CHAPTER 7

Sorting

All the programs in this file are selected from

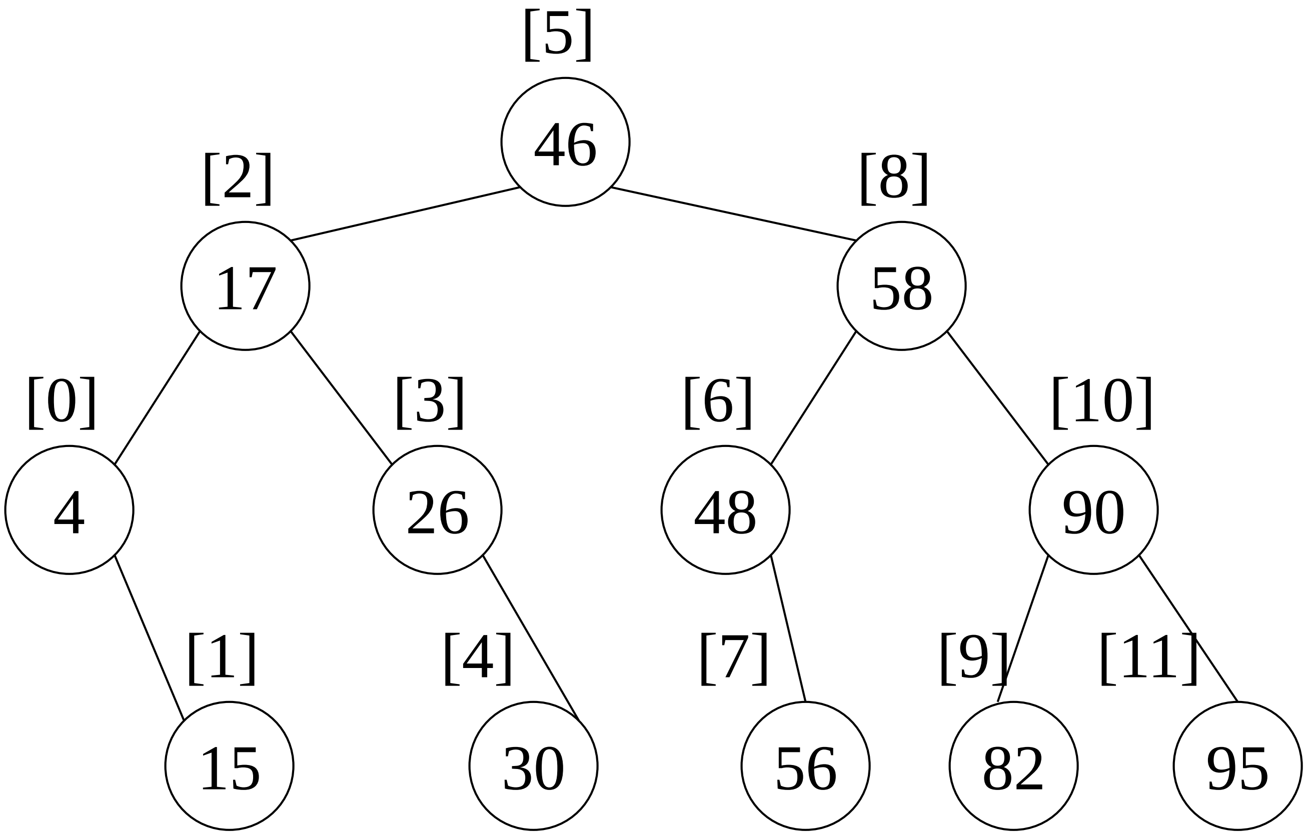
Ellis Horowitz, Sartaj Sahni, and Susan Anderson-Freed
"Fundamentals of Data Structures in C /2nd Edition",
Silicon Press, 2008.

Sequential Search

- Example

  44, 55, 12, 42, 94, 18, 06, 67
- unsuccessful search
  - n+1
- successful search

$$\sum_{i=0}^{n-1}(i+1)/n = \frac{n+1}{2}$$



4, 15, 17, 26, 30, 46, 48, 56, 58, 82, 90, 95

List Verification

- Compare lists to verify that they are identical or identify the discrepancies.

- complexity
  - random order: $O(mn)$
  - ordered list:
    $O(t_{sort}(n)+t_{sort}(m)+m+n) \Rightarrow O(\max\{n \log n, m \log m)$

Sorting Problem

- Definition
  - given $(R_0, R_1, \ldots, R_{n-1})$, where R_i = key + data
    find a permutation σ, such that $K_{\sigma(i-1)} \leq K_{\sigma(i)}$, $0<i<n-1$
- sorted
  - $K_{\sigma(i-1)} \leq K_{\sigma(i)}$, $0<i<n-1$
- stable
  - if $i < j$ and $K_i = K_j$ then R_i precedes R_j in the sorted list
- internal sort vs. external sort
- criteria
  - # of
  - # of

Sort

25	57	48	37	12	92	86	33
25							
25	57						
25	48	57					
25	37	48	57				
12	25	37	48	57			
12	25	37	48	57	92		
12	25	37	48	57	86	92	
12	25	33	37	48	57	86	92

worst case

i	0	1	2	3	4
-	5	4	3	2	1
1	4	5	3	2	1
2	3	4	5	2	1
3	2	3	4	5	1
4	1	2	3	4	5

$\boxed{O(n^2)}$

best case (record 0~3)

left out of order (LOO)

i	0	1	2	3	4
-	2	3	4	5	1
1	2	3	4	5	1
2	2	3	4	5	1
3	2	3	4	5	1
4	1	2	3	4	5

$\boxed{O(n)}$

Variation

- Binary Insertion Sort
  - sequential search --> binary search
  - reduce # of comparisons
  - # of moves unchanged
- Linked Insertion Sort
  - array --> linked list
  - sequential search(# of comparidons unchanged)
  - move --> 0

