**EECS2040 Data Structure Hw #6 (Chapter 7 Sorting, Chapter 8 Hashing)**

**due date 6/20/2021**

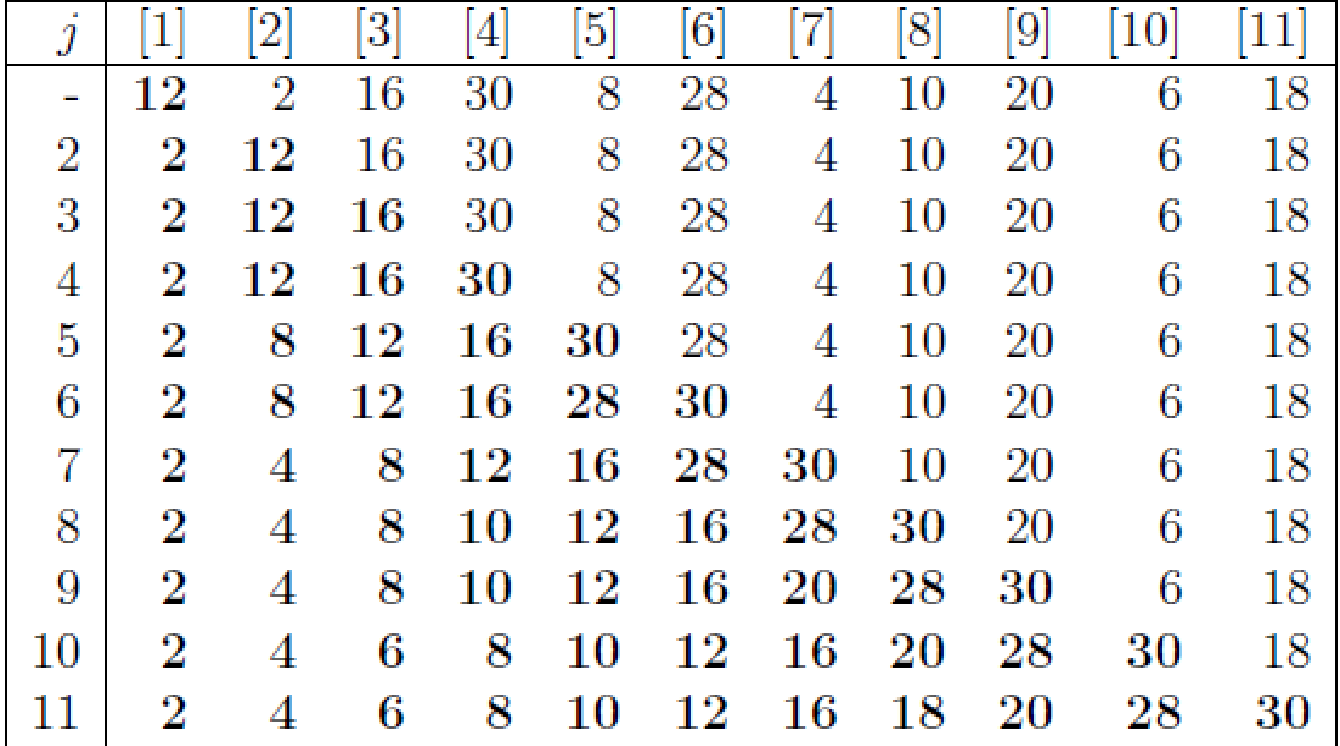
***Format***: Use a text editor to type your answers to the homework problem. You need to submit your HW in an HTML file or a DOC file named as **Hw6-SNo.doc** or **Hw6-SNo.html**, where SNo is your student number. Submit the **Hw6-SNo.doc or Hw6-SNo.html** file via eLearnl. Inside the file, you need to put the **header and your student number, name (e.g., EECS2040 Data Structure Hw #6 (Chapter 7, 8) due date 6/20/2021 by SNo, name)** first, and then the **problem** itself followed by your **answer** to that problem, one by one. The grading will be based on the correctness of your answers to the problems, and the **format**. Fail to comply with the aforementioned format (file name, header, problem, answer, problem, answer,…), will certainly degrade your score. If you have any questions, please feel free to ask me.

