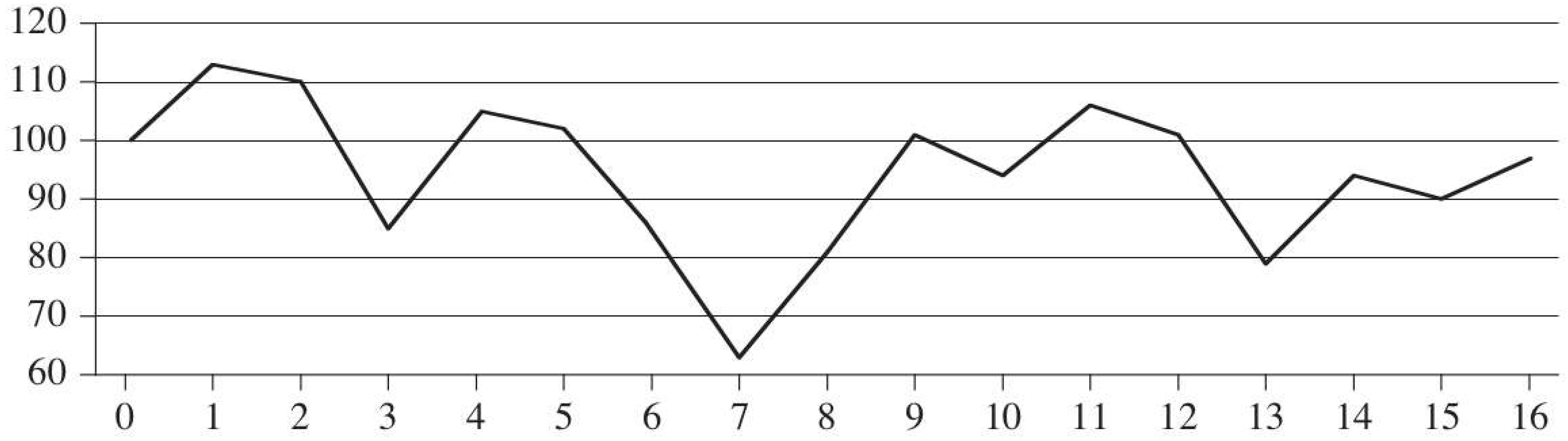

Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

Figure 4.1 Information about the price of stock in the Volatile Chemical Corporation after the close of trading over a period of 17 days. The horizontal axis of the chart indicates the day, and the vertical axis shows the price. The bottom row of the table gives the change in price from the previous day.

```
MaxSoFar := 0.0
for L := 1 to N do
     for U := L to N do
          Sum := 0.0
          for I := L to U do
               Sum := Sum + X[I]
          /* Sum now contains the
                sum of X[L..U] */
          MaxSoFar := max(MaxSoFar, Sum)
```

Algorithm 1. The cubic algorithm

```
MaxSoFar := 0.0
for L := 1 to N do
    Sum := 0.0
    for U := L to N do
        Sum := Sum + X[I]
        /* Sum now contains the
             sum of X[L..U] */
        MaxSoFar := max(MaxSoFar, Sum)
```

Algorithm 2. The first quadratic algorithm

```
CumArray[0] := 0.0
for I := 1 to N do
    CumArray[I] := CumArray[I - 1] + X[I]
MaxSoFar := 0.0
for L := 1 to N do
    for U := L to N do
        Sum := CumArray[U] - CumArray[L - 1]
        /* Sum now contains the
             sum of X[L..U] */
        MaxSoFar := max(MaxSoFar, Sum)
```

Algorithm 2b. An alternative quadratic algorithm

```
recursive function MaxSum(L, U)
    if L > U then     /* Zero-element vector */
         return 0.0
    if L = U then     /* One-element vector */
         return max(0.0, X[L])

    M := (L + U)/2     /* A is X[L..M], B is X[M + 1..U] */
    /* Find max crossing to left */
        Sum := 0.0; MaxToLeft := 0.0
        for I := M downto L do
            Sum := Sum + X[I]
            MaxToLeft := max(MaxToLeft, Sum)
    /* Find max crossing to right */
        Sum := 0.0; MaxToRight := 0.0
        for I := M + 1 to U do
            Sum := Sum + X[I]
            MaxToRight := max(MaxToRight, Sum)
    MaxCrossing := MaxToLeft + MaxToRight

    MaxInA := MaxSum(L, M)
    MaxInB := MaxSum(M + 1, U)
    return max(MaxCrossing, MaxInA, MaxInB)
```

Algorithm 3. A divide-and-conquer algorithm

```
MaxSoFar := 0.0
MaxEndingHere := 0.0
for I := 1 to N do
    MaxEndingHere := max(0.0,
        MaxEndingHere + X[I])
    MaxSoFar := max(MaxSoFar,
        MaxEndingHere)
```

Algorithm 4. The linear algorithm