duce the time taken by the combine step to $\Theta(n)$, then the time complexity of the algorithm will be $\Theta(n \log n)$. This can be achieved by a process called presorting, i.e., the elements in $S$ are sorted by their $y$-coordinates once and for all and stored in an array $Y$. Each time we need to sort $T$ in the combine step, we only need to extract its elements from $Y$ in $\Theta(n)$ time. This is easy to do since the points in $T$ are those points in $Y$ within distance $\delta$ from the vertical line $L$. This modification reduces the time required by the combine step to $\Theta(n)$. Thus, the recurrence relation becomes

$$T(n) = \begin{cases} 
1 & \text{if } n = 2 \\
3 & \text{if } n = 3 \\
2T(n/2) + \Theta(n) & \text{if } n > 3.
\end{cases}$$

The solution to this familiar recurrence is the desired $\Theta(n \log n)$ bound.

The above discussion implies Algorithm closestpair. In the algorithm, for a point $p$, $x(p)$ denotes the $x$-coordinate of point $p$.

The following theorem summarizes the main result. Its proof is embedded in the description of the algorithm and the analysis of its running time.

**Theorem 6.7** Given a set $S$ of $n$ points in the plane, Algorithm closestpair finds a pair of points in $S$ with minimum separation in $\Theta(n \log n)$ time.

### 6.10 Exercises

6.1. Modify Algorithm minmax so that it works when $n$ is not a power of 2. Is the number of comparisons performed by the new algorithm $\lceil 3n/2 - 2 \rceil$ even if $n$ is not a power of 2? Prove your answer.

6.2. Consider Algorithm slowminmax which is obtained from Algorithm minmax by replacing the test

if high − low = 1

by the test

if high = low

and making some other changes in the algorithm accordingly. Thus, in Algorithm slowminmax, the recursion is halted when the size of the input array is 1. Count the number of comparisons required by this algorithm to find the minimum and maximum of an array $A[1..n]$, where

$$T(n) = \begin{cases} 
1 & \text{if } n = 2 \\
3 & \text{if } n = 3 \\
2T(n/2) + \Theta(n) & \text{if } n > 3.
\end{cases}$$
Algorithm 6.7  closestpair

**Input:** A set \( S \) of \( n \) points in the plane.

**Output:** The minimum separation realized by two points in \( S \).

1. Sort the points in \( S \) in nondecreasing order of their \( x \)-coordinates.
2. \( Y \leftarrow \) The points in \( S \) sorted in nondecreasing order of their \( y \)-coordinates.
3. \( \delta \leftarrow \text{cp}(1, n) \)

Procedure \( \text{cp}(\text{low}, \text{high}) \)

1. if \( \text{high} - \text{low} + 1 \leq 3 \) then compute \( \delta \) by a straightforward method.
2. else
3. \( \text{mid} \leftarrow \lfloor (\text{low} + \text{high})/2 \rfloor \)
4. \( x_0 \leftarrow x(S[\text{mid}]) \)
5. \( \delta_l \leftarrow \text{cp}(\text{low}, \text{mid}) \)
6. \( \delta_r \leftarrow \text{cp}(\text{mid} + 1, \text{high}) \)
7. \( \delta \leftarrow \min\{\delta_l, \delta_r\} \)
8. \( k \leftarrow 0 \)
9. for \( i \leftarrow 1 \) to \( n \) \{ Extract \( T \) from \( Y \) \}
10. if \( |x(Y[i]) - x_0| \leq \delta \) then
11. \( k \leftarrow k + 1 \)
12. \( T[k] \leftarrow Y[i] \)
13. end if
14. end for \{ \( k \) is the size of \( T \) \}
15. \( \delta' \leftarrow 2\delta \) \{ Initialize \( \delta' \) to any number greater than \( \delta \) \}
16. for \( i \leftarrow 1 \) to \( k - 1 \) \{ Compute \( \delta' \) \}
17. for \( j \leftarrow i + 1 \) to \( \min\{i + 7, k\} \)
18. if \( d(T[i], T[j]) < \delta' \) then \( \delta' \leftarrow d(T[i], T[j]) \)
19. end for
20. end for
21. \( \delta \leftarrow \min\{\delta, \delta'\} \)
22. end if
23. return \( \delta \)

\( n \) is a power of 2. Explain why the number of comparisons in this algorithm is greater than that in Algorithm minmax. (Hint: In this case, the initial condition is \( C(1) = 0 \)).

