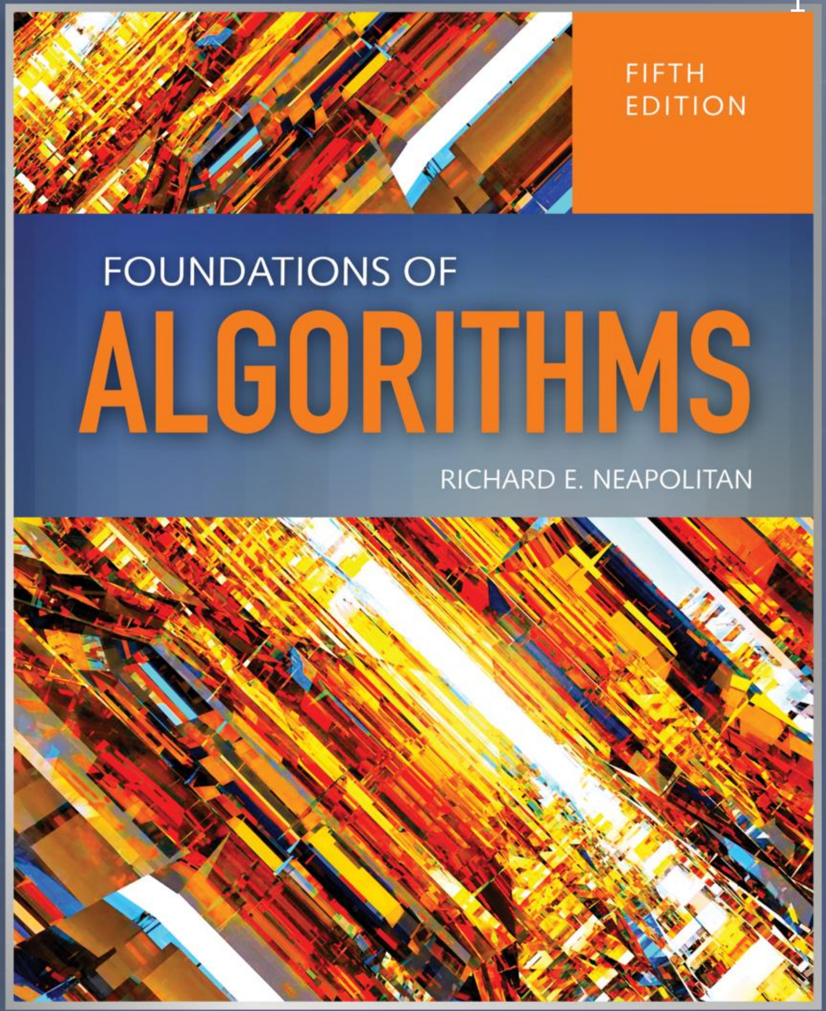


Divide and Conquer

Chapter 2

Objectives

- Describe the divide-and-conquer approach to solving problems
- Apply the divide-and-conquer approach to solve a problem
- Determine when the divide-and-conquer approach is an appropriate solution approach
- Determine complexity analysis of divide and conquer algorithms
- Contrast worst-case and average-case complexity analysis

Battle of Austerlitz – December 2, 1805

- Napoleon split the Austro-Russian Army and was able to conquer 2 weaker armies
- Divide an instance of a problem into 2 or more smaller instances
- Top-down approach

Binary Search

- Locate key x in an array of size n sorted in non-decreasing order
- Compare x with the middle element – if equal, done – quit. Else
  - *Divide* the Array into two sub-arrays approximately half as large
    - If x is smaller than the middle item, select left sub-array
    - If x is larger than the middle item, select right sub-array

Binary Search

- *Conquer* (solve) the sub-array: Is x in the sub-array using recursion until the sub-array is sufficiently small?
- *Obtain* the solution to the array from the solution to the subarray