Sort

3	7	9	6	8	5
3	7	6	8	5	9
3	6	7	5	8	9
3	6	5	7	8	9
3	5	6	7	8	9
3	5	6	7	8	9
3	5	6	7	8	9

Sort

25	57	48	37	12	92	86	33
25	57	48	37	12	33	86	92
25	57	48	37	12	33	86	92
25	33	48	37	12	57	86	92
25	33	12	37	48	57	86	92
25	33	12	37	48	57	86	92
25	12	33	37	48	57	86	92
12	25	33	37	48	57	86	92

Sort

- Given $(R_0, R_1, \ldots, R_{n-1})$
  K_i: pivot key
  if K_i is placed in $S(i)$,
  then $K_j \le K_{s(i)}$ for $j < S(i)$,
  $K_j \ge K_{s(i)}$ for $j > S(i)$.
- $\underbrace{R_0, \ldots, R_{S(i)-1}}, R_{S(i)}, \underbrace{R_{S(i)+1}, \ldots, R_{S(n-1)}}$

two partitions

Example for Quick Sort

R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	left	right
[26	5	37	1	61	11	59	15	48	19]	1	10
[11	5	19	1	15]	26	59	61	48	37	1	5
[1	5]	11	19	15	26	59	61	48	37	1	2
1	5	11	[15	19]	26	59	61	48	37	4	5
1	5	11	15	19	26	[59	61	48	37]	7	10
1	5	11	15	19	26	[48	37]	59	61	7	8
1	5	11	15	19	26	37	48	59	[61]	10	10
1	5	11	15	19	26	37	48	59	61		

Quick Sort

```
void quicksort(element list[], int left,
                                int right)
{
  int pivot, i, j;
  element temp;
  if (left < right) {
    i = left;    j = right+1;
    pivot = list[left].key;
    do {
      do i++; while (list[i].key < pivot);
      do j--; while (list[j].key > pivot);
      if (i < j) SWAP(list[i], list[j], temp);
    } while (i < j);
    SWAP(list[left], list[j], temp);
    quicksort(list, left, j-1);
    quicksort(list, j+1, right);
  }
}
```

Time and Space for Quick Sort

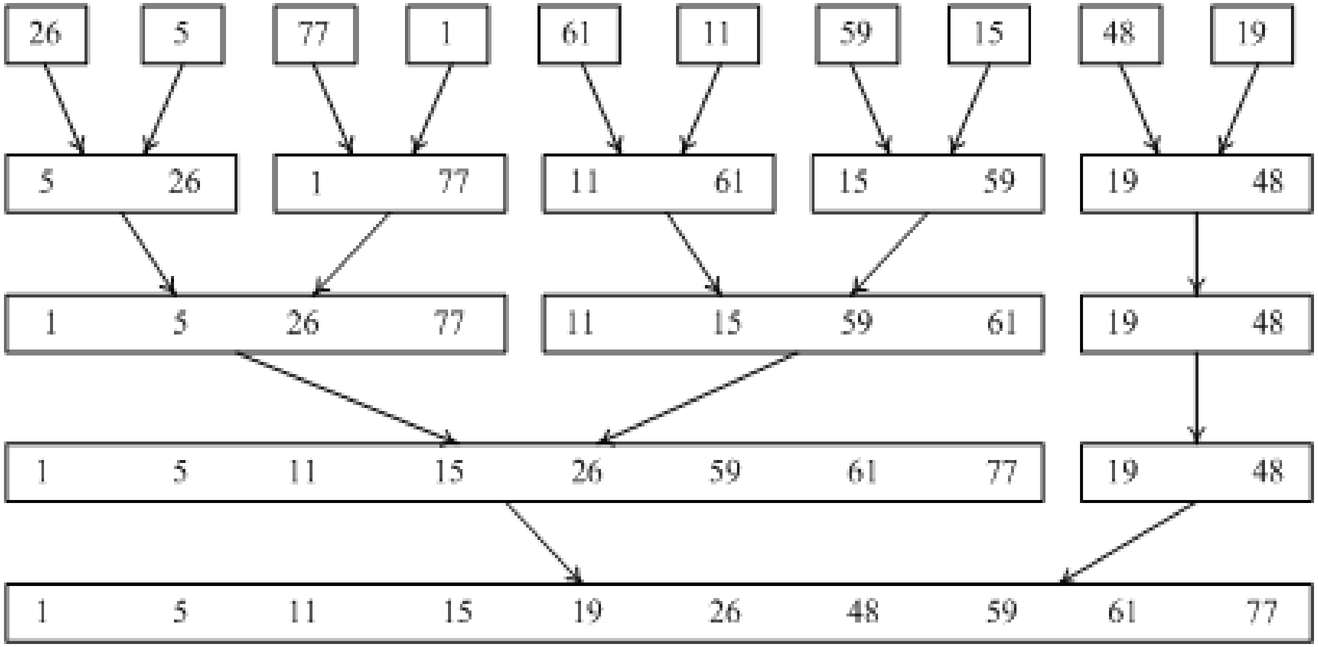
- Space complexity:
  - Average case and best case: $O(\log n)$
  - Worst case: $O(n)$
- Time complexity:
  - Average case and best case: $O(n \log n)$
  - Worst case: $O(n^2)$

Sort

- Given two sorted lists
  (list[i], …, list[m]), (list[m+1], …, list[n])
  generate a single sorted list
  (sorted[i], …, sorted[n])

