

subproblems become sufficiently small. Consider a modification to merge sort in which n/k sublists of length k are sorted using insertion sort and then merged using the standard merging mechanism, where k is a value to be determined.

- a. Show that insertion sort can sort the n/k sublists, each of length k , in $\Theta(nk)$ worst-case time.
- b. Show how to merge the sublists in $\Theta(n \lg(n/k))$ worst-case time.
- c. Given that the modified algorithm runs in $\Theta(nk + n \lg(n/k))$ worst-case time, what is the largest value of k as a function of n for which the modified algorithm has the same running time as standard merge sort, in terms of Θ -notation?
- d. How should we choose k in practice?

2-2 Correctness of bubblesort

Bubblesort is a popular, but inefficient, sorting algorithm. It works by repeatedly swapping adjacent elements that are out of order.

BUBBLESORT(A)

```

1  for  $i = 1$  to  $A.length - 1$ 
2      for  $j = A.length$  downto  $i + 1$ 
3          if  $A[j] < A[j - 1]$ 
4              exchange  $A[j]$  with  $A[j - 1]$ 
```

- a. Let A' denote the output of BUBBLESORT(A). To prove that BUBBLESORT is correct, we need to prove that it terminates and that

$$A'[1] \leq A'[2] \leq \dots \leq A'[n], \quad (2.3)$$

where $n = A.length$. In order to show that BUBBLESORT actually sorts, what else do we need to prove?

The next two parts will prove inequality (2.3).

- b. State precisely a loop invariant for the **for** loop in lines 2–4, and prove that this loop invariant holds. Your proof should use the structure of the loop invariant proof presented in this chapter.
- c. Using the termination condition of the loop invariant proved in part (b), state a loop invariant for the **for** loop in lines 1–4 that will allow you to prove inequality (2.3). Your proof should use the structure of the loop invariant proof presented in this chapter.

- d. What is the worst-case running time of bubblesort? How does it compare to the running time of insertion sort?

2-3 Correctness of Horner's rule

The following code fragment implements Horner's rule for evaluating a polynomial

$$\begin{aligned} P(x) &= \sum_{k=0}^n a_k x^k \\ &= a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + xa_n) \cdots)) , \end{aligned}$$

given the coefficients a_0, a_1, \dots, a_n and a value for x :

```

1  y = 0
2  for i = n downto 0
3      y = ai + x · y

```

- a. In terms of Θ -notation, what is the running time of this code fragment for Horner's rule?
- b. Write pseudocode to implement the naive polynomial-evaluation algorithm that computes each term of the polynomial from scratch. What is the running time of this algorithm? How does it compare to Horner's rule?
- c. Consider the following loop invariant:

At the start of each iteration of the **for** loop of lines 2–3,

$$y = \sum_{k=0}^{n-(i+1)} a_{k+i+1} x^k .$$

Interpret a summation with no terms as equaling 0. Following the structure of the loop invariant proof presented in this chapter, use this loop invariant to show that, at termination, $y = \sum_{k=0}^n a_k x^k$.

- d. Conclude by arguing that the given code fragment correctly evaluates a polynomial characterized by the coefficients a_0, a_1, \dots, a_n .

2-4 Inversions

Let $A[1 \dots n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an ***inversion*** of A .

- a. List the five inversions of the array $\langle 2, 3, 8, 6, 1 \rangle$.