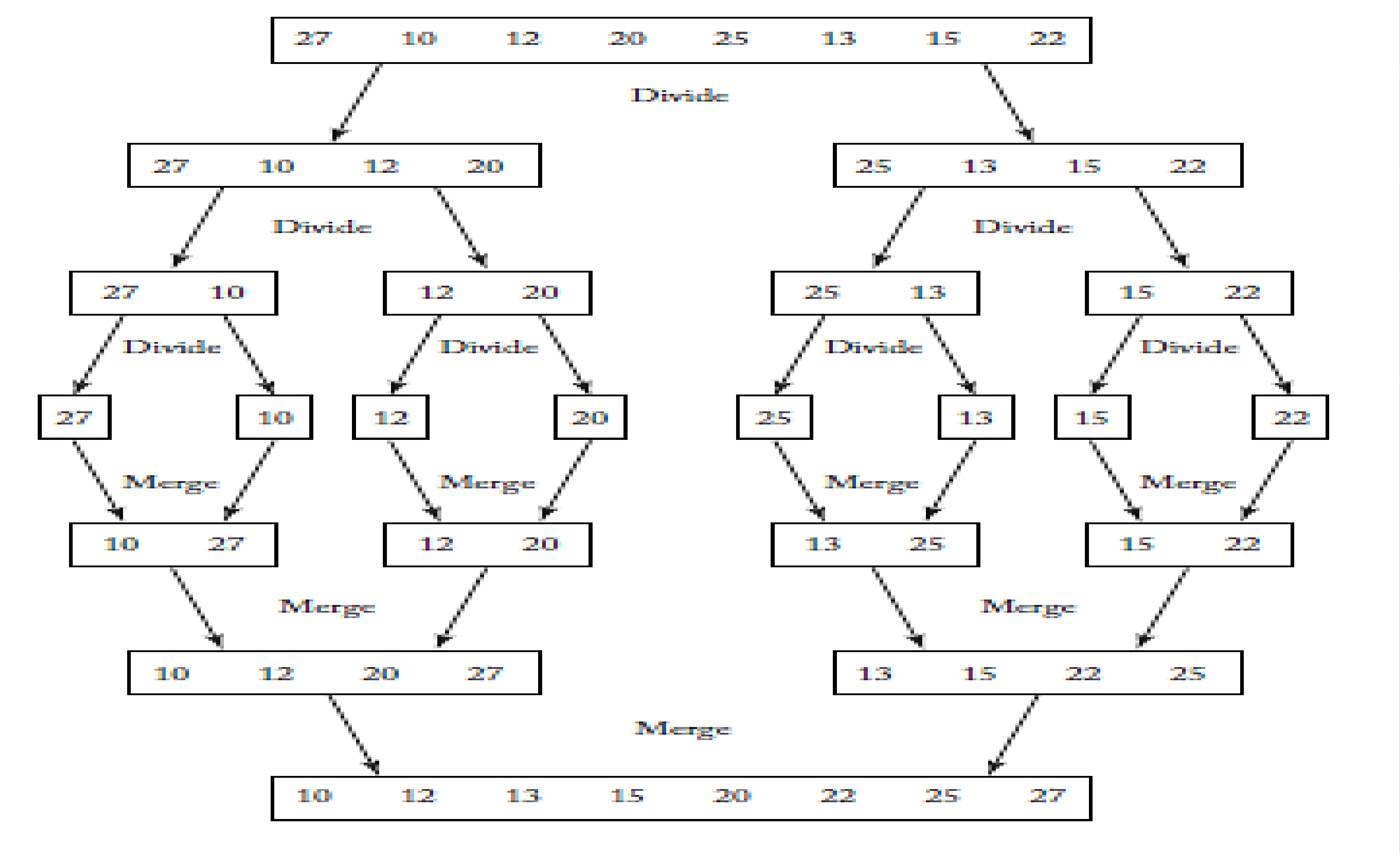
Worst Case Complexity Analysis

- n is a power of 2 and x > S[n]
- W(n) = W(n/2) + 1 for n>1 and n power of 2
- W(1) = 1
  - W(n/2) = the number of comparisons in the recursive call
  - 1 comparison at the top level
- Example B1 in Appendix B:
  - $W(n) = \lg n + 1$
- n not a power of 2
  - $W(n) = \lfloor \lg n \rfloor + 1 \in \theta(\lg n)$