**Part 1**

1. (50%) The list L: (12, 2, 16, 30, 8, 28, 4, 10, 20, 6, 18) is to be sorted by various sorting algorithm.
2. Write the status of the list at the end of each iteration of the **for** loop of InsertionSort (Program 7.5). Trace the program; understand it. Put your answer in the following table. (add necessary rows for your answer)

| j | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| - | 12 | 2 | 16 | 30 | 8 | 28 | 4 | 10 | 20 | 6 | 18 |
| 2 |  |  |  |  |  |  |  |  |  |  |  |
| .. |  |  |  |  |  |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |  |  |  |  |  |

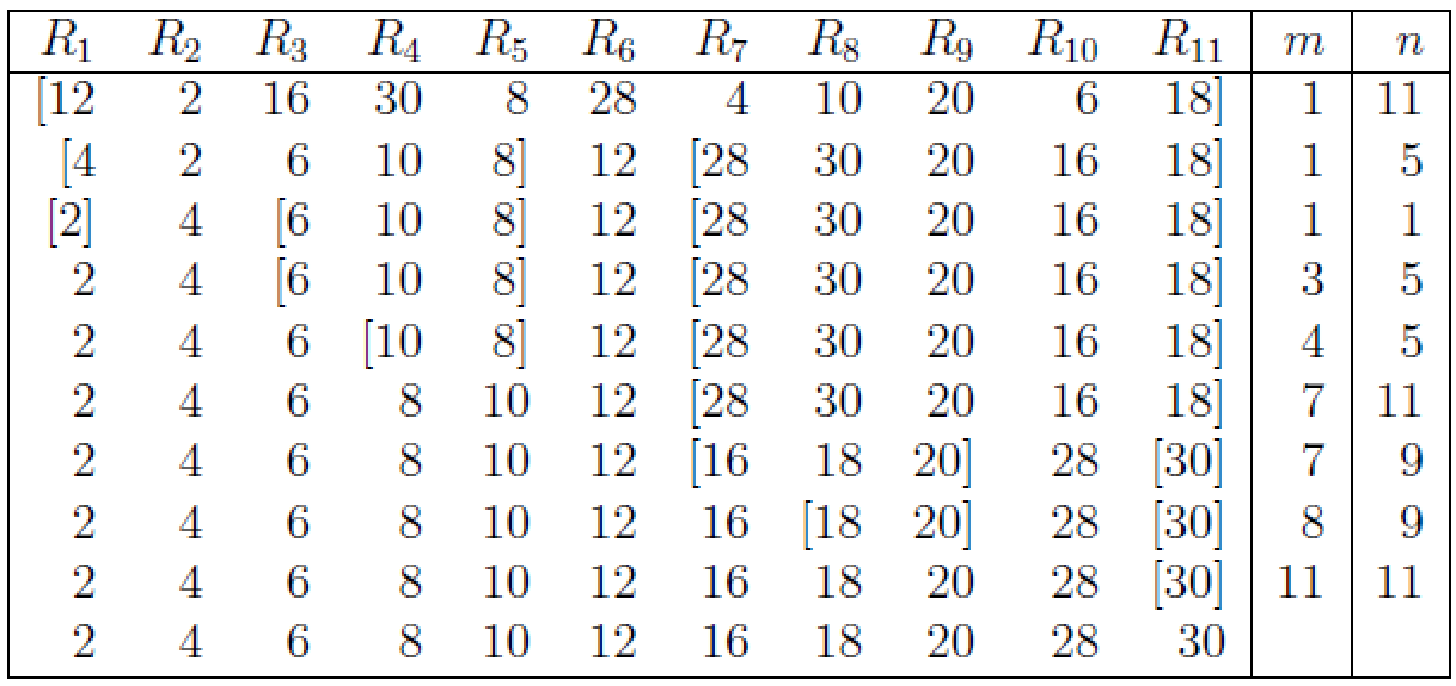
Sol:



1. Trace Program 7.6 QuickSort, use it on the list L, and draw a figure similar to Figure 7.1 Quick Sort example starting with the list L. Put your answer in the following table. (add necessary rows for your answer)

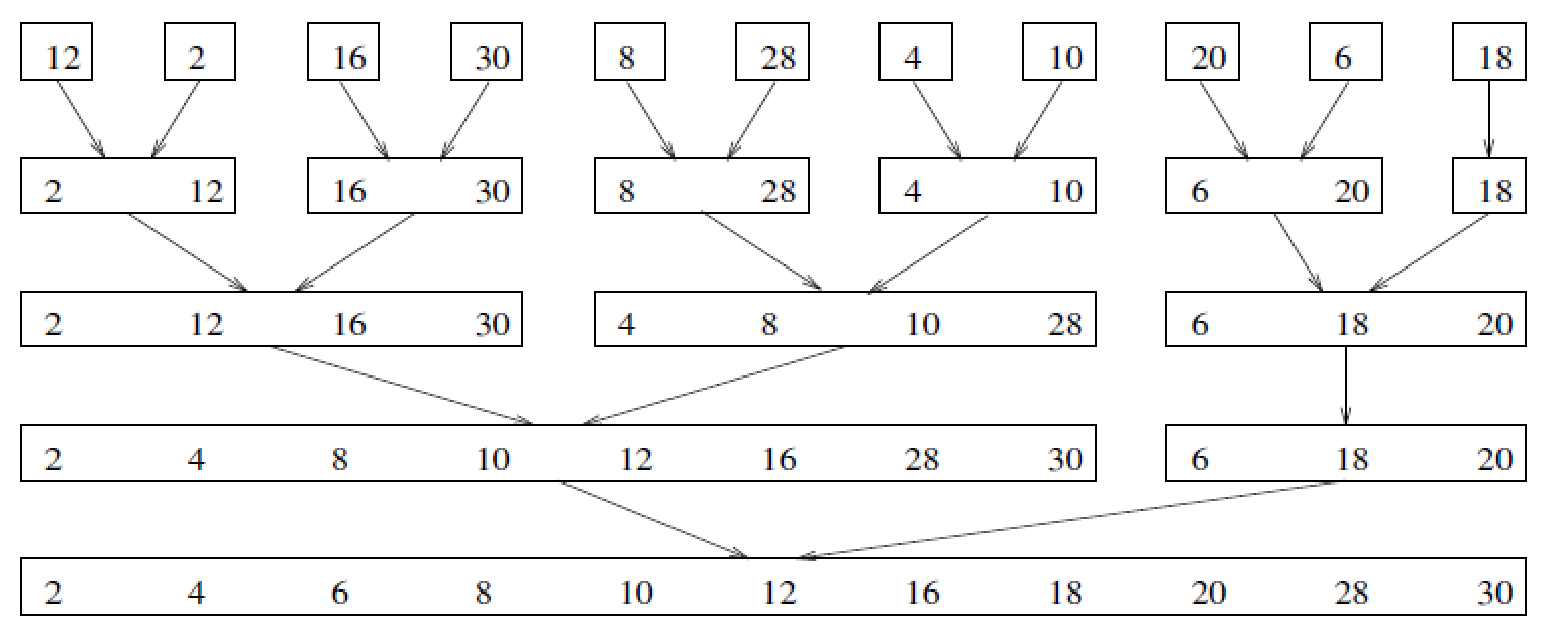
| R1 | R2 | R3 | R4 | R5 | R6 | R7 | R8 | R9 | R10 | R11 | left | right |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [12 | 2 | 16 | 30 | 8 | 28 | 4 | 10 | 20 | 6 | 18] | 1 | 11 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

Sol:

1. Write the status of the list L at the end of each phase of MergeSort (Program 7.9), i.e., draw the Merge tree (similar to Figure 7.4 in textbook) of this problem.