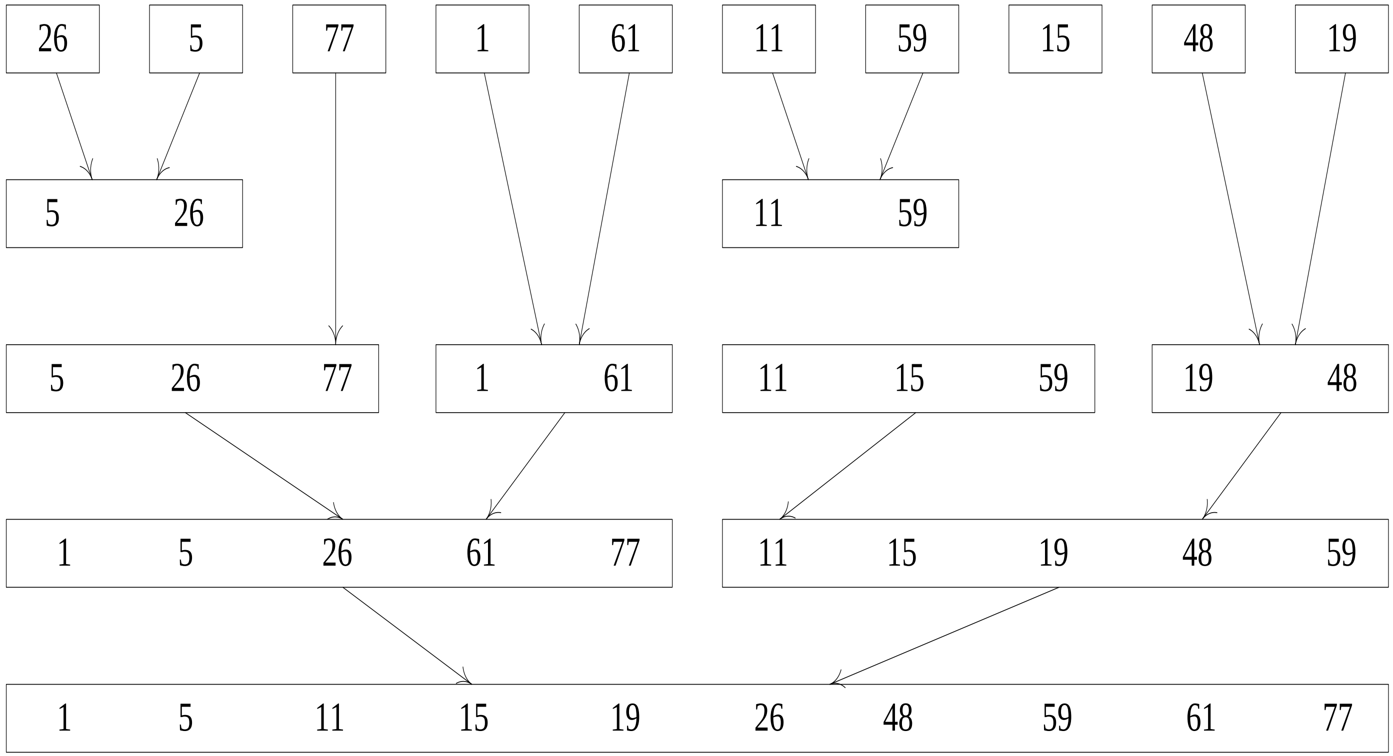
Merge Sort

Sort 26, 5, 77, 1, 61, 11, 59, 15, 48, 19



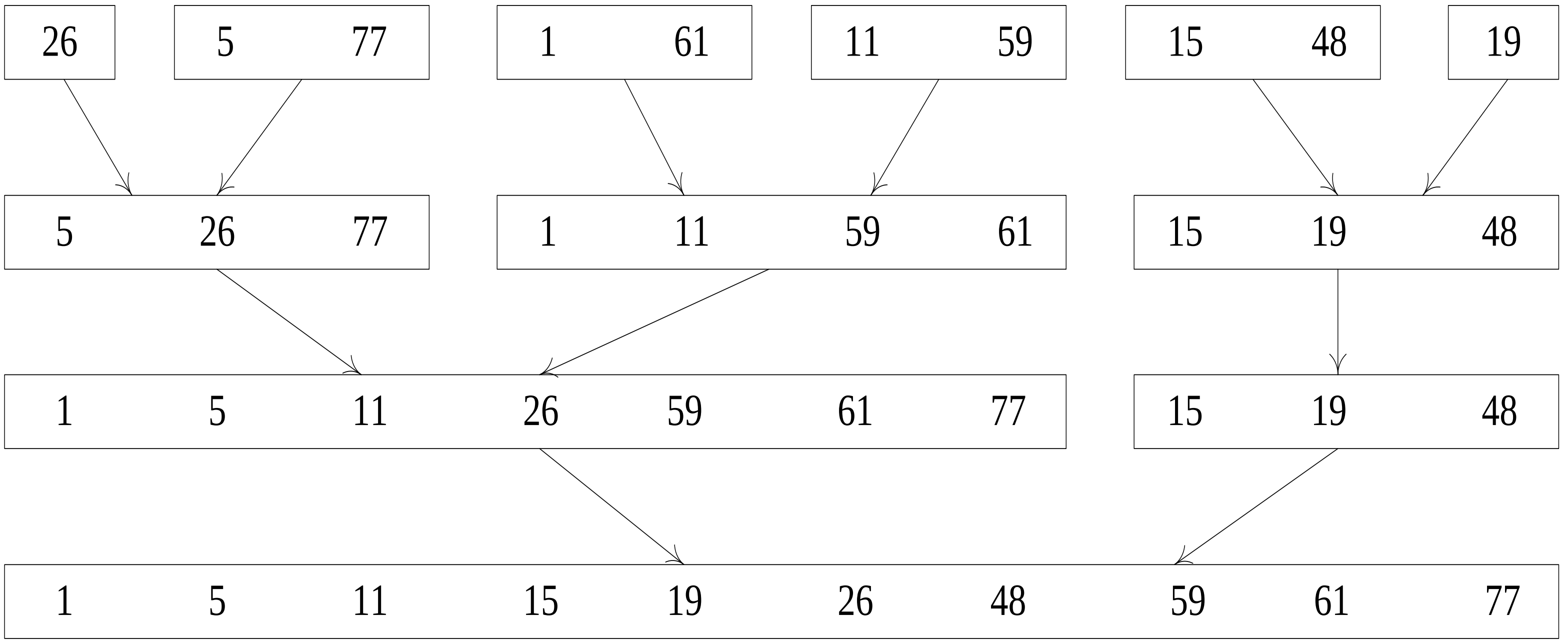
$O(n\log_2 n)$: $\lceil \log_2 n \rceil$ passes, $O(n)$ for each pass

Merge Sort



26 5 77 1 61 11 59 15 48 19
5 26
11 59
5 26 77
1 61
11 15 59
19 48
1 5 26 61 77
11 15 19 48 59
1 5 11 15 19 26 48 59 61 77

Merge Sort

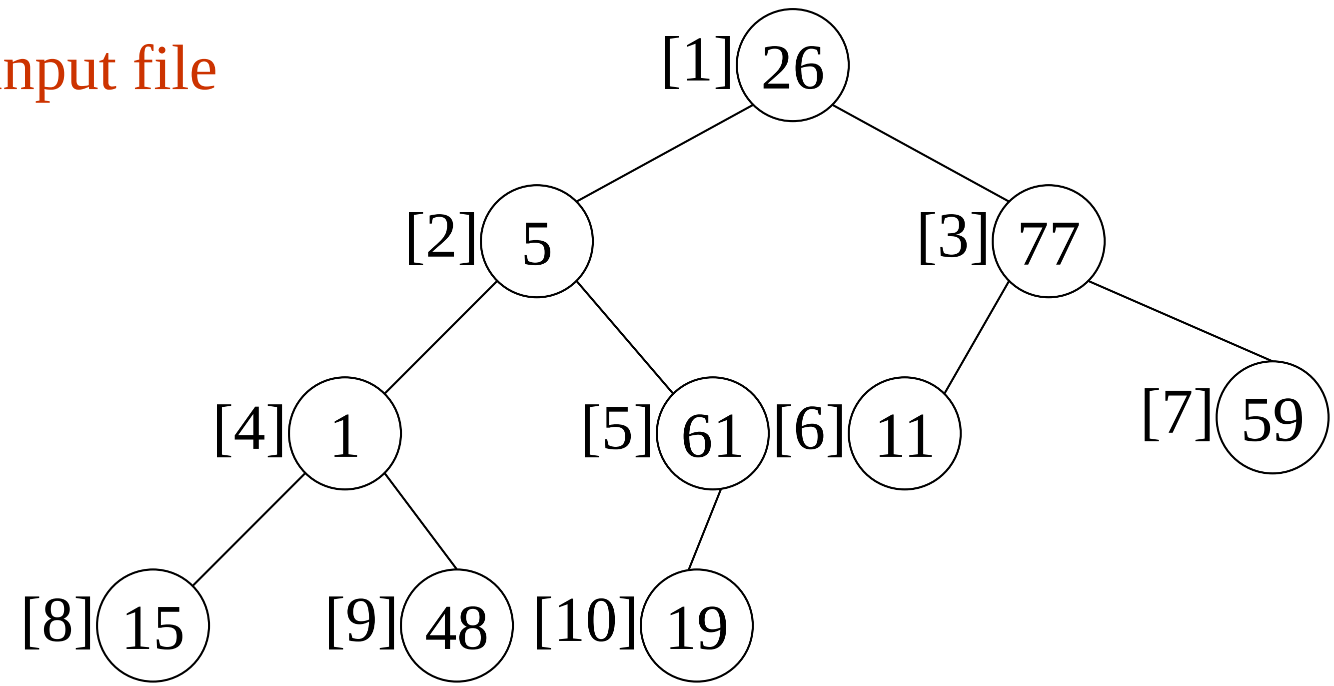


26
5 77
1 61
11 59
15 48
19
5 26 77
1 11 59 61
15 19 48
1 5 11 26 59 61 77
15 19 48
1 5 11 15 19 26 48 59 61 77

Sort

1	2	3	4	5	6	7	8	9	10
26	5	77	1	61	11	59	15	48	19

input file



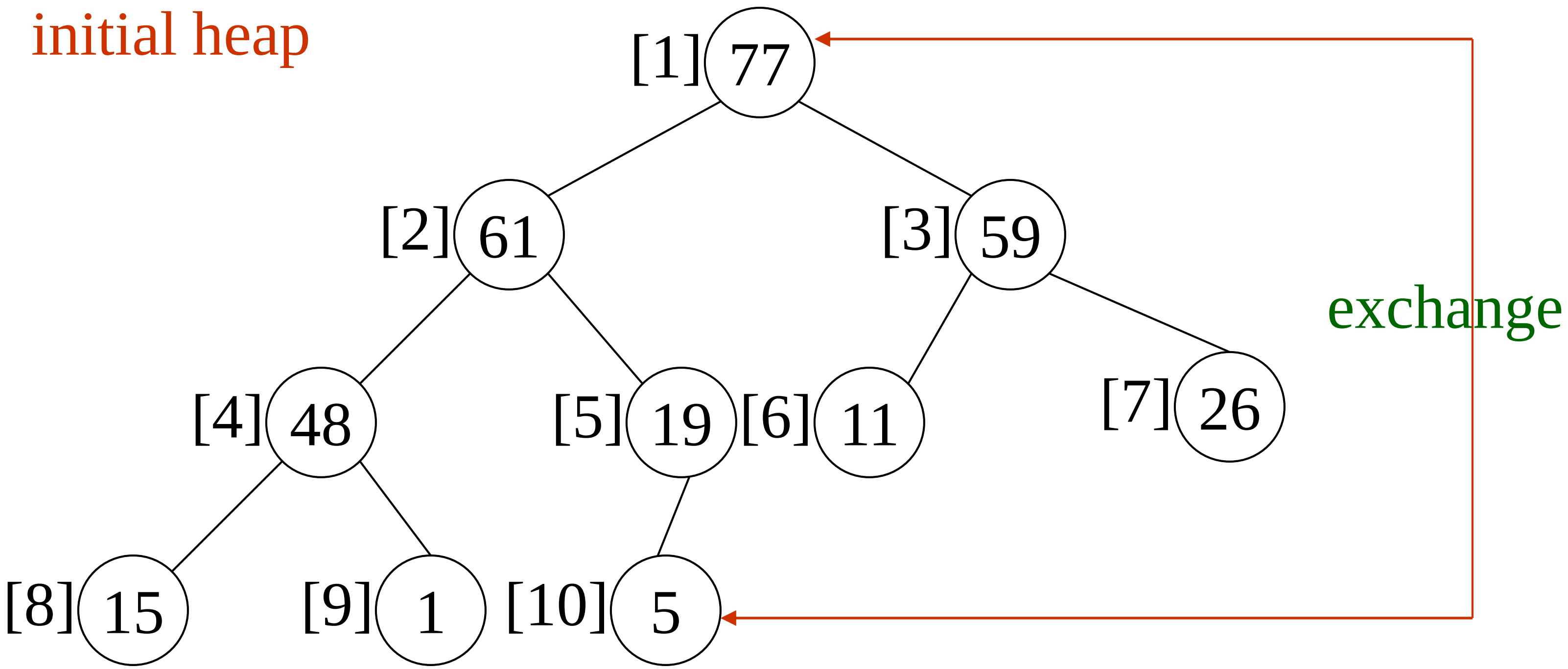


Figure 7.7: Array interpreted as a binary tree (p.354)

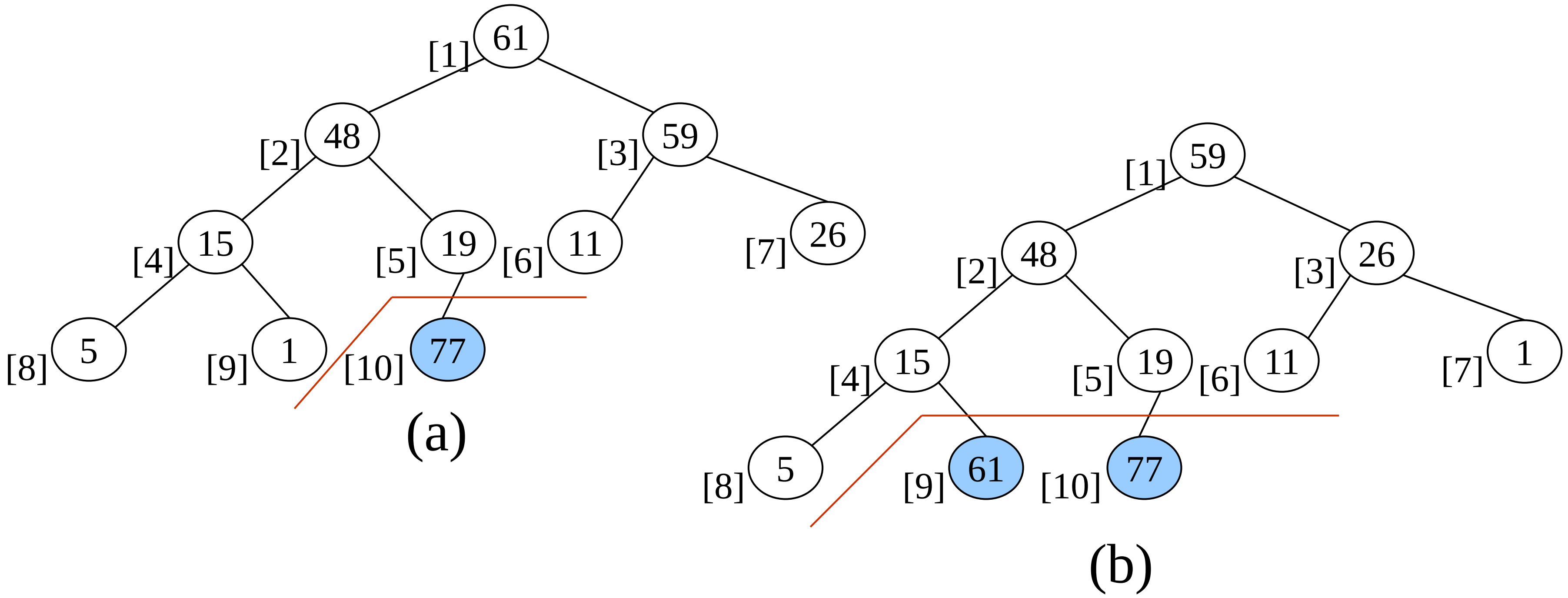
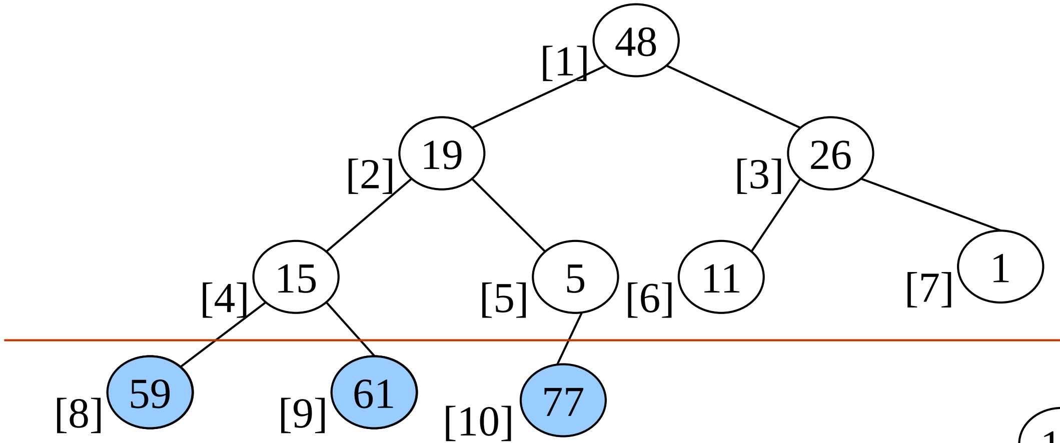
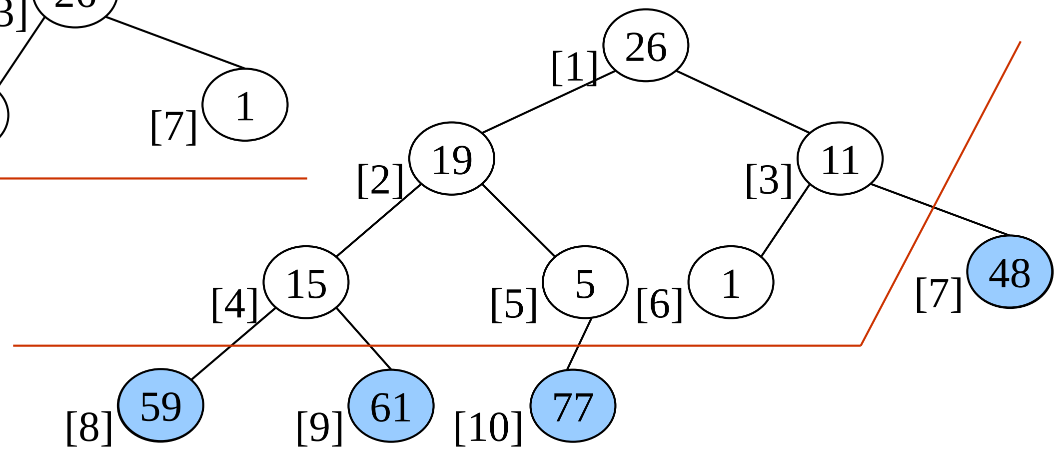


Figure 7.8: Heap sort example(p.355)



(c)



$O(n\log_2 n)$

(d)

Figure 7.8: Heap sort example (continued) (p.355)

Sort

Sort by keys

K^0, K^1, …, K^{r-1}

Most significant key — Least significant key

$R_0, R_1, \ldots, R_{n-1}$ are said to be sorted w.r.t. $K_0, K_1, \ldots, K_{r-1}$ iff

$$(k_i^0, k_i^1, \ldots, k_i^{r-1}) <= (k_{i+1}^0, k_{i+1}^1, \ldots, k_{i+1}^{r-1}) \quad 0 \le i < n-1$$

Most significant digit first: sort on K^0, then K^1, ...

Least significant digit first: sort on K^{r-1}, then K^{r-2}, ...

Radix Sort

$0 \leq K \leq 999$

$(K^0, \quad K^1, \quad K^2)$

MSD LSD

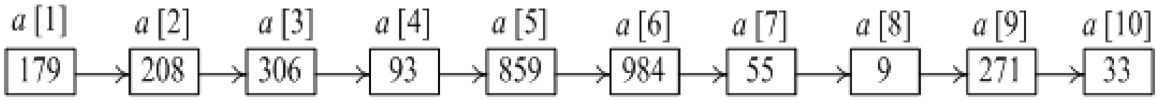
0-9 0-9 0-9

radix 10 sort

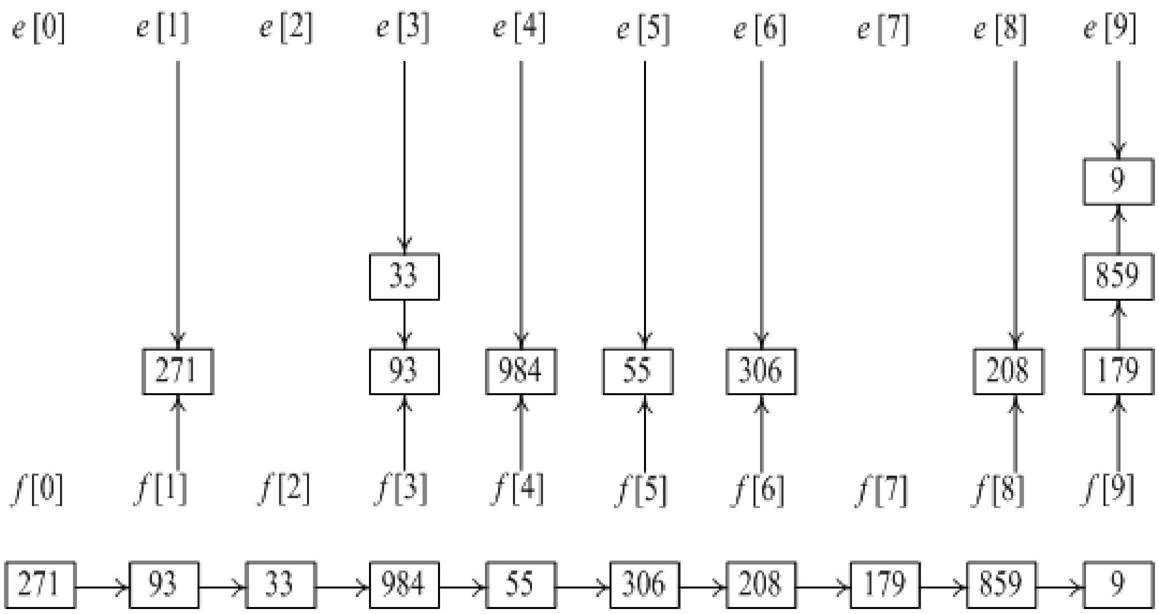
radix 2 sort

Example for Radix Sort

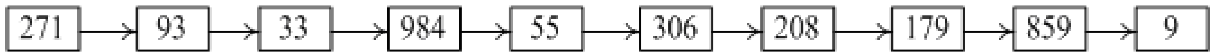
d (digit) = 3, r (radix) = 10 ascending order



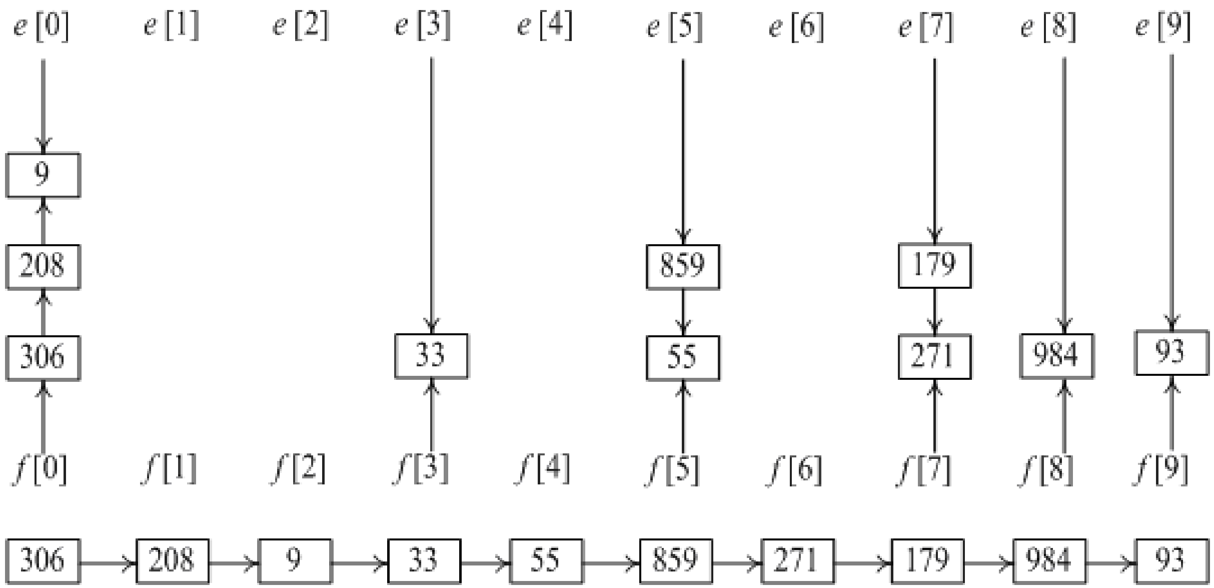
(a) Initial input



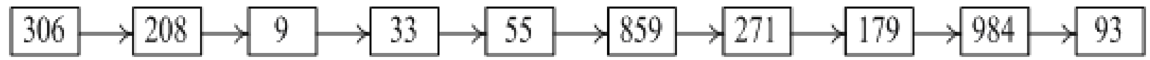
(b) First-pass queues and resulting chain



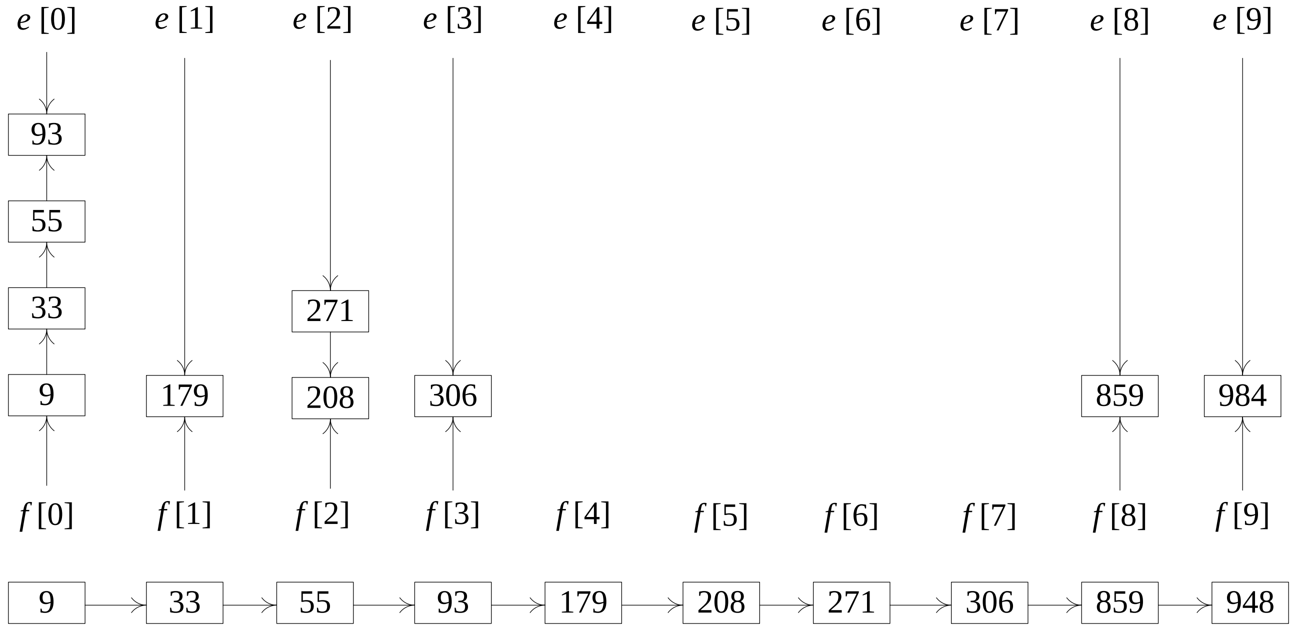
(b) First-pass queues and resulting chain



(c) Second-pass queues and resulting chain



(c) Second-pass queues and resulting chain

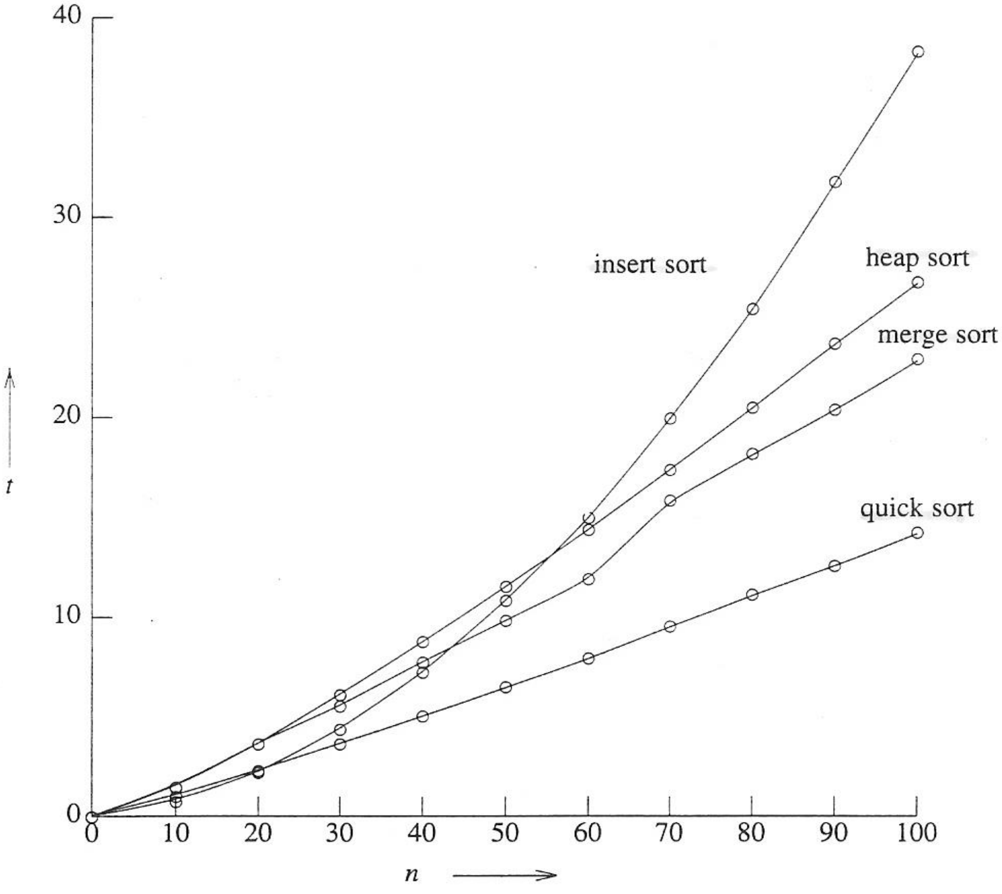


(d) Third-pass queues and resulting chain

Complexity of Sort

	stability	space	time		
			best	average	worst
Bubble Sort	stable	little	O(n)	$O(n^2)$	$O(n^2)$
Insertion Sort	stable	little	O(n)	$O(n^2)$	$O(n^2)$
Selection Sort	unstable	little	O(n)	$O(n^2)$	$O(n^2)$
Quick Sort	unstable	O(logn)			
Merge Sort	stable	O(n)	O(nlogn)	O(nlogn)	O(nlogn)
Heap Sort	unstable	little	O(nlogn)	O(nlogn)	O(nlogn)
Radix Sort	stable	O(n*p)			

Comparison(1/2)



n	***Insert***	***Heap***	***Merge***	***Quick***
0	0.000	0.000	0.000	0.000
50	0.004	0.009	0.008	0.006
100	0.011	0.019	0.017	0.013
200	0.033	0.042	0.037	0.029
300	0.067	0.066	0.059	0.045
400	0.117	0.090	0.079	0.061
500	0.179	0.116	0.100	0.079
1000	0.662	0.245	0.213	0.169
2000	2.439	0.519	0.459	0.358
3000	5.390	0.809	0.721	0.560
4000	9.530	1.105	0.972	0.761
5000	15.935	1.410	1.271	0.970

Comparison(2/2)

- $n < 20$:
- $20 \leq n < 45$:
- $n \geq 45$:
- hybrid method:

External Sorting

- Very large files (overheads in disk access)
  - seek time
  - latency time
  - transmission time
- merge sort
  - phase 1
    Segment the input file & sort the segments (runs)
  - phase 2
    Merge the runs

File: 4500 records, A1, …, A4500

internal memory: 750 records (3 blocks)

block length: 250 records

input disk vs. scratch pad (disk)

(1) sort three blocks at a time and write them out onto scratch pad

(2) three blocks: two input buffers & one output buffer