Mergesort

- Sort an array S of size n (for simplicity, let n be a power of 2)
- *Divide* S into 2 sub-arrays of size n/2
- *Conquer* (solve) recursively sort each sub-array until array is sufficiently small (size 1)
- *Combine* merge the solutions to the sub-arrays into a single sorted array

Figure 2.2



27 10 12 20 25 13 15 22
Divide
27 10 12 20
25 13 15 22
Divide
Divide
27 10
12 20
25 13
15 22
Divide
Divide
Divide
Divide
27
10
12
20
25
13
15
22
Merge
Merge
Merge
Merge
10 27
12 20
13 25
15 22
Merge
Merge
10 12 20 27
13 15 22 25
Merge
10 12 13 15 20 22 25 27

Merge

- Merges the two arrays U and V created by the recursive calls to mergesort
- Input size
  - h the number of items in U
  - m the number of items in V
- Basic operation:
  - Comparison of U[I] to V[j]
- Worst case:
  - Loop exited with one index at exit point and the other one less than the exit point

Worst-Case Time Complexity Analysis

- $W(n)$ = time to sort U + time to sort V + time to merge
- $W(n) = W(h) + W(m) + h+m-1$
- First analysis assumes n is a power of 2
  - $h = \lfloor n/2 \rfloor = n/2$
  - $m = n – h = n – n/2 = n/2$
  - $h + m = n/2 + n/2 = n$
- $W(n) = W(n/2) + W(n/2) + n – 1$
- $= 2W(n/2) + n-1$ for $n > 1$ and n a power of 2
- $W(1) = 0$
- From B19 in Appendix B
  - $W(n)=n\lg n-(n-1) \in \theta(n\lg n)$