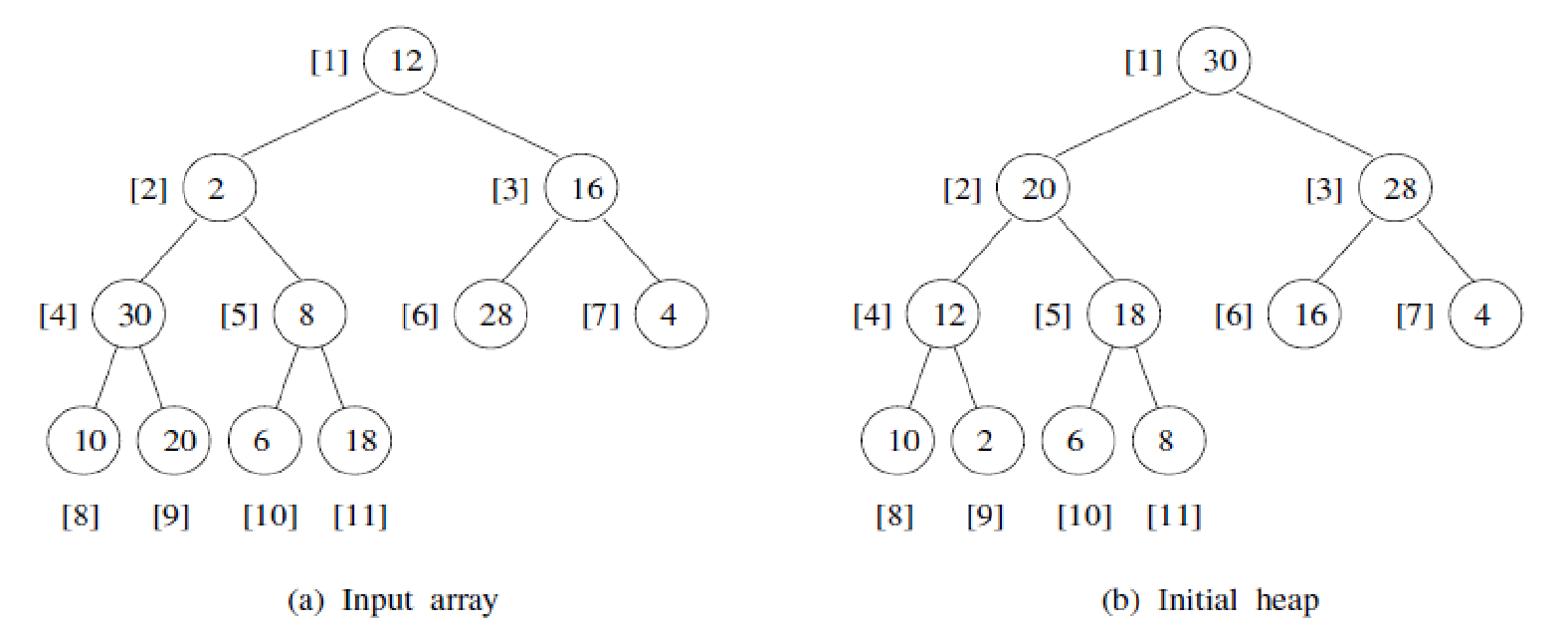
Sol:

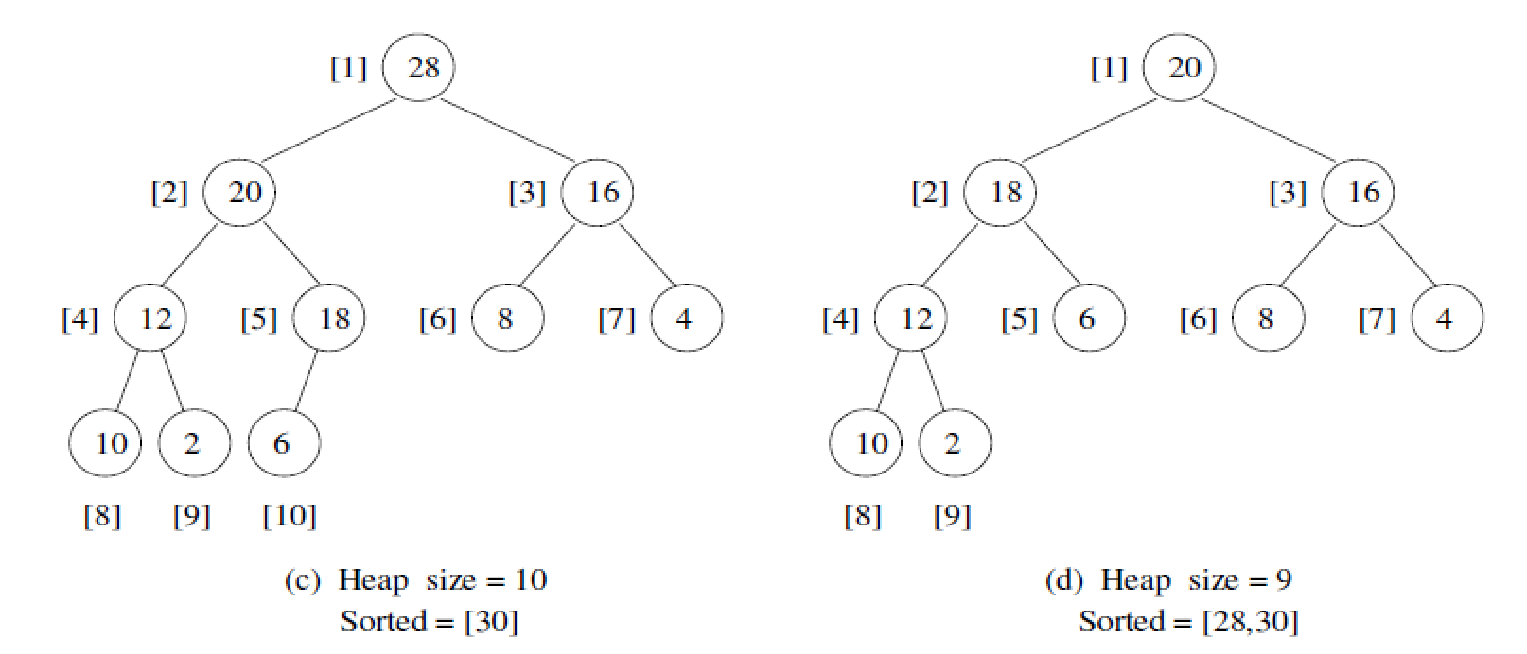


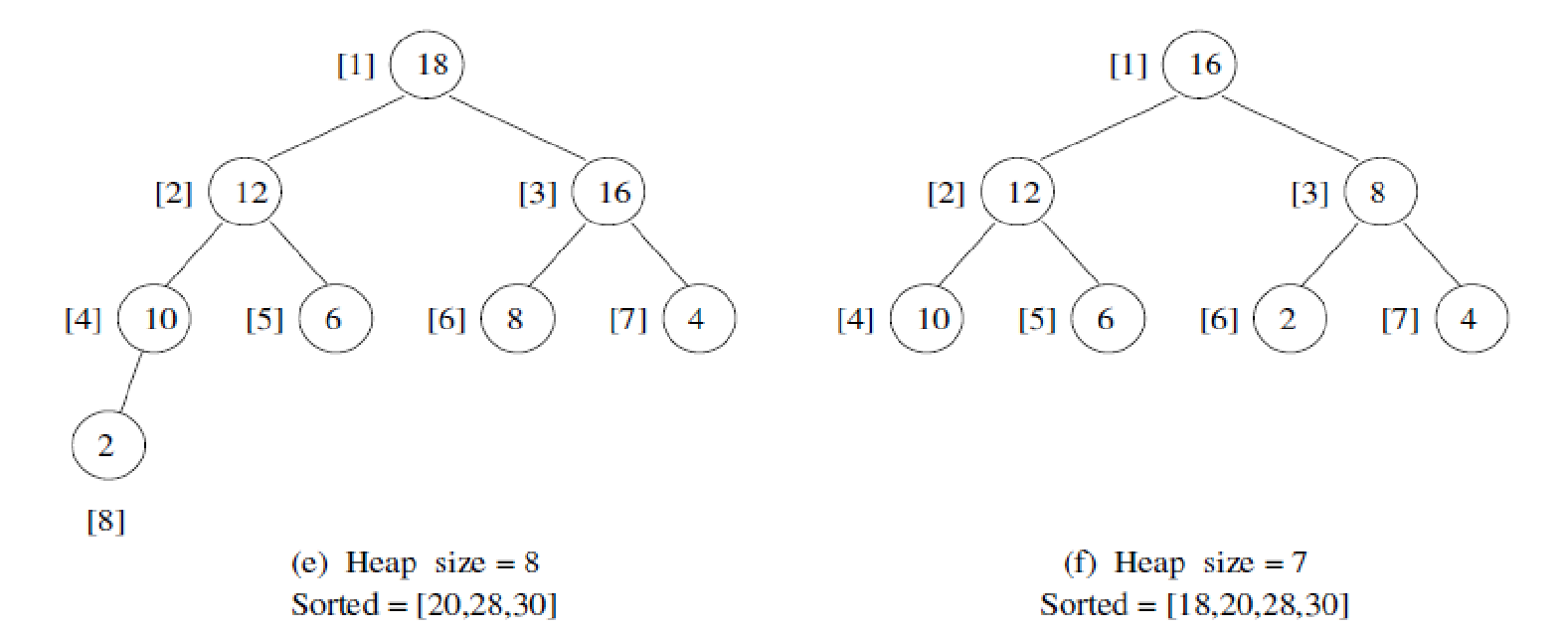


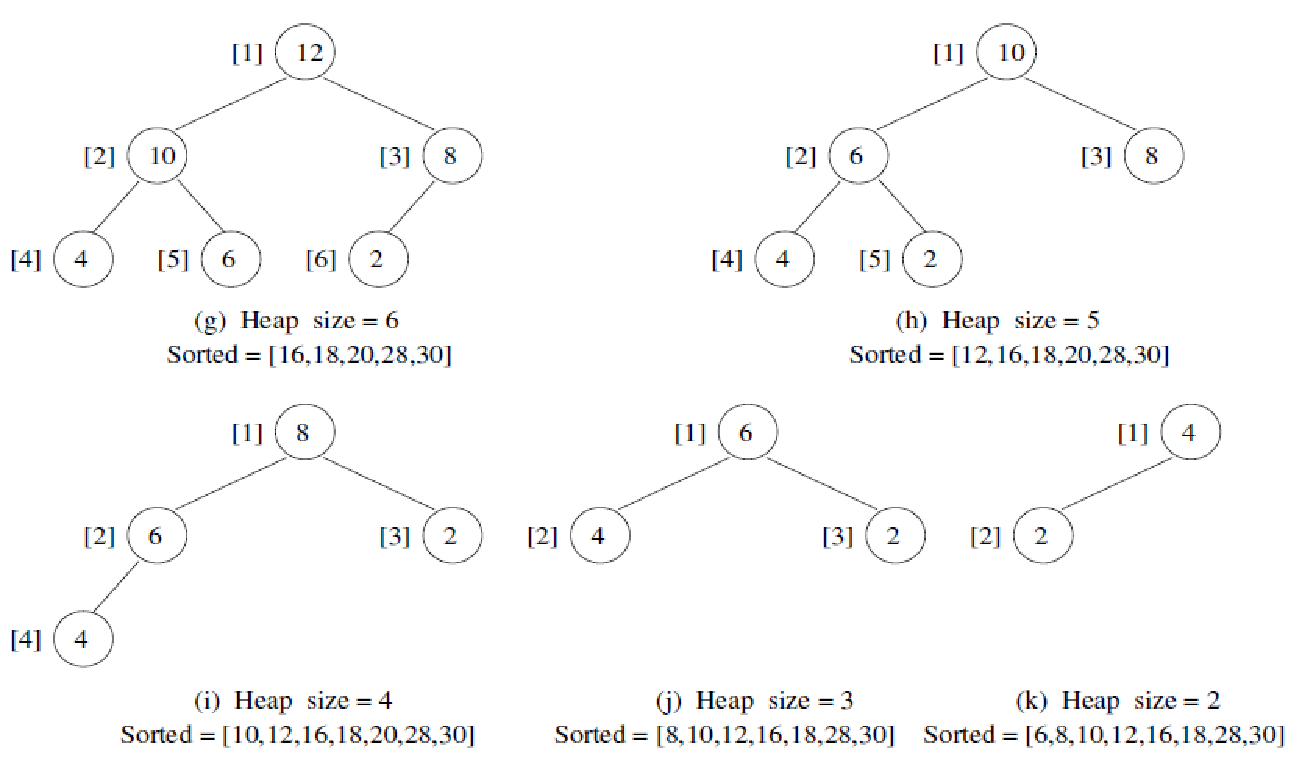
1. Write the status of the list L at the end of the first **for** loop as well as at the end of the second **for** loop of HeapSort (Program 7.14), i.e., you need to draw the following trees for: 1) input array, 2) initial heap, and 9 more trees with heap size from 10 down to 2 with corresponding sorted array as shown in Figure 7.8 in textbook.

