume that lists L1 and L2 are empty at the beginning of each series. Show where the current position is in the list.

(a) L1.append(10);

L1.append(20);

L1.append(15);

|  |  |
| --- | --- |
|  | curr |
| L1: | 10 |

|  |  |  |
| --- | --- | --- |
|  |  | curr |
| L1: | 10 | 20 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | curr |
| L1: | 10 | 20 | 15 |

(b) L2.append(10);

L2.append(20);

L2.append(15);

L2.moveToStart();

L2.insert(39);

L2.next();

L2.insert(12);

|  |  |
| --- | --- |
|  | curr |
| L2: | 10 |

|  |  |  |
| --- | --- | --- |
|  |  | curr |
| L2: | 10 | 20 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | curr |
| L2: | 10 | 20 | 15 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | curr |  |  |
| L2: | 10 | 20 | 15 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | curr |  |  |  |
| L2: | 39 | 10 | 20 | 15 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | curr |  |  |
| L2: | 39 | 10 | 20 | 15 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | curr |  |  |  |
| L2: | 39 | 12 | 10 | 20 | 15 |

**4.3** Write a series of C++ statements that uses the List ADT of Figure 4.1 to create a list capable of holding twenty elements and which actually stores the list with the following conﬁguration.

list L1(20);

L1.append(2);

L1.append(23);

L1.append(15);

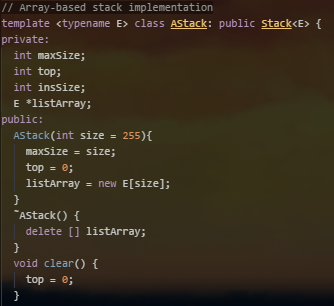
L1.append(5);

L1.append(9);

L1.next();

L1.next();

**4.6** Modify the code of Figure 4.18 to support storing variable-length strings of at most 255 characters. The stack array should have type char. A string is represented by a series of characters (one character per stack element), with the length of the string stored in the stack element immediately above the string itself, as illustrated by Figure 4.35. The **push** operation would store an element requiring *i* storage units in the *i* positions beginning with the current value of **top** and store the size in the position *i* storage units above **top**. The value of **top** would then be reset above the newly inserted element. The **pop** operation need only look at the size value stored in position top−1 and then pop off the appropriate number of units. You may store the string on the stack in reverse order if you prefer, provided that when it is popped from the stack, it is returned in its proper order.





**4.7** Deﬁne an ADT for a bag (see Section 2.1) and create an array-based implementation for bags. Be sure that your bag ADT does not rely in any way on knowing or controlling the position of an element. Then, implement the dictionary ADT of Figure 4.28 using your bag implementation.