6.3. Derive an iterative minimax algorithm that finds both the minimum and maximum in a set of \( n \) elements using only \( 3n/2 - 2 \) comparisons, where \( n \) is a power of 2.

6.4. Give a divide-and-conquer version of Algorithm linearsearch given in Sec. 1.3. The algorithm should start by dividing the input elements into approximately two halves. How much work space is required by the algorithm?
6.5. Give a divide-and-conquer algorithm to find the sum of all numbers in an array \( A[1..n] \) of integers. The algorithm should start by dividing the input elements into approximately two halves. How much work space is required by the algorithm?

6.6. Let \( A[1..n] \) be an array of \( n \) integers and \( x \) an integer. Derive a divide-and-conquer algorithm to find the frequency of \( x \) in \( A \), i.e., the number of times \( x \) appears in \( A \). What is the time complexity of your algorithm?

6.7. Modify Algorithm `binarysearchrec` so that it searches for two keys. In other words, given an array \( A[1..n] \) of \( n \) elements and two elements \( x_1 \) and \( x_2 \), the algorithm should return two integers \( k_1 \) and \( k_2 \) representing the positions of \( x_1 \) and \( x_2 \), respectively, in \( A \).

6.8. Design a search algorithm that divides a sorted array into one third and two thirds instead of two halves as in Algorithm `binarysearchrec`. Analyze the time complexity of the algorithm.

6.9. Modify Algorithm `binarysearchrec` so that it divides the sorted array into three equal parts instead of two as in Algorithm `binarysearchrec`. In each iteration, the algorithm should test the element \( x \) to be searched for against two entries in the array. Analyze the time complexity of the algorithm.

6.10. Use Algorithm `mergesort` to sort the array

(a) \[
\begin{array}{cccccccc}
32 & 15 & 14 & 15 & 11 & 17 & 25 & 51 \\
\end{array}
\]

(b) \[
\begin{array}{cccccccc}
12 & 25 & 17 & 19 & 51 & 32 & 45 & 18 & 22 & 37 & 15 \\
\end{array}
\]

6.11. Use mathematical induction to prove the correctness of Algorithm `mergesort`. Assume that Algorithm `merge` works correctly.

6.12. Show that the space complexity of Algorithm `mergesort` is \( \Theta(n) \).

6.13. It was shown in Sec. 6.3 that algorithms `bottomupsort` and `mergesort` are very similar. Given an example of an array of numbers in which

(a) Algorithm `bottomupsort` and Algorithm `mergesort` perform the same number of element comparisons.

(b) Algorithm `bottomupsort` performs more element comparisons than Algorithm `mergesort`.

(c) Algorithm `bottomupsort` performs fewer element comparisons than Algorithm `mergesort`.

6.14. Consider the following modification of Algorithm `mergesort`. The algorithm first divides the input array \( A[low..high] \) into four parts \( A_1, A_2, A_3 \) and \( A_4 \) instead of two. It then sorts each part recursively, and finally merges the four sorted parts to obtain the original array in sorted order. Assume for simplicity that \( n \) is a power of 4.

(a) Write out the modified algorithm.
(b) Analyze its running time.

6.15. What will be the running time of the modified algorithm in Exercise 6.14 if the input array is divided into $k$ parts instead of 4? Here, $k$ is a fixed positive integer greater than 1.

6.16. Consider the following modification to Algorithm mergesort. We apply the algorithm on the input array $A[1..n]$ and continue the recursive calls until the size of a subinstance becomes relatively small, say $m$ or less. At this point, we switch to Algorithm insertionsort and apply it on the small instance. So, the first test of the modified algorithm will look like the following:

```
if high - low + 1 ≤ m then insertionsort(A[low..high]).
```

What is the largest value of $m$ in terms of $n$ such that the running time of the modified algorithm will still be $\Theta(n \log n)$? You may assume for simplicity that $n$ is a power of 2.

6.17. Use Algorithm select to find the $k$th smallest element in the list of numbers given in Example 6.1, where

(a) $k = 1$. (b) $k = 9$. (c) $k = 17$. (d) $k = 22$. (e) $k = 25$.

6.18. What will happen if in Algorithm select the true median of the elements is chosen as the pivot instead of the median of medians? Explain.

6.19. Let $A[1..105]$ be a sorted array of 105 integers. Suppose we run Algorithm select to find the 17th element in $A$. How many recursive calls to Procedure select will there be? Explain your answer clearly.