Sol:





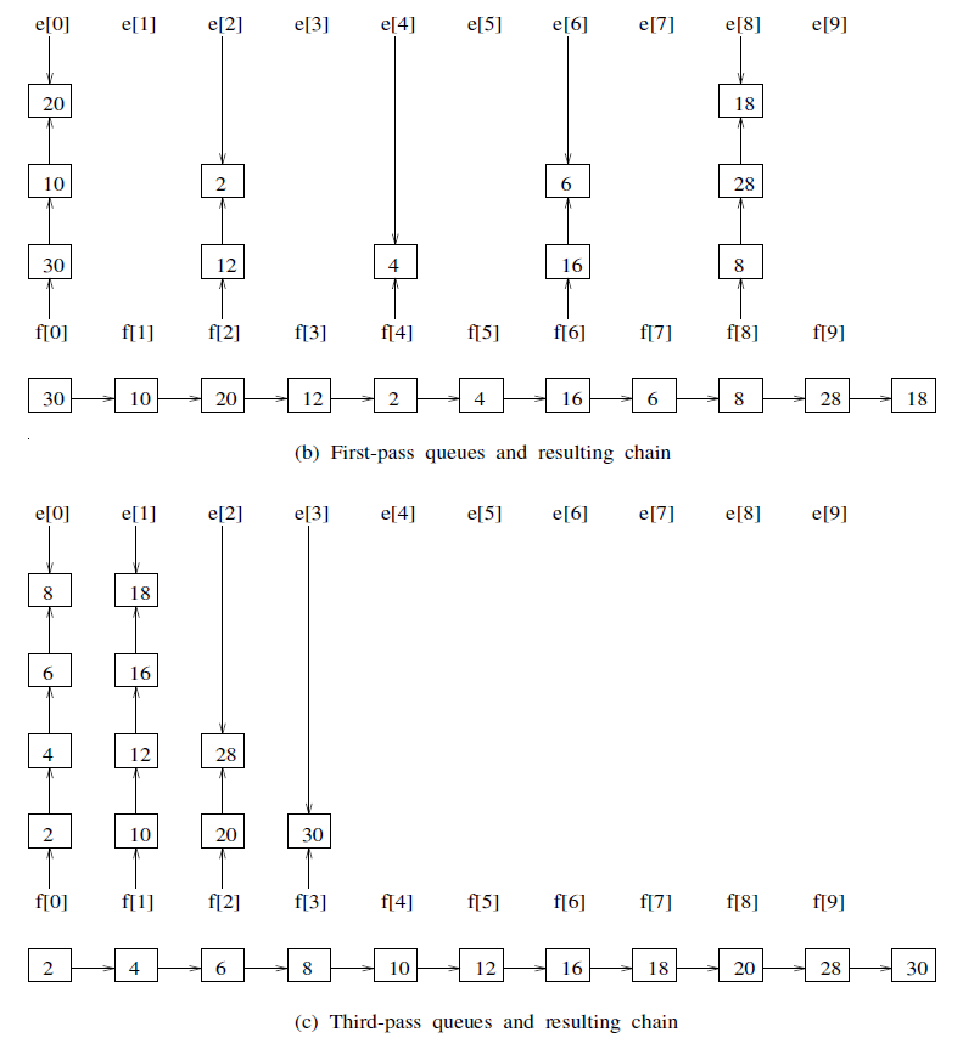




1. Write the status of the list L at the end of each pass of RadixSort (Program 7.15), using r = 10. That is fill the missing parts (the boxes with numbers between e[j] and f[j] enclosed by red dashed rectangle in the (b) and (c) part of the following figure, and the missing numbers in the resulting chain (red boxes) in (b).)



Sol:



1. (10%) QuickSort (Program 7.6) is an unstable sorting method. Give an example of an input list in which the order of records with equal keys is not preserved.

Sol:

The list (2,2,3,4), by Quicksort ,it will becomes (2,2,3,4)

therefore, the order of records with equal keys is not preserved.

1. (10%) Show that MergeSort (Program 7.9) is stable

Sol:

When we merge lists A and B, the order of records with equal keys in A is preserved in the resulting list, C, since we put a series of records with equal keys in A into C sequentially. Similarly, the order of records with equal keys in B is preserved in the resulting list, C. In addition, if the record in A has the same key as the record in B, we put the one in A into C first and put the one in B into C later. Thus, the order of records with equal keys is preserved in C after merging, so MergeSort is stable.

1. (10%) If we have n records with integer keys in the range [0,n2),then they can be sorted in O(nlogn) time using Heap Sort or Merge Sort. Radix Sort on a single key (i.e., d = 1 and r = n2) takes O(n2) time. Show how to interpret the keys as two subkeys so that Radix Sort will take only O(n) time to sort n records. (Hint: Each key, Ki, may be written as Ki = Ki1\*n + Ki2 with Ki1 and Ki2 integers in the range [0,n).)

