Chapter 2.3: Designing Algorithms and Merge sort

- Recursive algorithms are useful and common.
- Divide-and-Conquer is an useful recursive technique for designing algorithms. It consists of three steps:

  Divide the problem into a number of subproblems that are smaller instances of the same problem.

  Conquer the subproblems by solving them recursively. If the subproblems are small enough, just solve the problems directly.

  Combine the solutions of the subproblems into the solution for the original problem.

- **Merge sort** is an example of a divide-and-conquer algorithm.

  Divide the n-element sequence to be sorted into two subsequences of $n/2$ elements each. problem.

  Conquer sort the two subsequences recursively using merge sort.

  Combine by merging the two sorted subsequences to produce a sorted answer.

- The base case for the recursion is when the sequence to be sorted has length 1.
- The key part is to merge to two sorted subsequence. Let's examine the merge procedure show below.

```
MERGE(A, p, q, r)
  n1 = q - p + 1
  n2 = r - q
  let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
  for i = 1 to n1
      L[i] = A[p + i - 1]
  for j = 1 to n2
      R[j] = A[q + j]
  L[n1 + 1] = ∞
  R[n2 + 1] = ∞
  i = 1
  j = 1
  for k = p to r
      if L[i] ≤ R[j]
          A[k] = L[i]
          i = i + 1
      else A[k] = R[j]
          j = j + 1
```

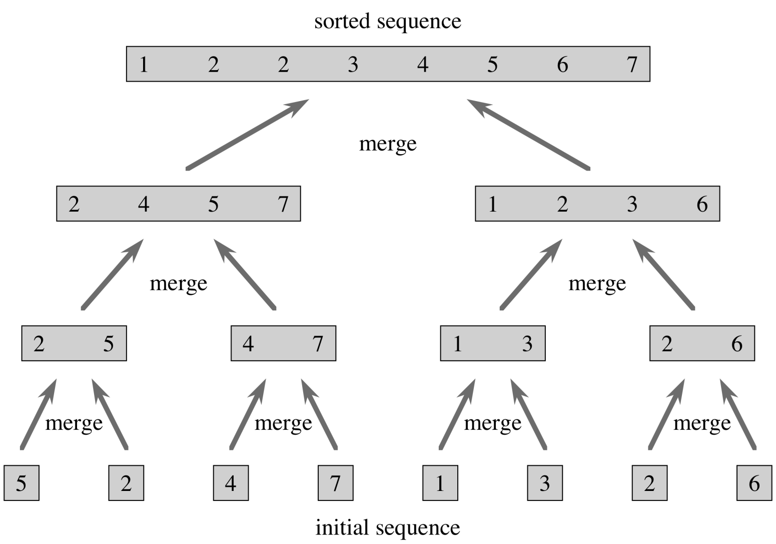
- Observations about the Merge procedure
  - The worst-case run-time is $\Theta(n)$.

- The algorithm is **oblivious** in that its run-time doesn't change due to the instance of the problem.
- The pseudo-code uses sentinels, which is a special value used to simplify code.

- Now we can write out the pseudo-code for the merge sort algorithm:

```
MERGE-SORT(A, p, r)
  if p < r                          // check for base case
      q = ⌊(p + r)/2⌋               // divide
      MERGE-SORT(A, p, q)           // conquer
      MERGE-SORT(A, q + 1, r)       // conquer
      MERGE(A, p, q, r)             // combine
```

- Here is an example:



- Try to run the Merge_Sort to an input $A =< 5, 2, 4, 6, 1, 3, 2, 6 >$.

- Try Exercise 2.3-1.

- The run time of a divide-and-conquer algorithm can be described by a recurrence equation (or just recurrence). Here is the general form:

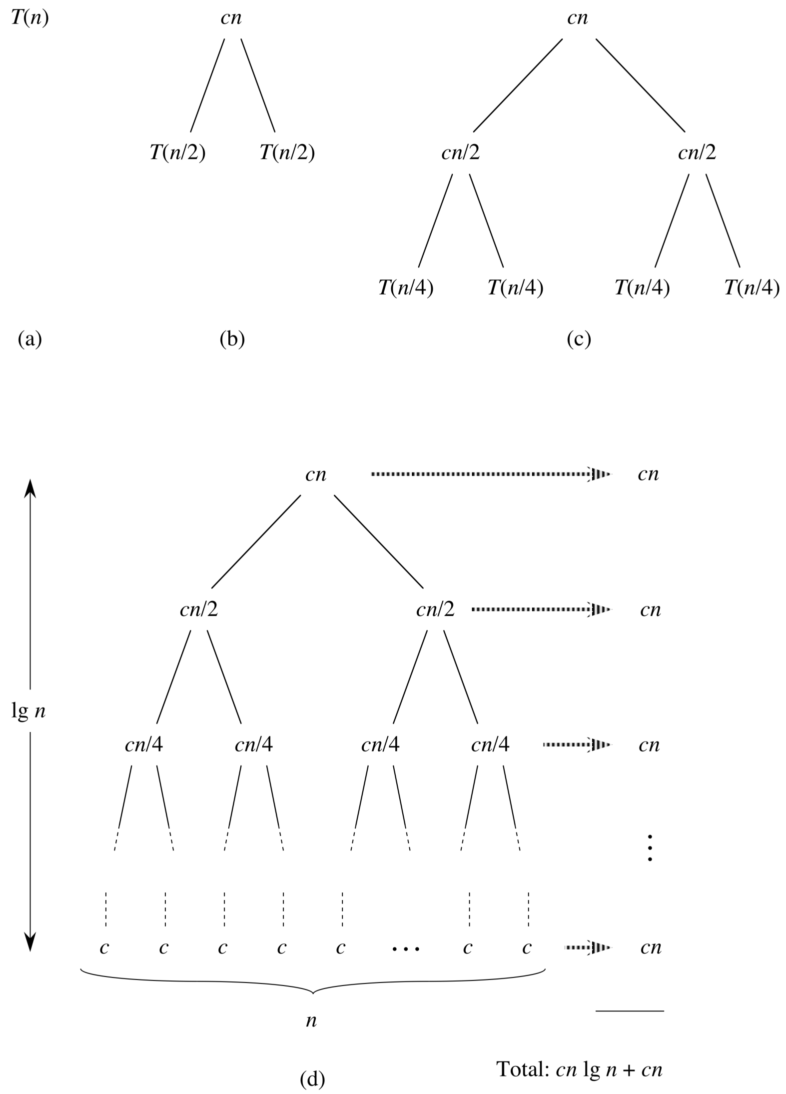
$$T(n) = \begin{cases} \Theta(1) & \text{if } n \le c, \\ aT(n/b) + D(n) + C(n) & otherwise \end{cases}$$

- For merge sort, a = 2 and b = 2.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \le 1, \\ 2T(n/2) + cn & otherwise \end{cases}$$

- Techniques for dealing with recurrences: substitution method, guess and then prove by mathematical induction (Ex 2.3-3), draw a recurrence tree and then calculate the run-time.

- Running time analysis, by drawing the recurrence tree:



  - $\lceil \log_2 n \rceil$ levels of merging in the tree
  - Each level takes linear (to n) time
  - Thus, total running time is $\Theta(n \log n)$

- Ex 2.3-5: Binary Search

- Ex 2.3-6. Insertion sort combined with binary search

- Problem 2.1: merge sort combined with insertion sort