6.20. Explain the behavior of Algorithm select if the input array is already sorted in nondecreasing order. Compare that to the behavior of Algorithm binarysearchrec.

6.21. In Algorithm select, groups of size 5 are sorted in each invocation of the algorithm. This means that finding a procedure that sorts a group of size 5 that uses the fewest number of comparisons is important. Show that it is possible to sort five elements using only seven comparisons.

6.22. One reason that Algorithm select is inefficient is that it does not make full use of the comparisons that it makes: After it discards one portion of the elements, it starts on the subproblem from scratch. Give a precise count of the number of comparisons the algorithm performs when presented with $n$ elements. Note that it is possible to sort five elements using only seven comparisons (see Exercise 6.21).

6.23. Based on the number of comparisons counted in Exercise 6.22, determine for what values of $n$ one should use a straightforward sorting method and extract the $k$th element directly.

6.24. Let $g$ denote the size of each group in Algorithm select for some positive integer $g \geq 3$. Derive the running time of the algorithm in terms of $g$. 


What happens when \( g \) is too large compared to the value used in the algorithm, namely 5?

6.25. Which of the following group sizes 3, 4, 5, 7, 9, 11 guarantees \( \Theta(n) \) worst case performance for Algorithm \textit{select}? Prove your answer. (See Exercise 6.24).

6.26. Rewrite Algorithm \textit{select} using Algorithm \textit{split} to partition the input array. Assume for simplicity that all input elements are distinct. What is the advantage of the modified algorithm?

6.27. Let \( A[1..n] \) and \( B[1..n] \) be two arrays of distinct integers sorted in increasing order. Give an efficient algorithm to find the median of the \( 2n \) elements in both \( A \) and \( B \). What is the running time of your algorithm?

6.28. Make use of the algorithm obtained in Exercise 6.27 to device a divide-and-conquer algorithm for finding the median in an array \( A[1..n] \). What is the time complexity of your algorithm? (Hint: Make use of Algorithm \textit{mergesort}).

6.29. Consider the problem of finding all the first \( k \) smallest elements in an array \( A[1..n] \) of \( n \) distinct elements. Here, \( k \) is not constant, i.e., it is part of the input. We can solve this problem easily by sorting the elements and returning \( A[1..k] \). This, however, costs \( O(n \log n) \) time. Give a \( \Theta(n) \) time algorithm for this problem. Note that running Algorithm \textit{select} \( k \) times costs \( \Theta(kn) = O(n^2) \) time, as \( k \) is not constant.

6.30. Consider the \textit{multiselection} problem: Given a set \( S \) of \( n \) elements and a set \( K \) of \( r \) ranks \( k_1, k_2, \ldots, k_r \), find the \( k_1 \)th, \( k_2 \)th, \ldots, \( k_r \)th smallest elements. For example, if \( K = \{2, 7, 9, 50\} \), the problem is to find the 2nd, 7th, 9th and 50th smallest elements. This problem can be solved trivially in \( \Theta(rn) \) time by running Algorithm \textit{select} \( r \) times, once for each rank \( k_j \), \( 1 \leq j \leq r \). Give an \( O(n \log r) \) time algorithm to solve this problem.

6.31. Apply Algorithm \textit{split} on the array \[27 13 31 18 45 16 17 53\].

6.32. Let \( f(n) \) be the number of element interchanges that Algorithm \textit{split} makes when presented with the input array \( A[1..n] \) excluding interchanging \( A[low] \) with \( A[i] \).

(a) For what input arrays \( A[1..n] \) is \( f(n) = 0 \)?

(b) What is the maximum value of \( f(n) \)? Explain when this maximum is achieved?

6.33. Modify Algorithm \textit{split} so that it partitions the elements in \( A[low..high] \) around \( x \), where \( x \) is the median of \( \{A[low], A[(low + high)/2], A[high]\} \). Will this improve the running time of Algorithm \textit{quicksort}? Explain.

6.34. Algorithm \textit{split} is used to partition an array \( A[low..high] \) around \( A[low] \).
Another algorithm to achieve the same result works as follows. The algorithm has two pointers $i$ and $j$. Initially, $i = \text{low}$ and $j = \text{high}$. Let the pivot be $x = A[\text{low}]$. The pointers $i$ and $j$ move from left to right and from right to left, respectively, until it is found that $A[i] > x$ and $A[j] \leq x$. At this point $A[i]$ and $A[j]$ are interchanged. This process continues until $i \geq j$. Write out the complete algorithm. What is the number of comparisons performed by the algorithm?

6.35. Let $A[1..n]$ be a set of integers. Give an algorithm to reorder the elements in $A$ so that all negative integers are positioned to the left of all nonnegative integers. Your algorithm should run in time $\Theta(n)$.

6.36. Use Algorithm quicksort to sort the array

(a) $\begin{array}{cccccc}24 & 33 & 24 & 45 & 12 & 24 \end{array}$
(b) $\begin{array}{cccc}3 & 4 & 5 & 6 & 7 \end{array}$
(c) $\begin{array}{cccccccccc}23 & 32 & 27 & 18 & 45 & 11 & 63 & 12 & 19 & 16 & 25 & 52 & 14 \end{array}$

6.37. Show that the work space needed by Algorithm quicksort varies between $\Theta(\log n)$ and $\Theta(n)$. What is its average space complexity?

6.38. Explain the behavior of Algorithm quicksort when the input is already sorted in decreasing order. You may assume that the input elements are all distinct.

6.39. Explain the behavior of Algorithm quicksort when the input array $A[1..n]$ consists of $n$ identical elements.

6.40. Modify Algorithm quicksort slightly so that it will solve the selection problem. What is the time complexity of the new algorithm in the worst case and on the average?

6.41. Give an iterative version of Algorithm quicksort.

6.42. Which of the following sorting algorithms are stable (see Exercise 5.14)?
(a) heapsort (b) mergesort (c) quicksort.

6.43. A sorting algorithm is called adaptive if its running time depends not only on the number of elements $n$, but also on their order. Which of the following sorting algorithms are adaptive?
(a) selectionsort (b) insertion sort (c) bubblesort (d) heapsort (e) bottomup sort (f) mergesort (g) quicksort (h) radixsort.

6.44. Let $x = a + bi$ and $y = c + di$ be two complex numbers. The product $xy$ can easily be calculated using four multiplications, that is, $xy = (ac - bd) + (ad + bc)i$. Devise a method for computing the product $xy$ using only three multiplications.

6.45. Write out an algorithm for the traditional algorithm for matrix multiplication described in Sec. 6.8.
6.46. Show that the traditional algorithm for matrix multiplication described in Sec. 6.8 requires $n^3$ multiplications and $n^3 - n^2$ additions (see Exercise 6.45).

6.47. Explain how to modify Strassen’s algorithm for matrix multiplication so that it can also be used with matrices whose size is not necessarily a power of 2.

6.48. Suppose we modify the algorithm for the closest pair problem so that not each point in $T$ is compared with seven points in $T$. Instead, every point to the left of the vertical line $L$ is compared with a number of points to its right.

   (a) What are the necessary modifications to the algorithm?
   (b) How many points to the right of $L$ have to be compared with every point to its left? Explain.

6.49. Rewrite the algorithm for the closest pair problem without the presorting step. The time complexity of your algorithm should be $\Theta(n \log n)$. (Hint: Make use of Algorithm MERGESORT).

6.50. Design a divide-and-conquer algorithm to determine whether two given binary trees $T_1$ and $T_2$ are identical.

6.51. Design a divide-and-conquer algorithm that computes the height of a binary tree.

6.52. Give a divide-and-conquer algorithm to find the second largest element in an array of $n$ numbers. Derive the time complexity of your algorithm.

6.53. Consider the following algorithm that attempts to find a minimum cost spanning tree $MST(G)$ for a weighted undirected graph $G = (V, E)$ (see Sec. 8.3). Divide $G$ into two subgraphs $G_1$ and $G_2$ of approximately the same number of vertices. Compute $T_1 = MST(G_1)$ and $T_2 = MST(G_2)$. Find an edge $e$ of minimum weight that connects $G_1$ with $G_2$. Return $T_1 \cup T_2 \cup \{e\}$. Show that this algorithm does not always compute a spanning tree of minimum weight. What is the shape of the spanning tree computed by the algorithm?

6.54. Let $B$ be an $n \times n$ chessboard, where $n$ is a power of 2. Use a divide-and-conquer argument to describe (in words) how to cover all squares of $B$ except one with L-shaped tiles. For example, if $n = 2$, then there are four squares three of which can be covered by one L-shaped tile, and if $n = 4$, then there are 16 squares of which 15 can be covered by 5 L-shaped tiles.

6.55. Use a combinatorial argument to show that if $n$ is a power of 2, then $n^2 \equiv 1 \pmod{3}$. (Hint: Use the result of Exercise 6.54).