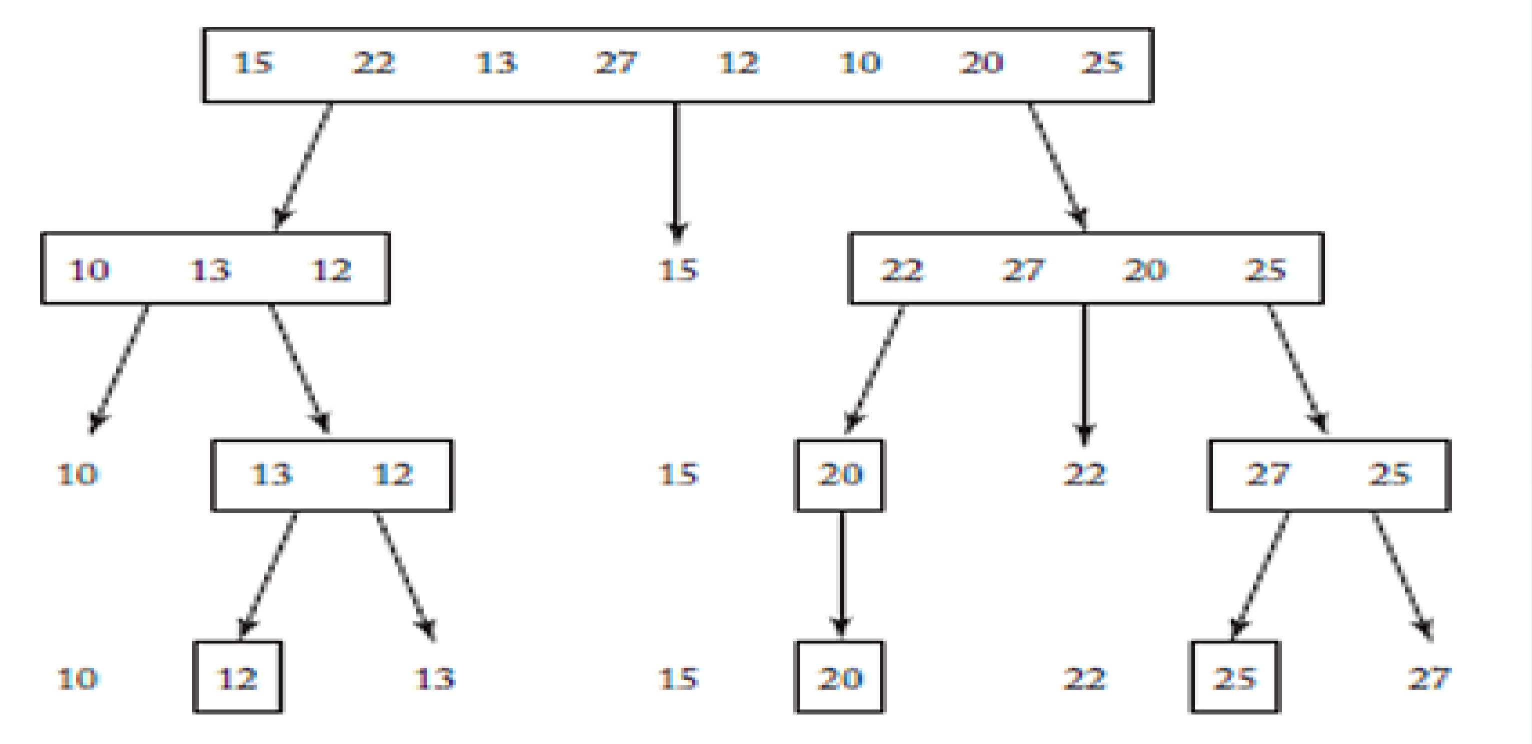
Worst-Case Analysis Mergesort

- n not a power of 2
- $W(n)=W(\lfloor n/2 \rfloor) + W(\lceil n/2 \rceil) + n-1$
- From Theorem B4:
- $W(n) \in \Theta(n\lg n)$

Quicksort

- Array recursively divided into two partitions and recursively sorted
- Division based on a pivot
- pivot divides the two sub-arrays
- All items < pivot placed in sub-array before pivot
- All items >= pivot placed in sub-array after pivot

Figure 2.3


15 22 13 27 12 10 20 25
10 13 12
15
22 27 20 25
10
13 12
15
20
22
27 25
10
12
13
15
20
22
25
27

Basic operation

- Comparison of S[i] with pivotitem
- Input size n

Every-Case Complexity Analysis of Partition

- $T(n) = n - 1$

Worst-Case Complexity Analysis of Quicksort

- Array is sorted in non-decreasing order
- Array is repeatedly sorted into an empty sub-array which is less than the pivot and a sub-array of n-1 containing items greater than pivot
- If there are k keys in the current sub-array, k-1 key comparisons are executed

Worst-Case Complexity Analysis of Quicksort

- T(n) is specified because analysis is for the every-case complexity for the class of instances already sorted in non-decreasing order
- T(n) = time to sort left sub-array + time to sort right sub-array + time to partition
- $T(n) = T(0) + (T(n-1) + n – 1$
- $T(n) = T(n – 1) + n – 1$ for $n > 0$
- $T(0) = 0$
- From B16
  - $T(n) = n(n-1)/2$

Worst Case

- At least n(n-1)/2
- Use induction to show it is the worst case
  - W(n)<=n(n-1)/2

Average-Case Time Complexity of Quicksort

- Value of pivotpoint is equally likely to be any of the numbers from 1 to n
- Average obtained is the average sorting time when every possible ordering is sorted the same number of times
- An= p=1n1n(Ap-1+ An-p+n-1)

$$A(n) = \sum_{p=1}^{n} \frac{1}{n}\left(A(p-1) + A(n-p)\right) + n - 1$$

- 1/n is the probability pivotpoint is p
- Solving equation and using B22
  - A(n) ε θ(nlgn)