Sol:

Let Ki1 = Ki/n and Ki2= Ki%n, Then clearly Ki = Ki1\*n + Ki2 with Ki1 and Ki2 integers in the range [0,n). Consider Ki2 as the least significant digit and Ki1 as the most significant digit. Then radix sort will sort records in the following way:

for (int i = 2; i; i--)

do a stable sort on records by Ki.

The complexity of radix sort is O(d(n+r)), where n is the number of records, r is the radix, and d is the number of passes. Here, d = 2, n = n and r = n, so the complexity is O(n).

1. (10%) Show that the hash function h(k) = k%17 does not satisfy the one-way property, weak collision resistance, or strong collision resistance.

Sol:

One-way property: given c, it’s hard to find k such that h(k) = c.

For h(k) = k%17, given c, then it is easy to know that k must be of the form c+17\*m. So h(k) does not satisfy one-way property.

Weak collision resistance: known h and M, hard to find synonym M’ such that h(M) = h(M’)

Since h(k) = k%17, and given k and h(k), It’s easy to know that k’ = k + 17\*m for any m integer, h(k’) = h(k). So h(k) does not satisfy weak collision resistance.

Strong collision resistance: hard to find a pair (x,y) such that h(x) = h(y)

Given h(k) = k%17, so it’s easy to find a pair (x,y), y = x + 17\*m for any integer m, such that h(x) = h(y). So h(k) does not satisfy strong collision resistance.

1. (10%) The probability *P*(*u*) that an arbitrary query made after *u* updates results in a filter error is given by . By differentiating *P*(*u*) with respect to *h*, show that *P*(*u*) is minimized when *h* = (log*e*2)*m*/*u*.

Sol: