Quick sorting

Yulia Burkatovskaya

Outline

- Data structures
- Insertion sorting (Shell algorithm)
- Heapsort
- Quick sorting (Hoare algorithm)
- Merge sorting
- Stability

Data structures

- An **array** can be defined as an **ordered** collection of **items** indexed by **contiguous integers**.
- Random access to elements.
- A[i]:
  - A – array
  - i – index
  - A[i] - element

0	1	2	3	4	5	6	7	8	9	10	11
12	45	27	14	78	67	84	91	17	32	83	53

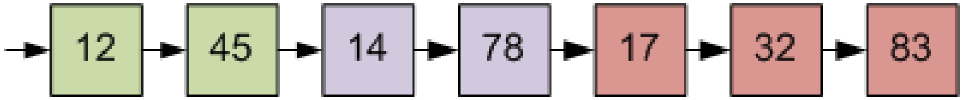
Data structures

- An **array** can be defined as an **ordered** collection of **items** indexed by **contiguous integers**.
- Random access to elements (O(1)).
- A[i]:
  - A – array
  - i – index
  - A[i] - element

0	1	2	3	4	5	6	7	8	9	10	11
12	45	27	14	78	67	84	91	17	32	83	53

Data structures

- **Linked list** is a linear collection of data elements, whose order is not given by their physical placement in memory. Instead, each element **points** to the next.
- **Sequential** access (O(n)).



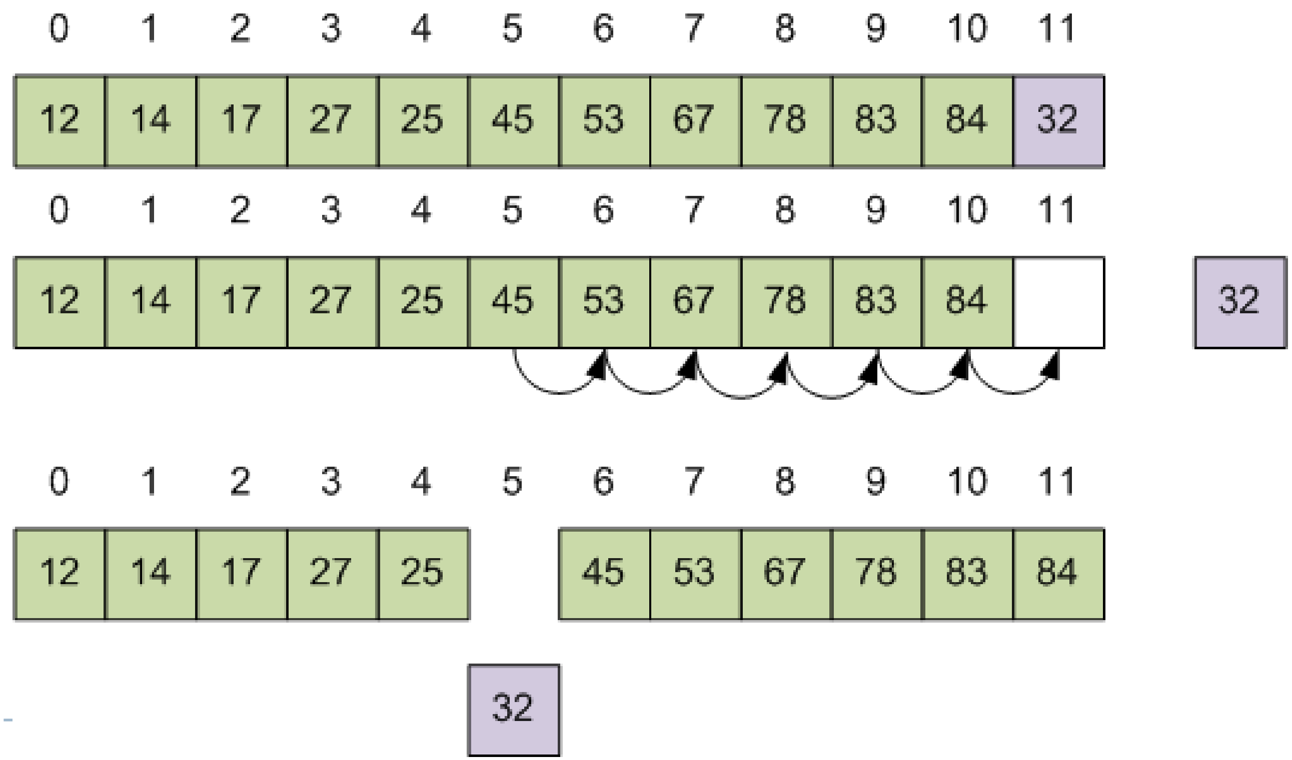
Data structures

- A **data file** is a computer file which stores data. **File** is a linear collection of data elements. It is placed in the file storage.
- **Sequential** access (O(n)).

12	45	27	14	78	67	84	91	17	32	83	53

Insertion sort

- **Idea**. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.



Insertion sort

- **Pseudocode**
- `i ← 1`
- `while i < length(A)`
  - `x ← A[i]`
  - `j ← i - 1`
  - `while j >= 0 and A[j] > x`
    - `A[j+1] ← A[j]`
    - `j ← j - 1`
  - `end while`
- `A[j+1] ← x`
- `i ← i + 1`
- `end while`

Insertion sort

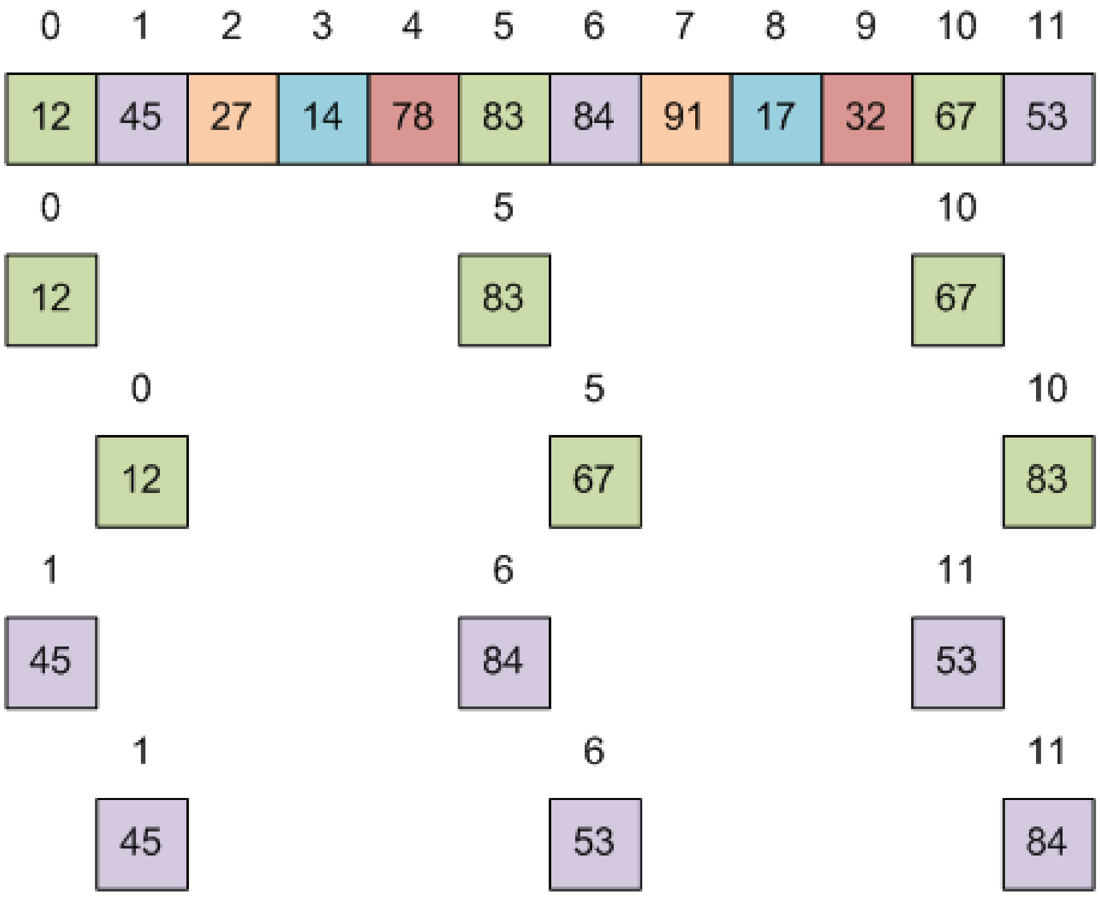
- **Computational complexity**:
- **Worst case**: N^2 (array is in the reverse order, (i-1) shifts for A[i]).
- **Best case**: N (Array is ordered).

- **Data structures:** array, doubly linked list

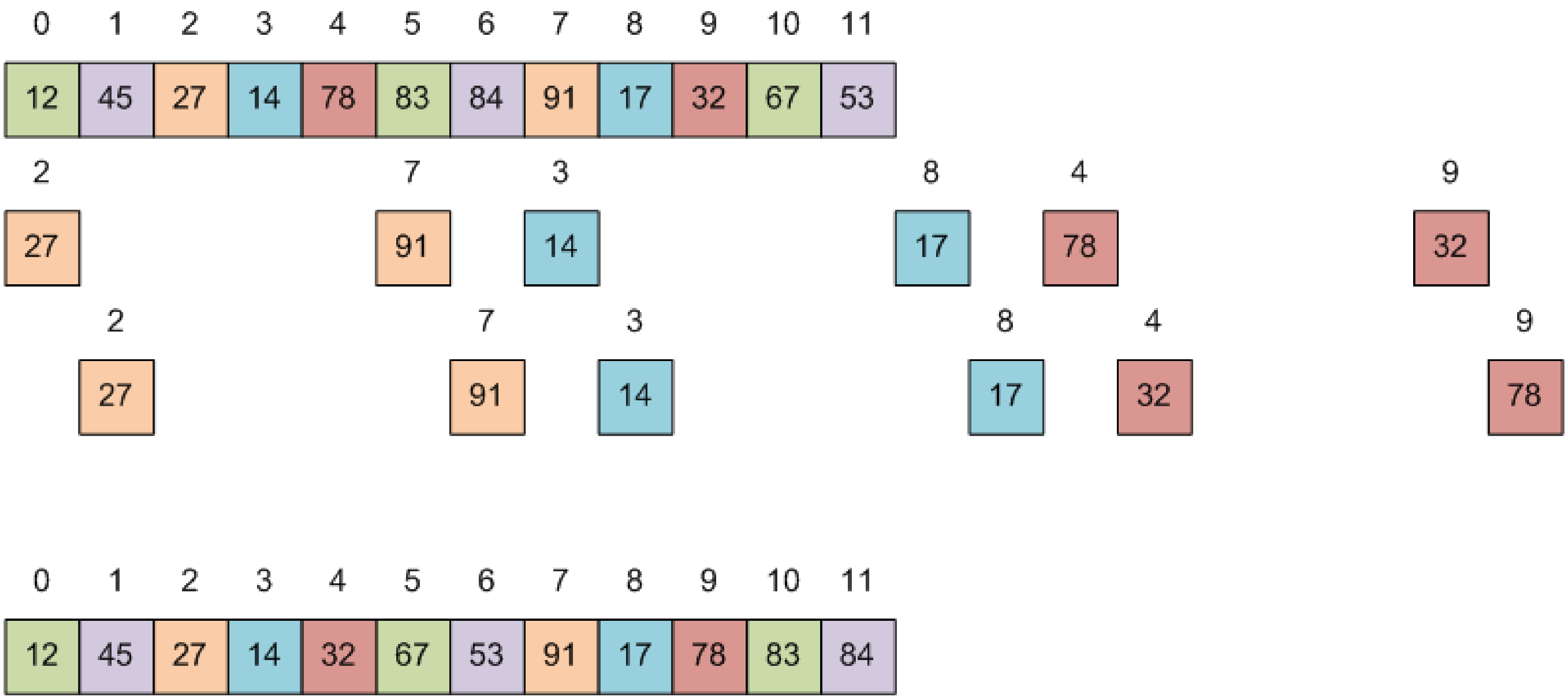
Shell algorithm

- *Shell, D. L. (1959). "A High-Speed Sorting Procedure". Communications of the ACM.* ***2*** *(7): 30–32.*
- **Idea**:
- Divide an array into h subarrays, with the gap h.
- Sort every subarray using insertion.
- Decrease h.
- Repeat until h=1.

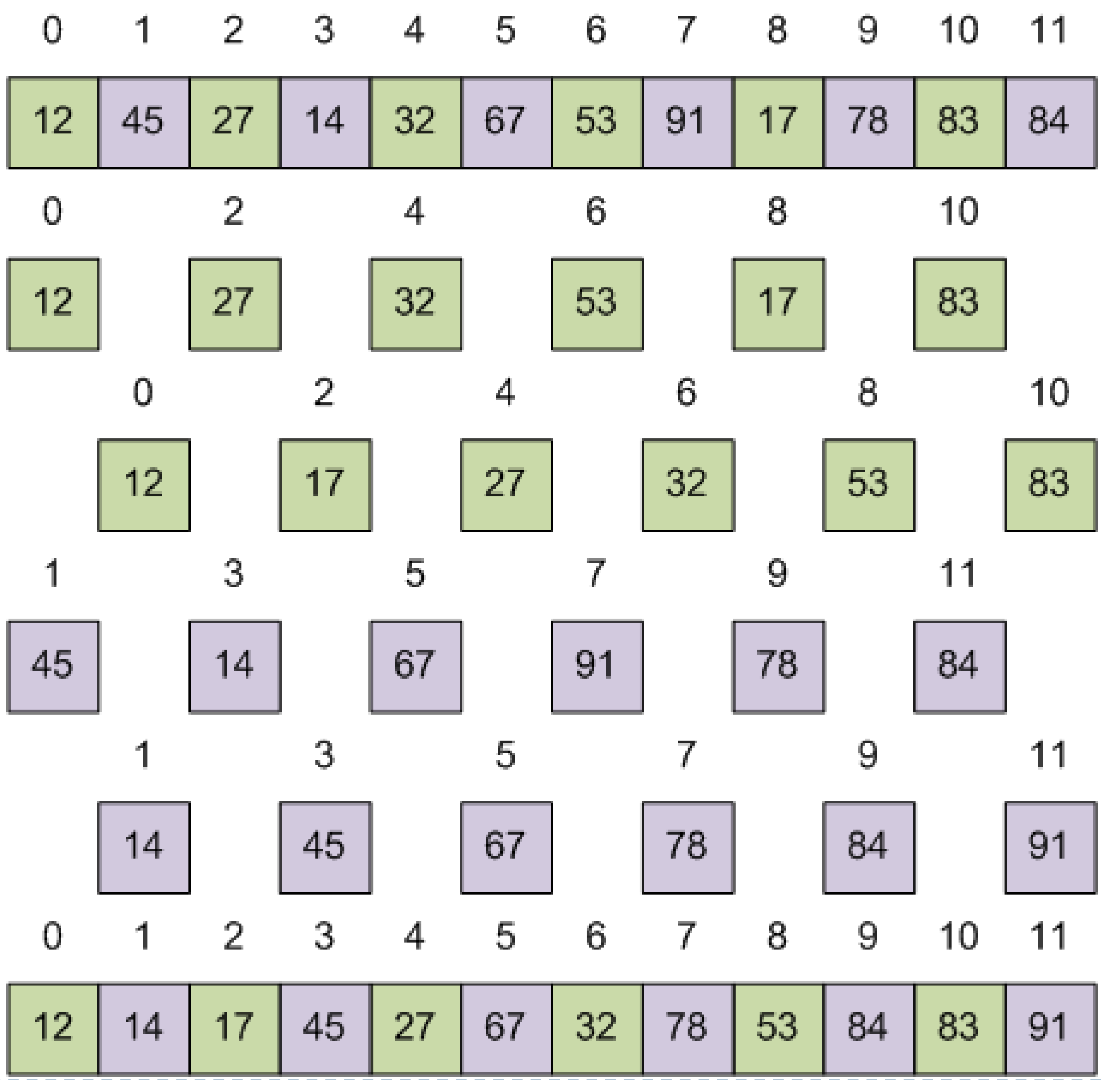
Shell algorithm



Shell algorithm



Shell algorithm



Shell algorithm

- First (for big gaps) – arrays are short
- Then (for small gaps) – small number of shifts

0	1	2	3	4	5	6	7	8	9	10	11
12	14	17	45	27	67	32	78	53	84	83	91

0	1	2	3	4	5	6	7	8	9	10	11
12	14	17	27	32	45	53	67	78	83	84	91

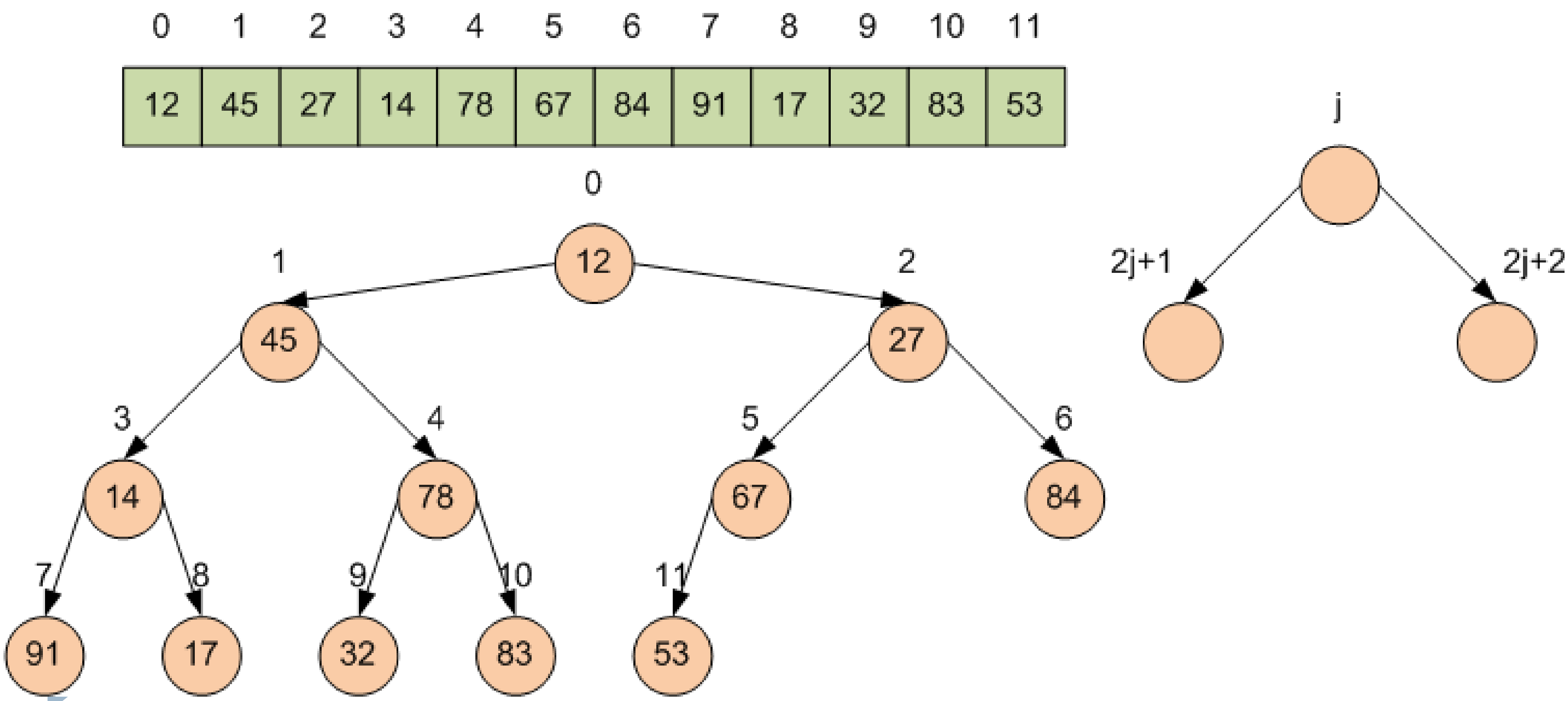
Shell algorithm

- **Gap sequence**
- It's a kind of magic (Donald Knuth).

General term	Gaps	Worst-case complexity	Authors

Heapsort

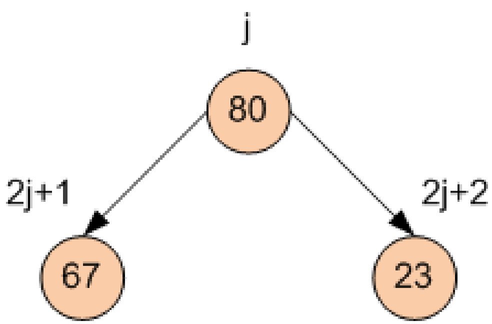
- The **(binary) heap data structure** is an array object that we can view as a nearly complete binary tree (the height is *N*ln*N*).



Heapsort

- **Heap property**

$$a\ \ i\ \geq \max\ \ a\ \ 2i+1\ ,a\ \ 2i+2$$



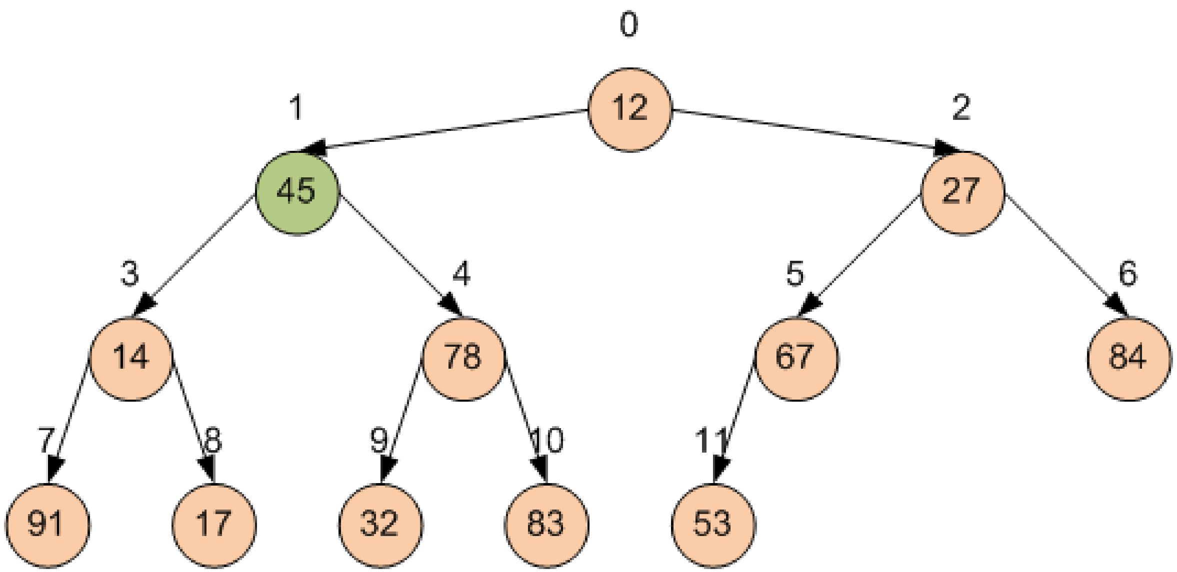
$a_j \geq a_{2j+1}$

$a_j \geq a_{2j+2}$

- *Williams, J. W. J. (1964), "Algorithm 232 - Heapsort", Communications of the ACM,* ***7*** *(6): 347–348,*

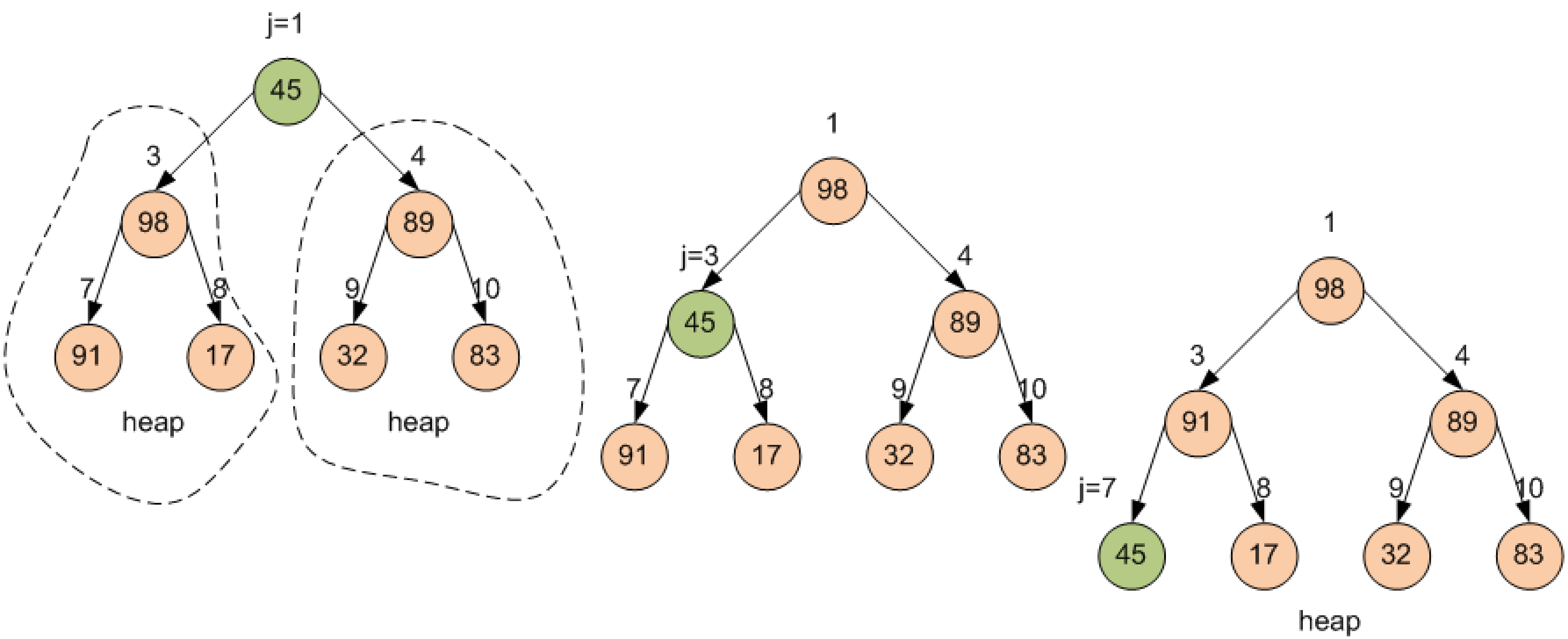
Heapsort

- **HEAPIFY** (maintaining heap property)



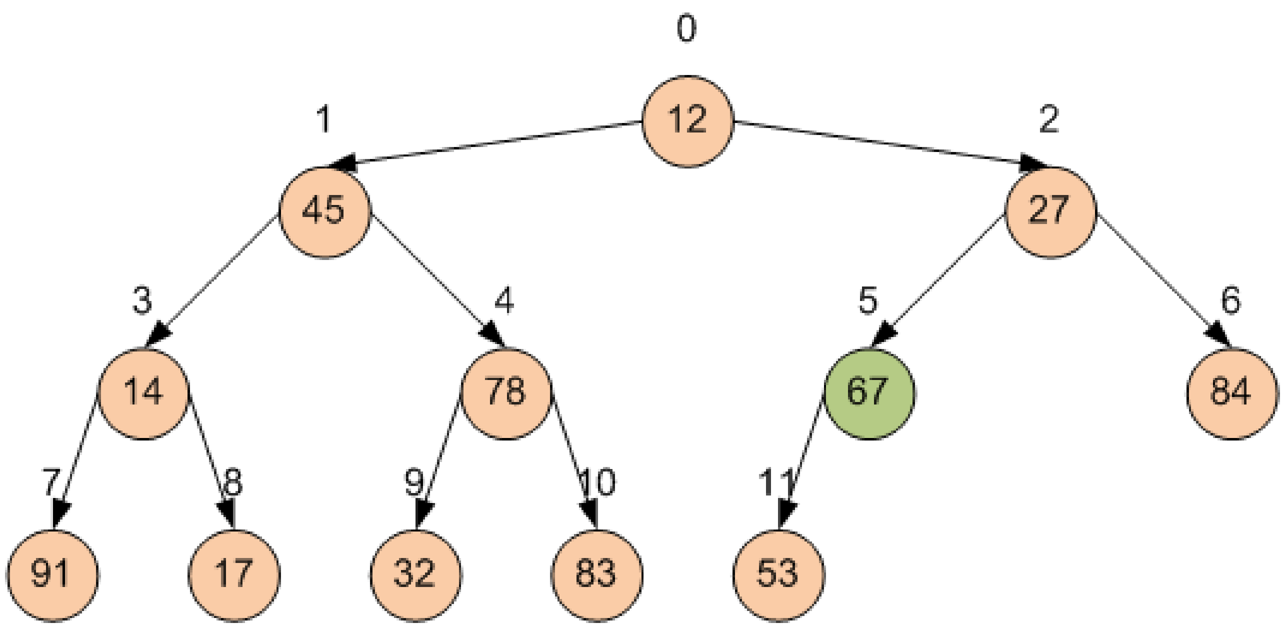
Heapsort

- **HEAPIFY** (maintaining heap property)
- Complexity O(ln N)

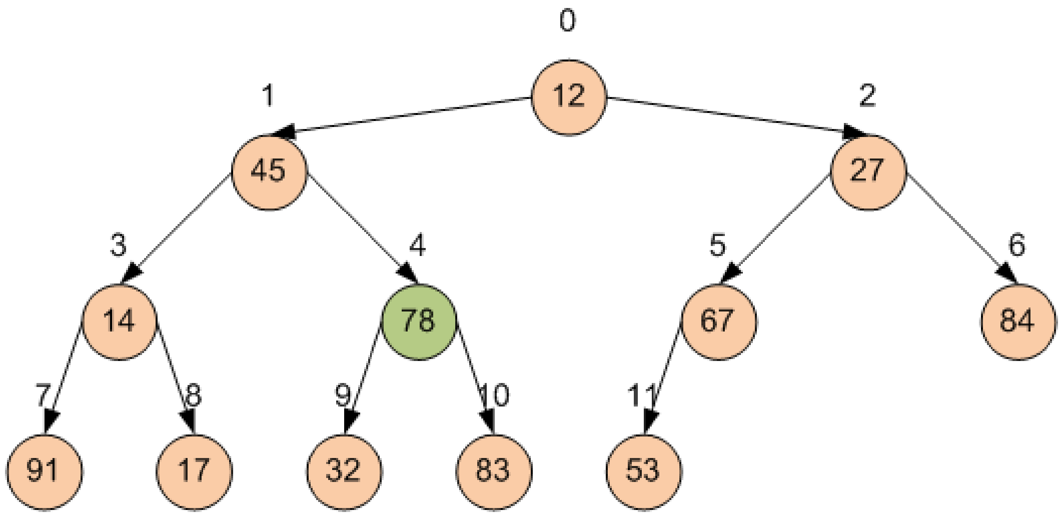


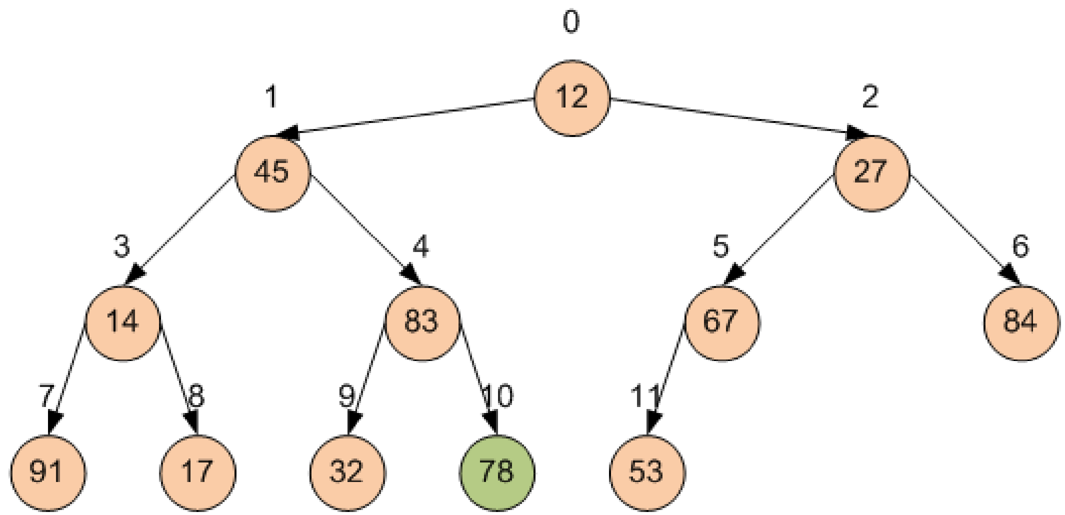
Heapsort

- **BUILDHEAP** (construction a heap using HEAPIFY)
- Complexity O(N/2)
- `j = N/2-1`
- `Use HEAPIFY(j)`
- `j=j-1`
- `Until j<0`

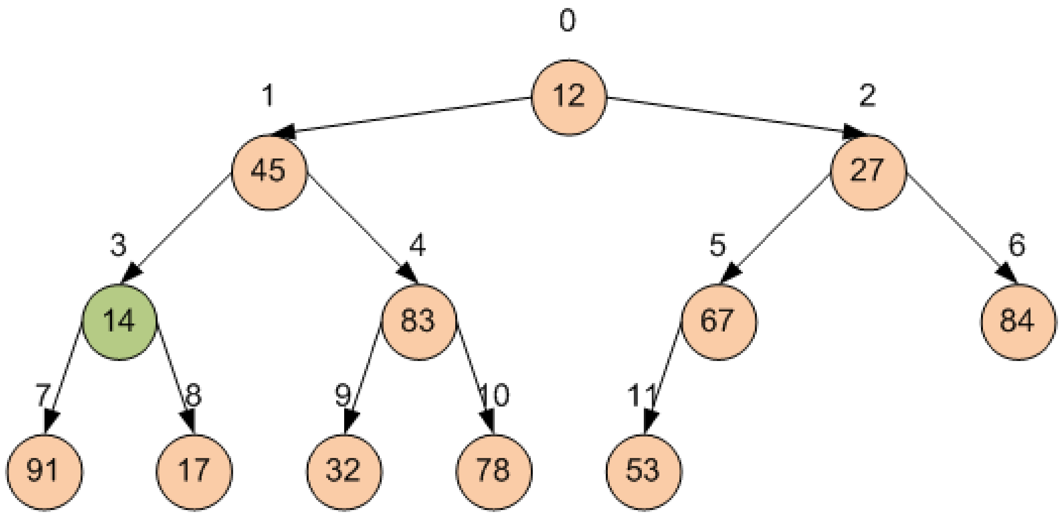


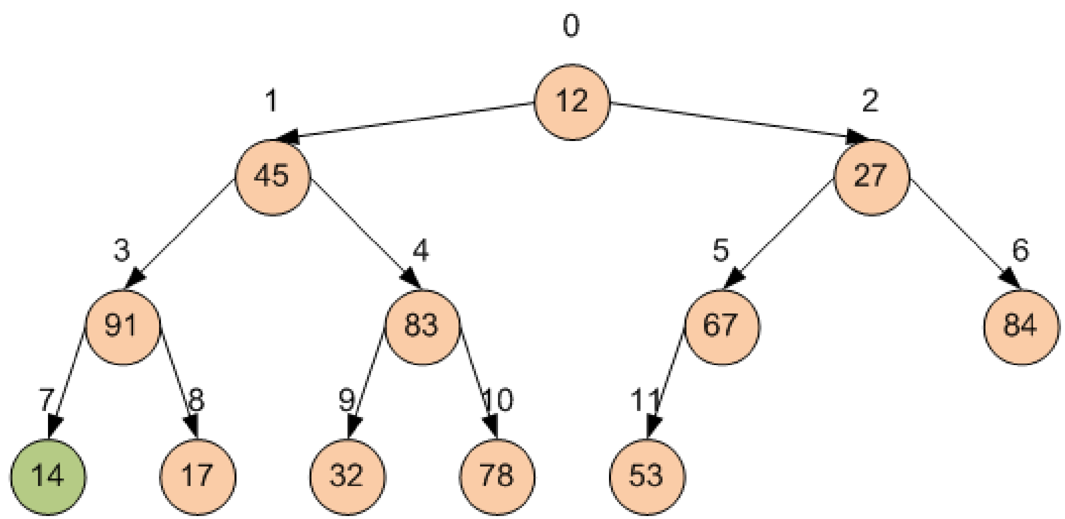
Heapsort


0
12
1
45
2
27
3
14
4
78
5
67
6
84
7
91
8
17
9
32
10
83
11
53

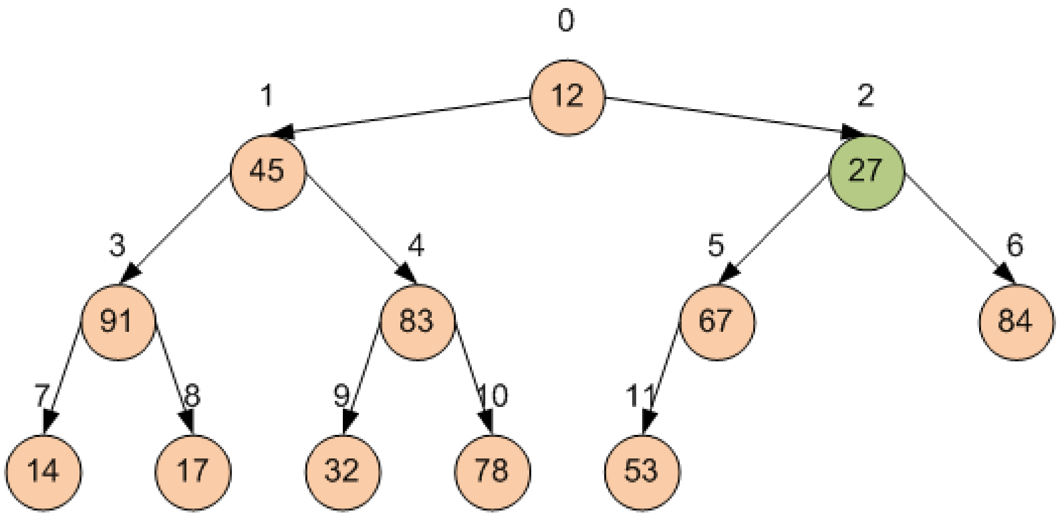

0
12
1
45
2
27
3
14
4
83
5
67
6
84
7
91
8
17
9
32
10
78
11
53

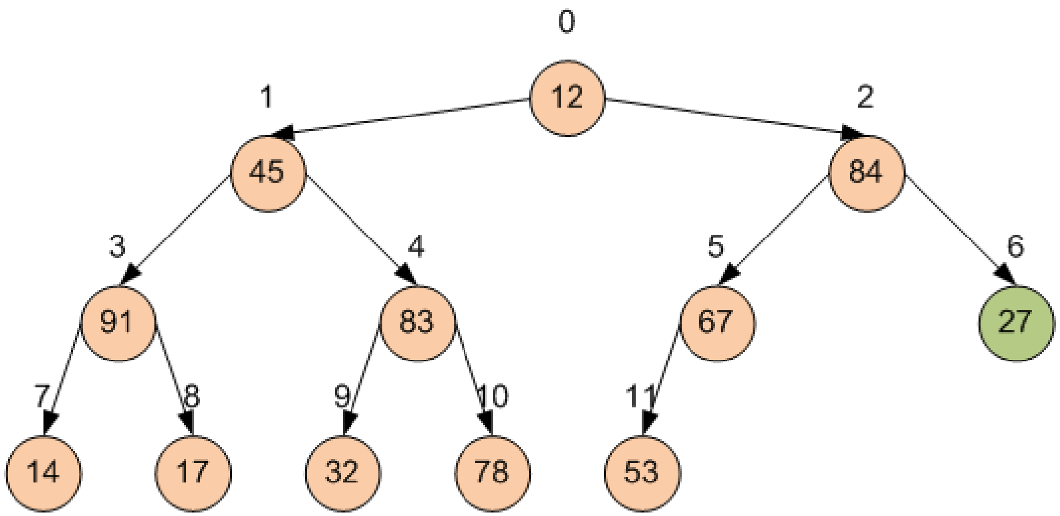
Heapsort


0
12
1
45
2
27
3
14
4
83
5
67
6
84
7
91
8
17
9
32
10
78
11
53

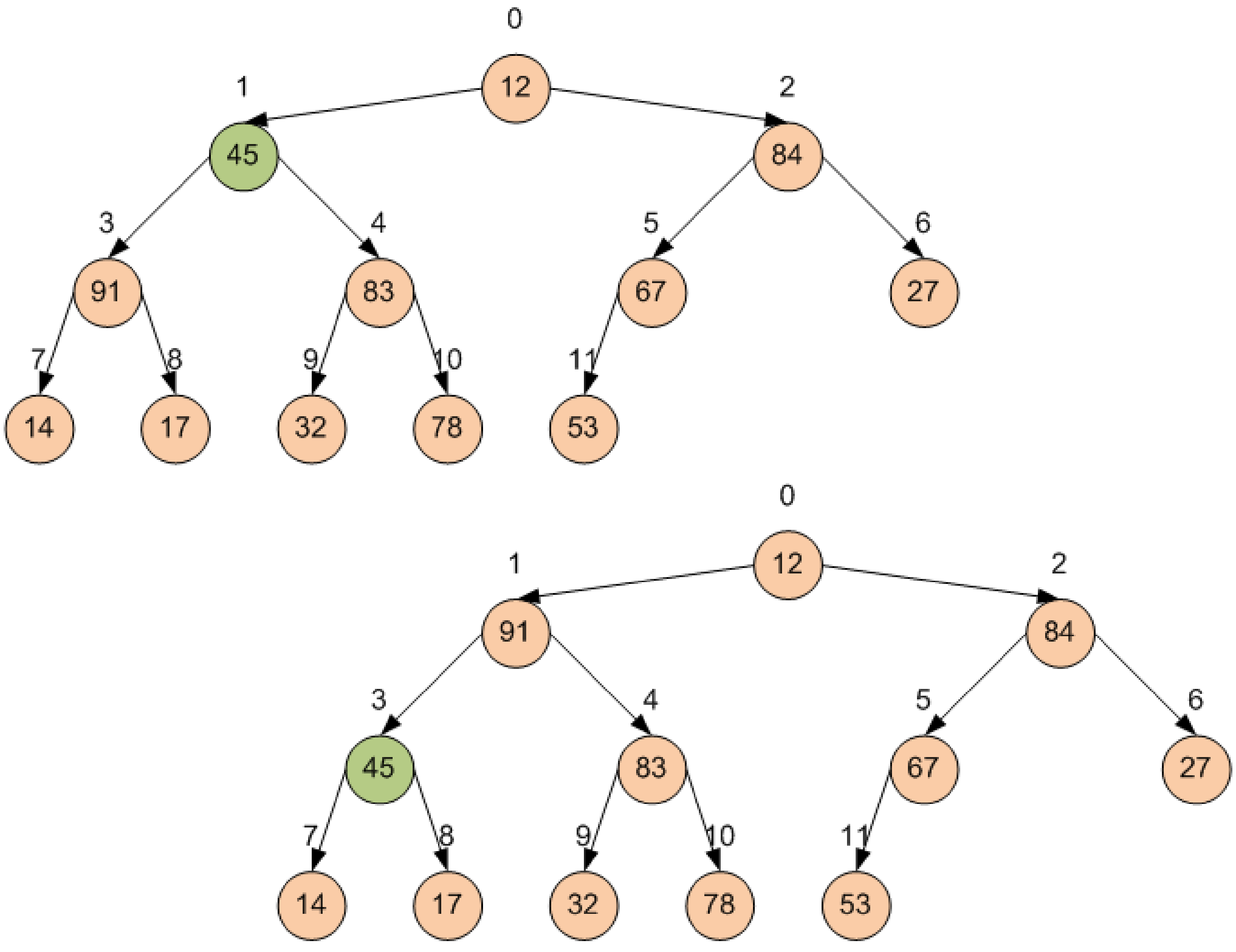

0
12
1
45
2
27
3
91
4
83
5
67
6
84
7
14
8
17
9
32
10
78
11
53

Heapsort


0
1
2
12
45
27
3
4
5
6
91
83
67
84
7
8
9
10
11
14
17
32
78
53

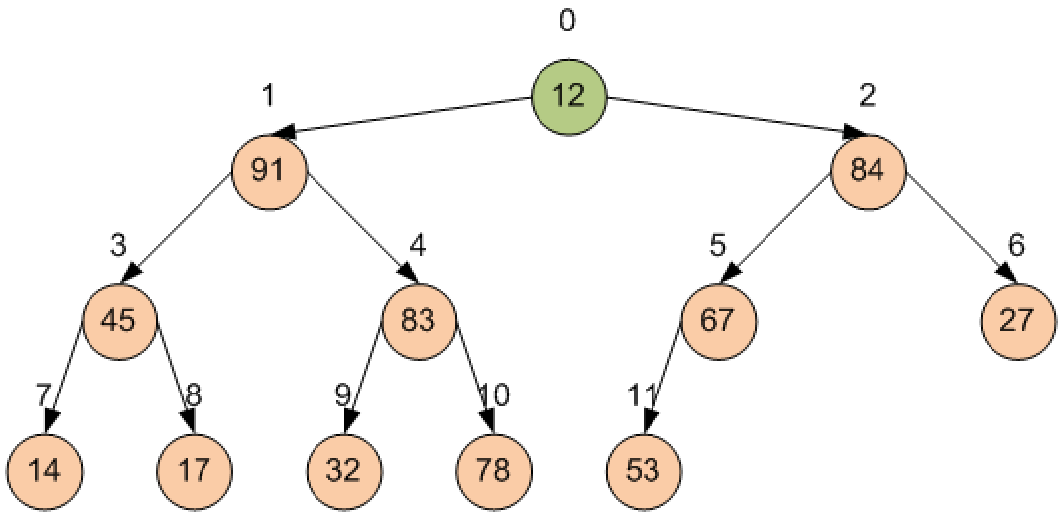

0
1
2
12
45
84
3
4
5
6
91
83
67
27
7
8
9
10
11
14
17
32
78
53

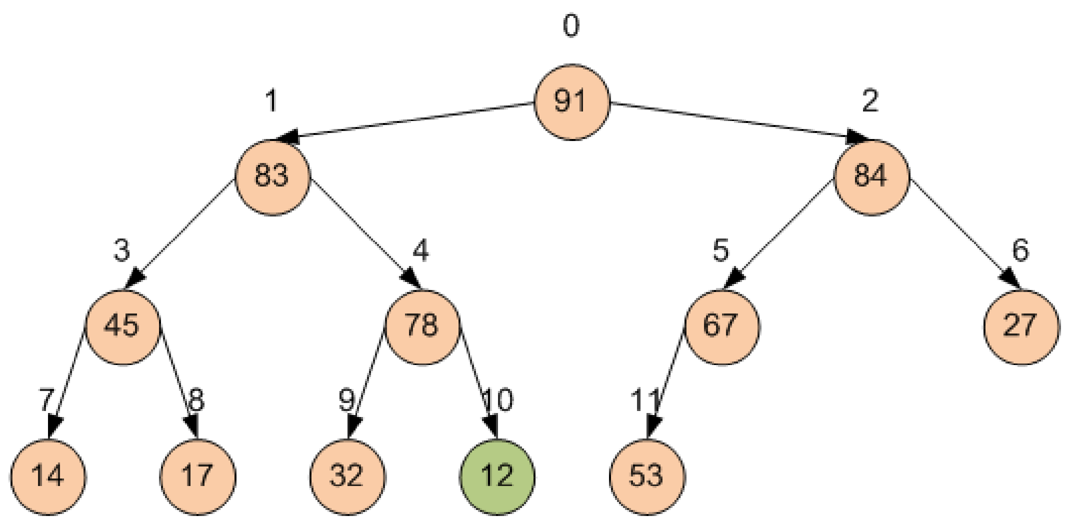
Heapsort



0
12
1
45
2
84
3
91
4
83
5
67
6
27
7
14
8
17
9
32
10
78
11
53
0
12
1
91
2
84
3
45
4
83
5
67
6
27
7
14
8
17
9
32
10
78
11
53

Heapsort


0
12
1
2
91
84
3
4
5
6
45
83
67
27
7
8
9
10
11
14
17
32
78
53

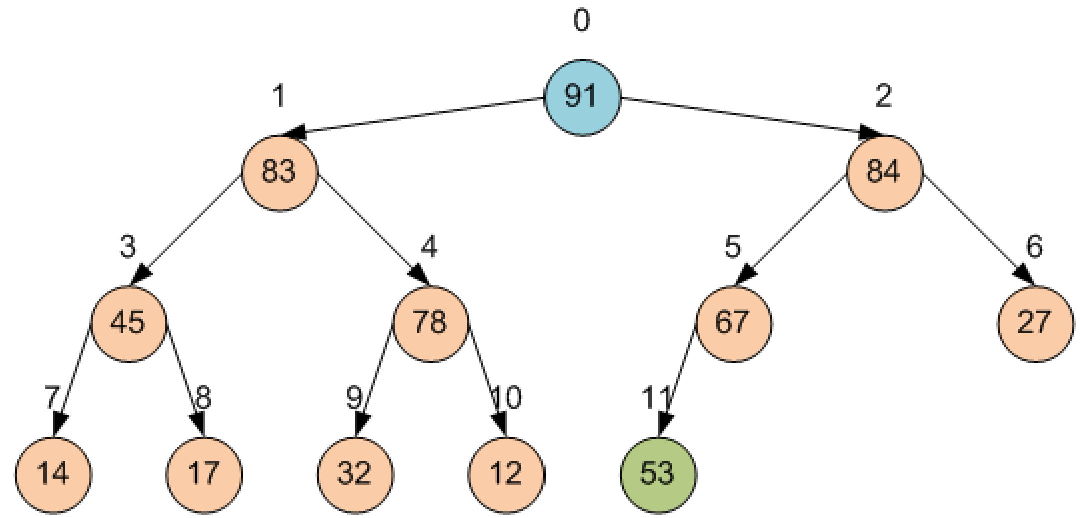

0
91
1
2
83
84
3
4
5
6
45
78
67
27
7
8
9
10
11
14
17
32
12
53

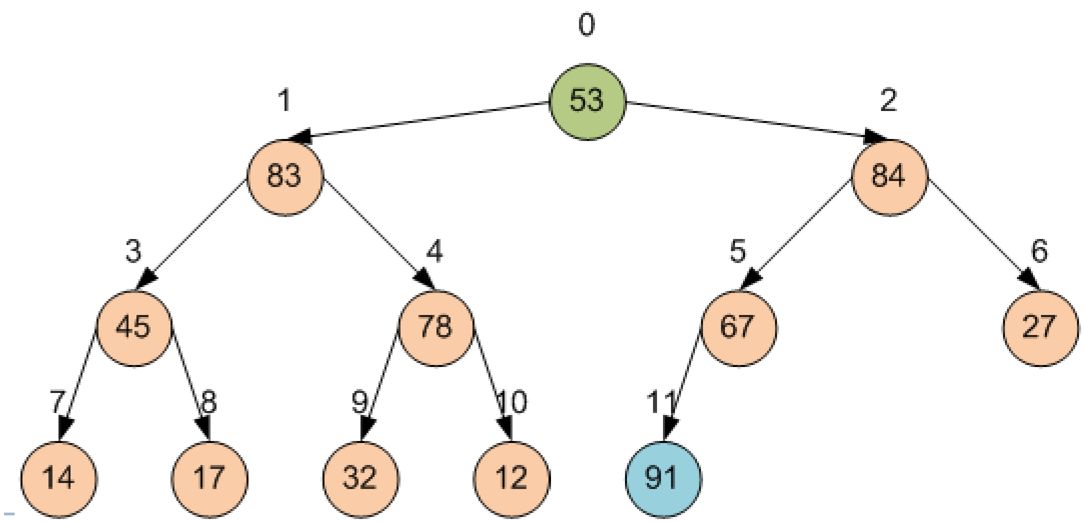
Heapsort

- **HEAPSORT**
- Swap A[0] and A[N-1] (the maximum element becames the last
- N=N-1
- Restore the heap property for A[0]…A[N-1] by using HEAPIFY(0)
- Until N=0
- Complexity *N*ln*N*

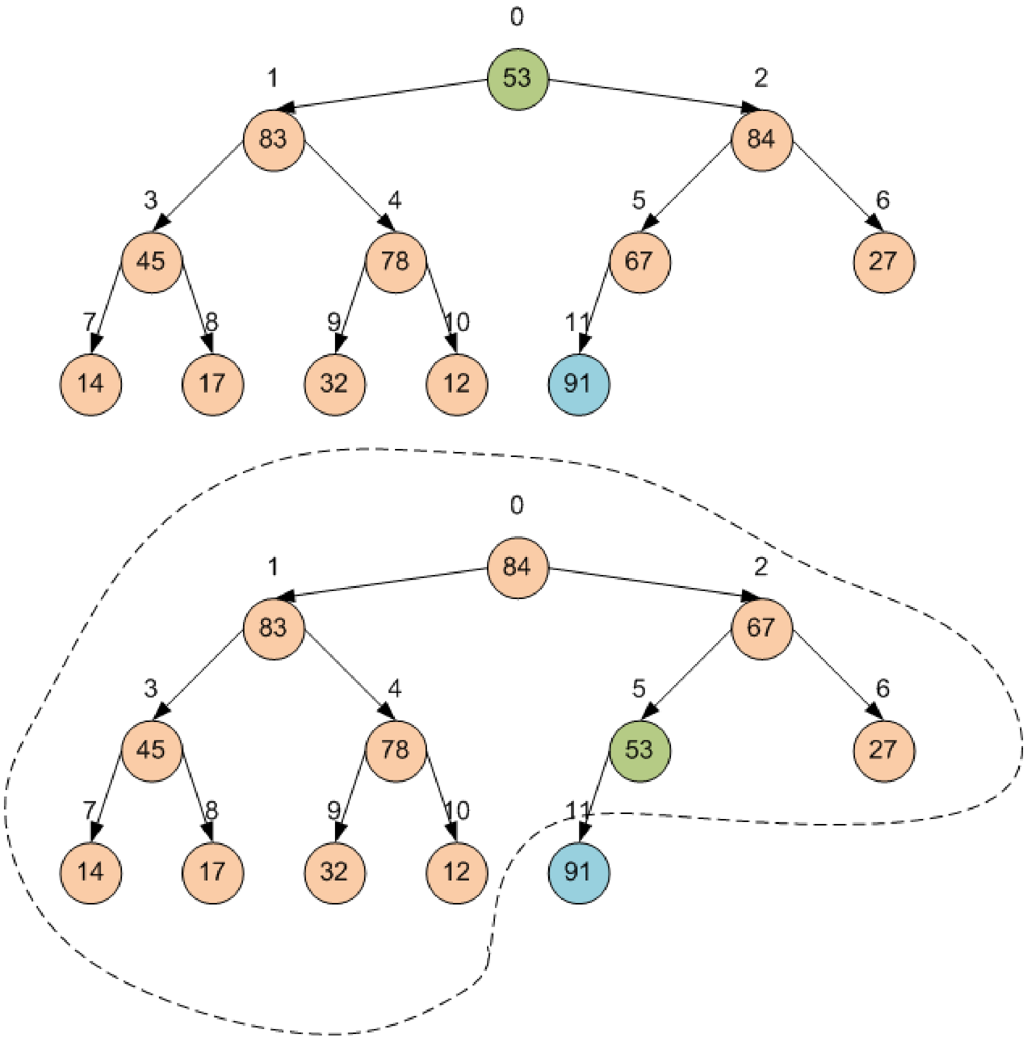
Heapsort

- **HEAPSORT**

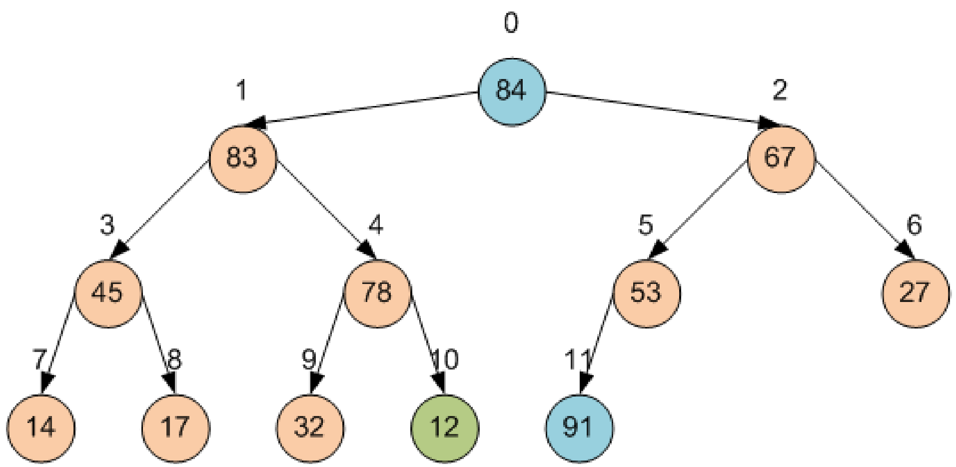


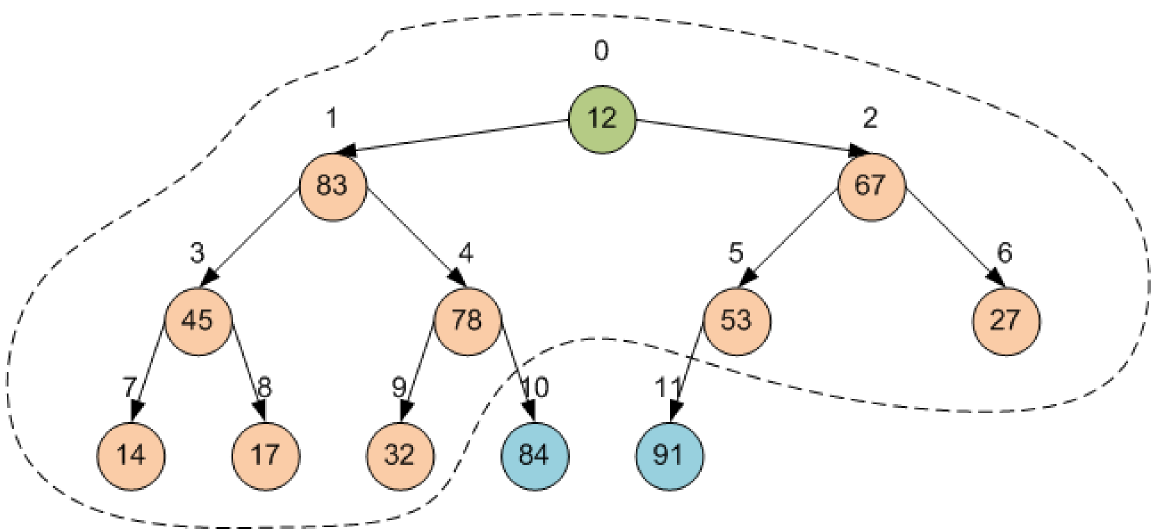


Heapsort

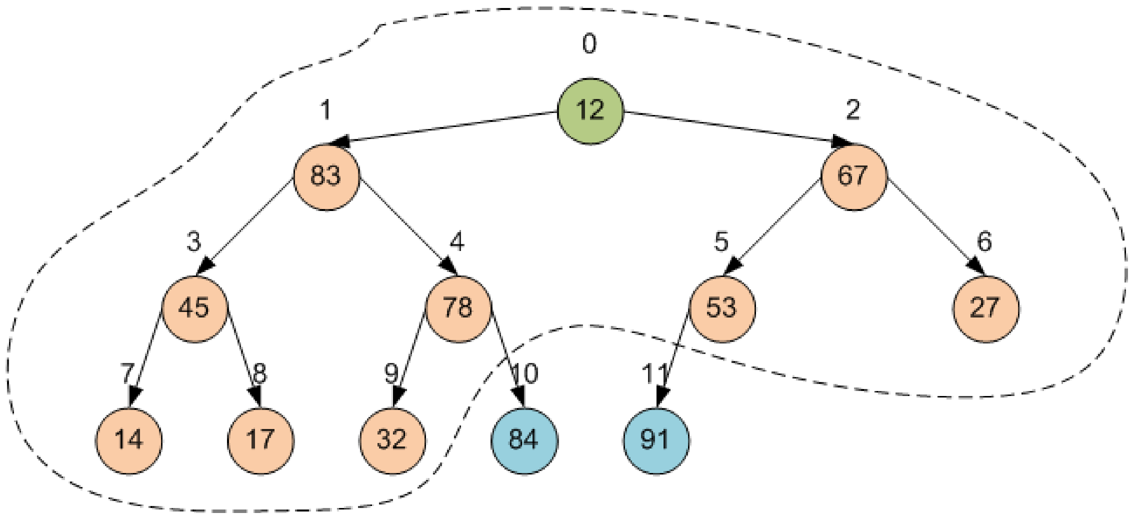

0
53
1
2
83
84
3
4
5
6
45
78
67
27
7
8
9
10
11
14
17
32
12
91
0
84
1
2
83
67
3
4
5
6
45
78
53
27
7
8
9
10
11
14
17
32
12
91

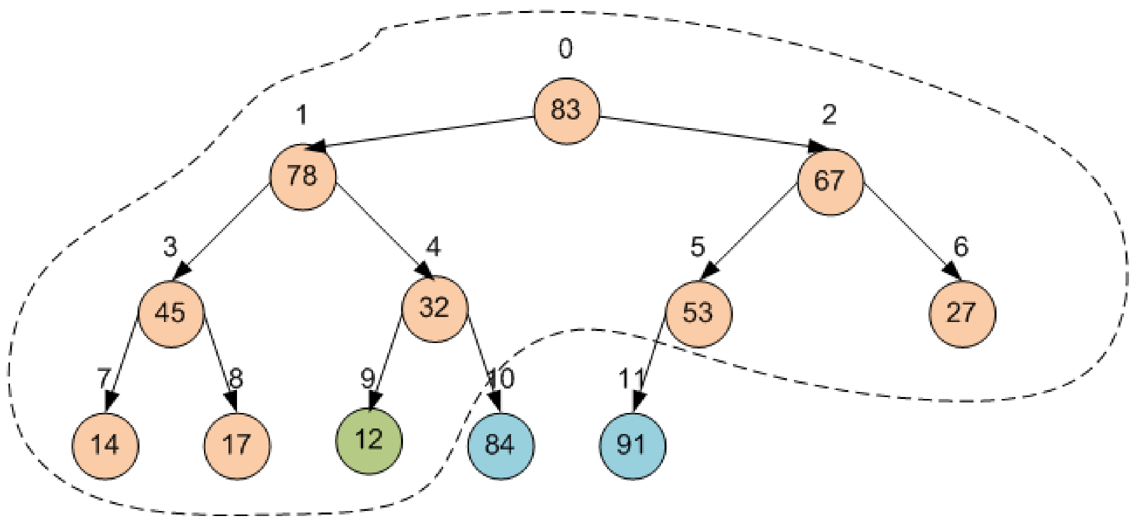
Heapsort


0
84
1
2
83
67
3
4
5
6
45
78
53
27
7
8
9
10
11
14
17
32
12
91

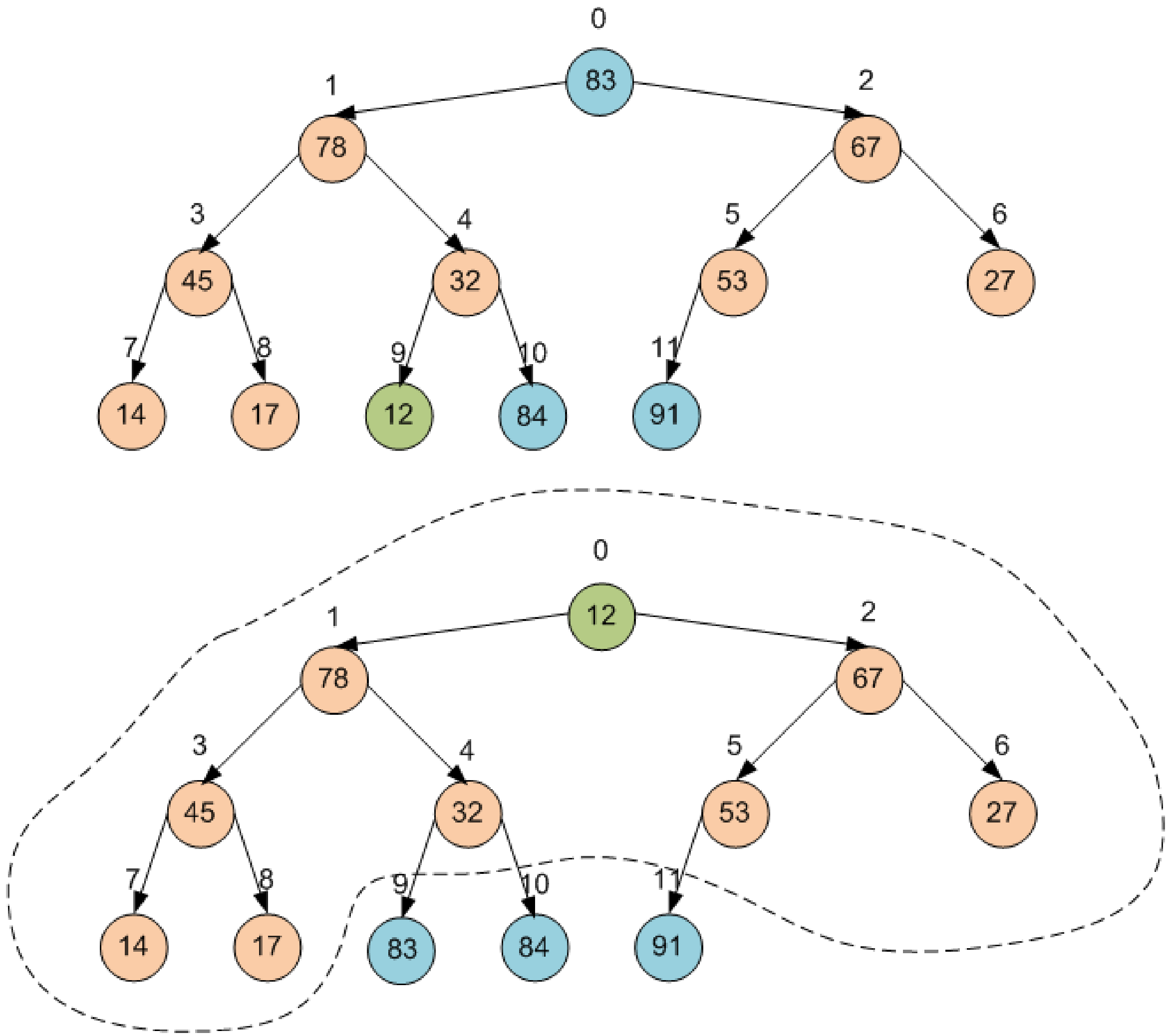

0
12
1
2
83
67
3
4
5
6
45
78
53
27
7
8
9
10
11
14
17
32
84
91

Heapsort

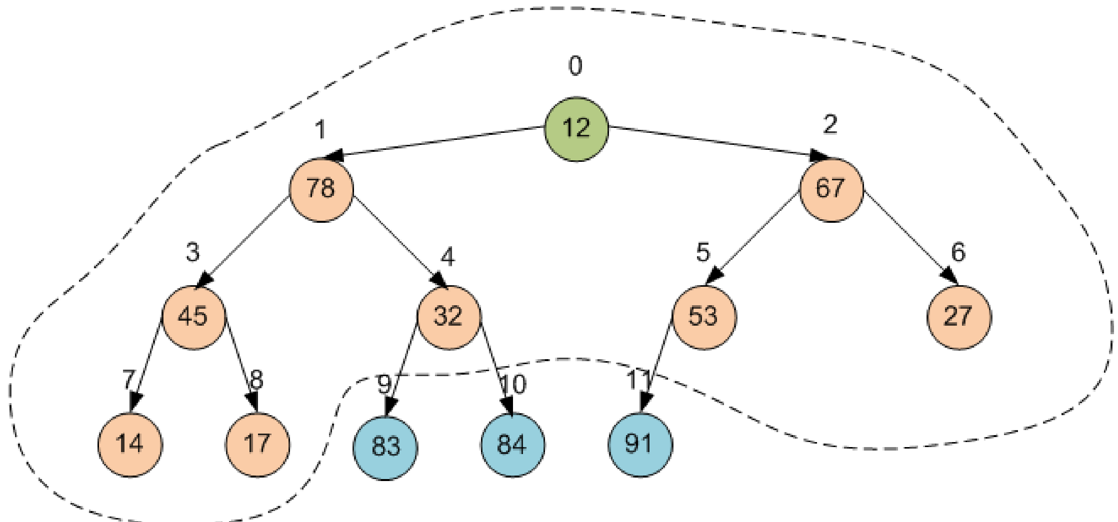

0
12
1
83
2
67
3
45
4
78
5
53
6
27
7
14
8
17
9
32
10
84
11
91

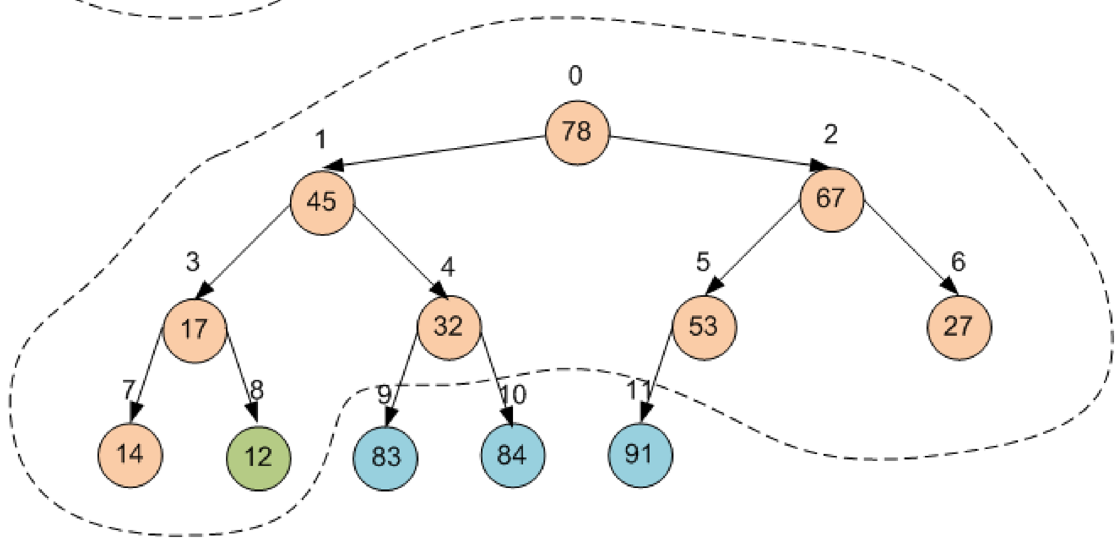

0
83
1
78
2
67
3
45
4
32
5
53
6
27
7
14
8
17
9
12
10
84
11
91

Heapsort


0
83
1
78
2
67
3
45
4
32
5
53
6
27
7
14
8
17
9
12
10
84
11
91
0
12
1
78
2
67
3
45
4
32
5
53
6
27
7
14
8
17
9
83
10
84
11
91

Heapsort


0
12
1
78
2
67
3
45
4
32
5
53
6
27
7
14
8
17
9
83
10
84
11
91

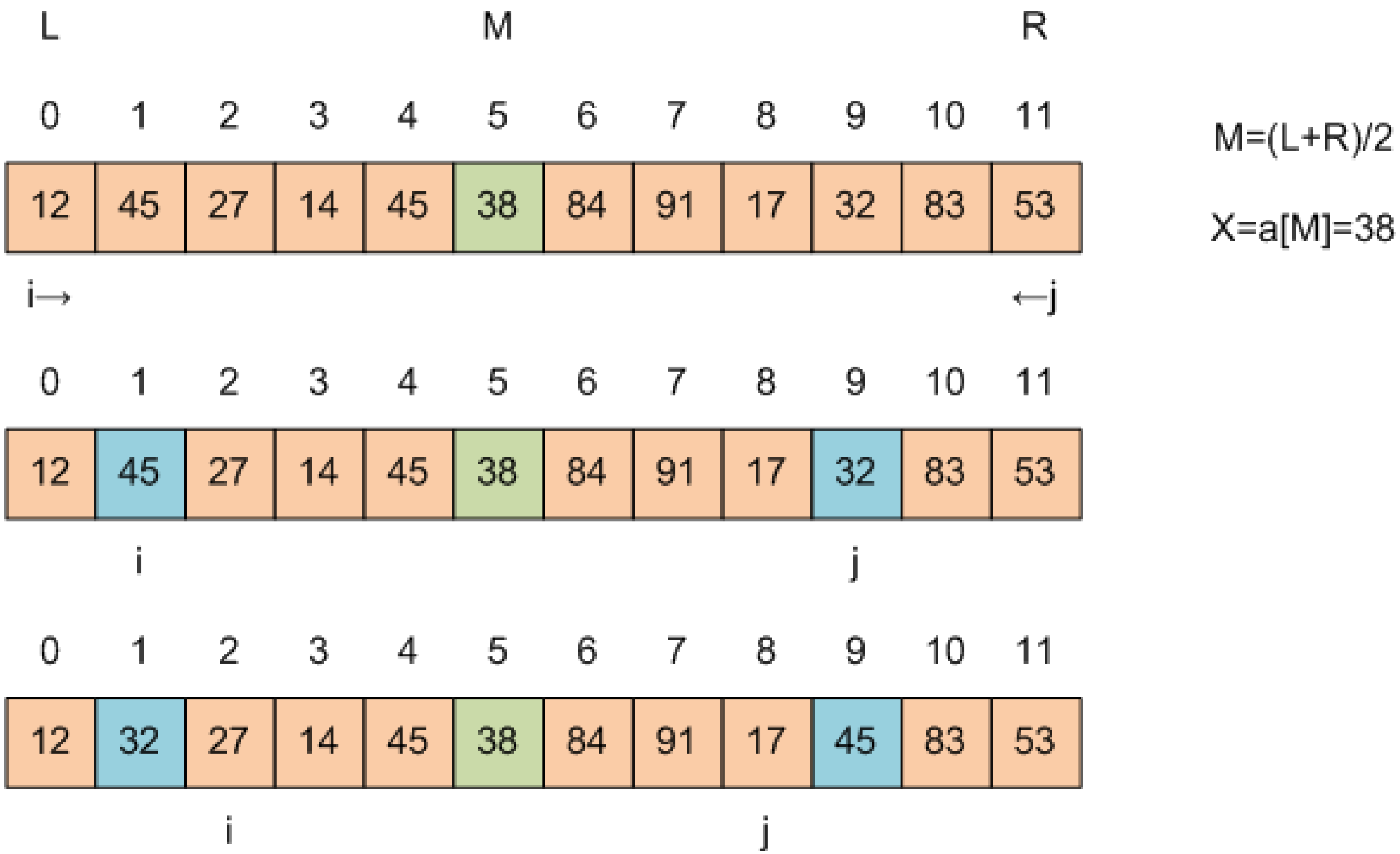

0
78
1
45
2
67
3
17
4
32
5
53
6
27
7
14
8
12
9
83
10
84
11
91

Quicksort (Hoare algorithm)

- *Hoare, C. A. R. (1961). "Algorithm 64: Quicksort". Comm. ACM. **4** (7): 321.*

- **Idea**.
- Pick an element, called a ***pivot***, from the array.
- ***Partitioning***: reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way).
- ***Recursively*** apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values.

Quicksort (Hoare algorithm)



Quicksort (Hoare algorithm)

0	1	2	3	4	5	6	7	8	9	10	11
12	32	27	14	45	38	84	91	17	45	83	53
				i				j			

0	1	2	3	4	5	6	7	8	9	10	11
12	32	27	14	17	38	84	91	45	45	83	53
					i		j				

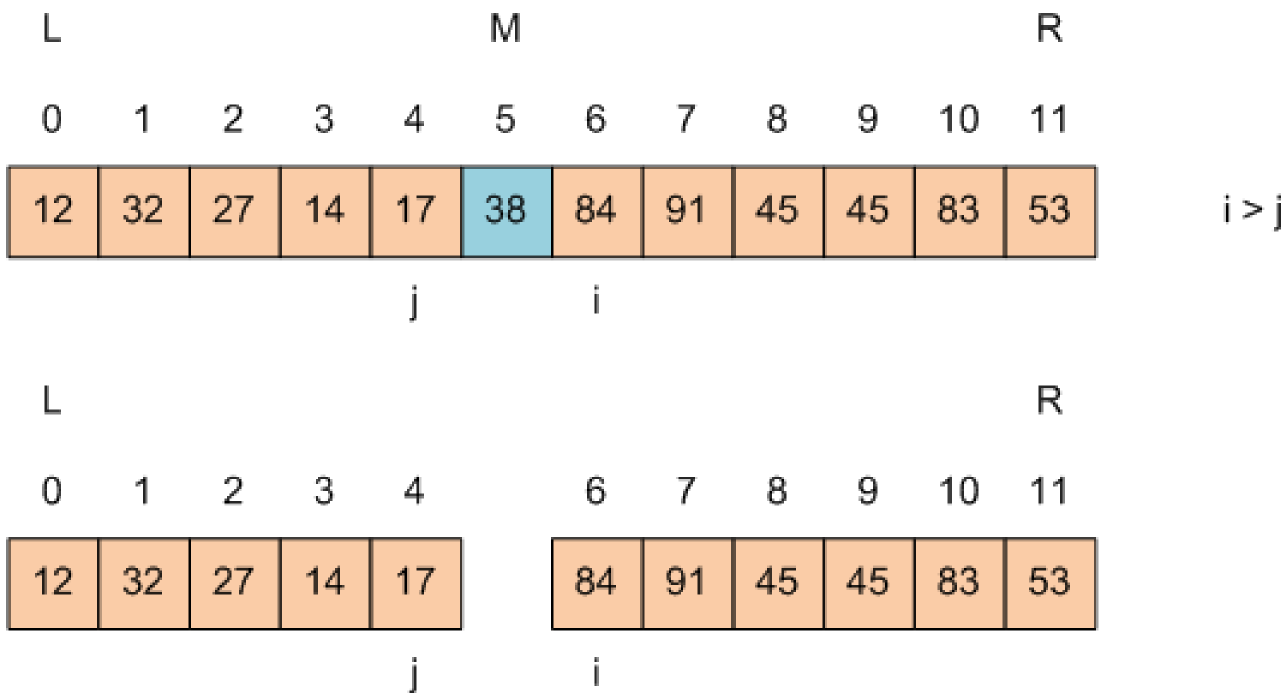
0	1	2	3	4	5	6	7	8	9	10	11
12	32	27	14	17	38	84	91	45	45	83	53
					i j						

0	1	2	3	4	5	6	7	8	9	10	11
12	32	27	14	17	38	84	91	45	45	83	53
				j		i					

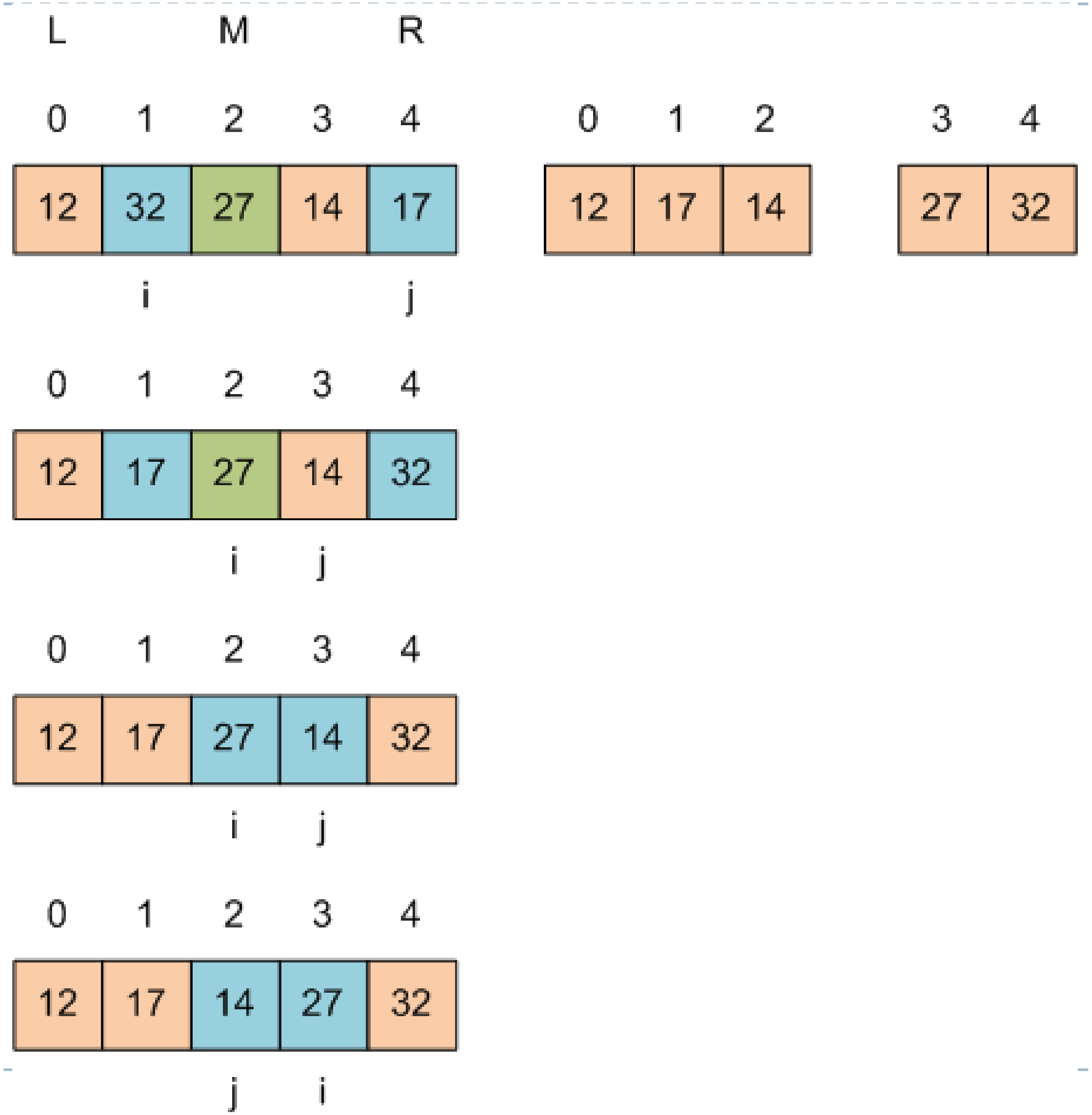
i > j

Quicksort (Hoare algorithm)

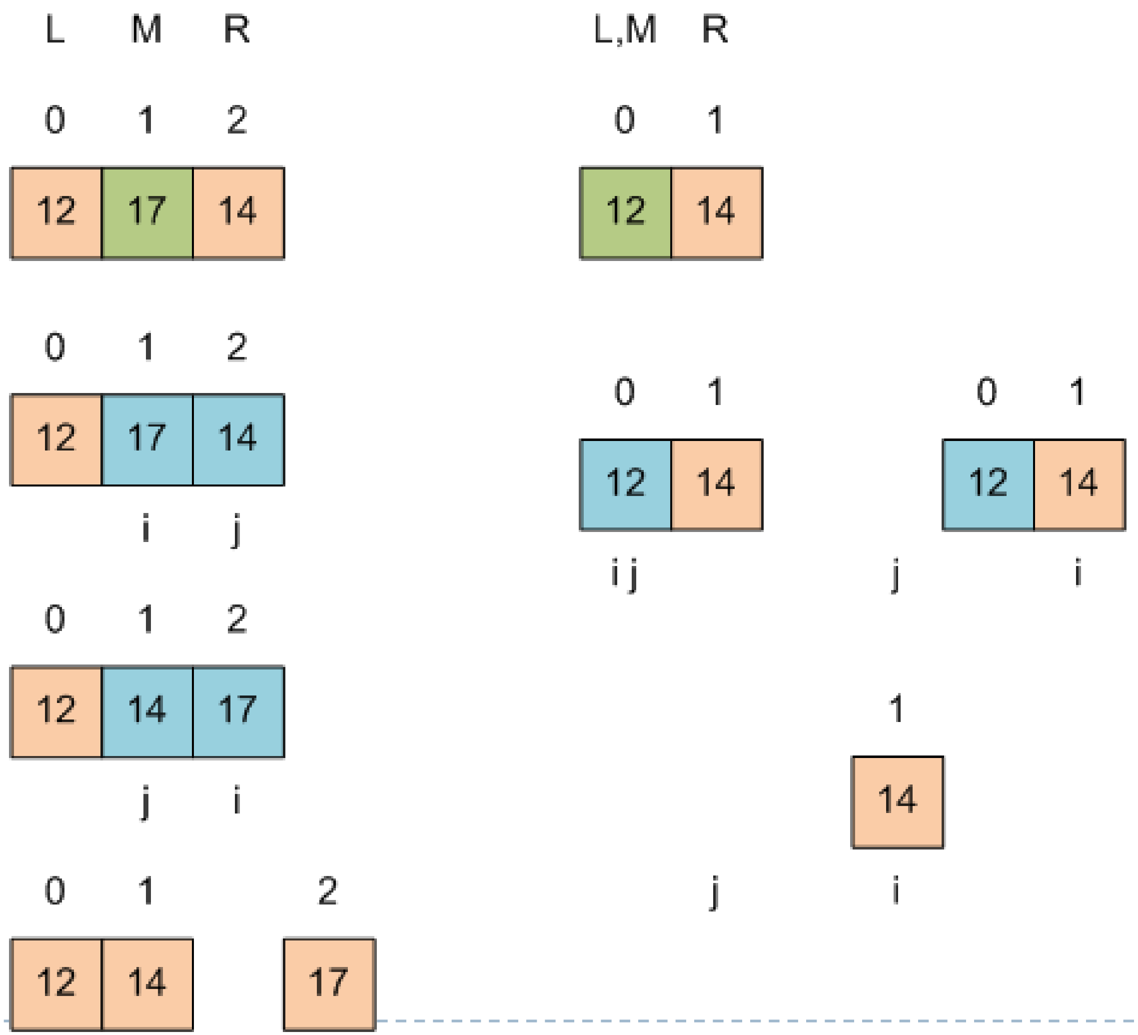
- Recursive calls (L,j) and (I,R)



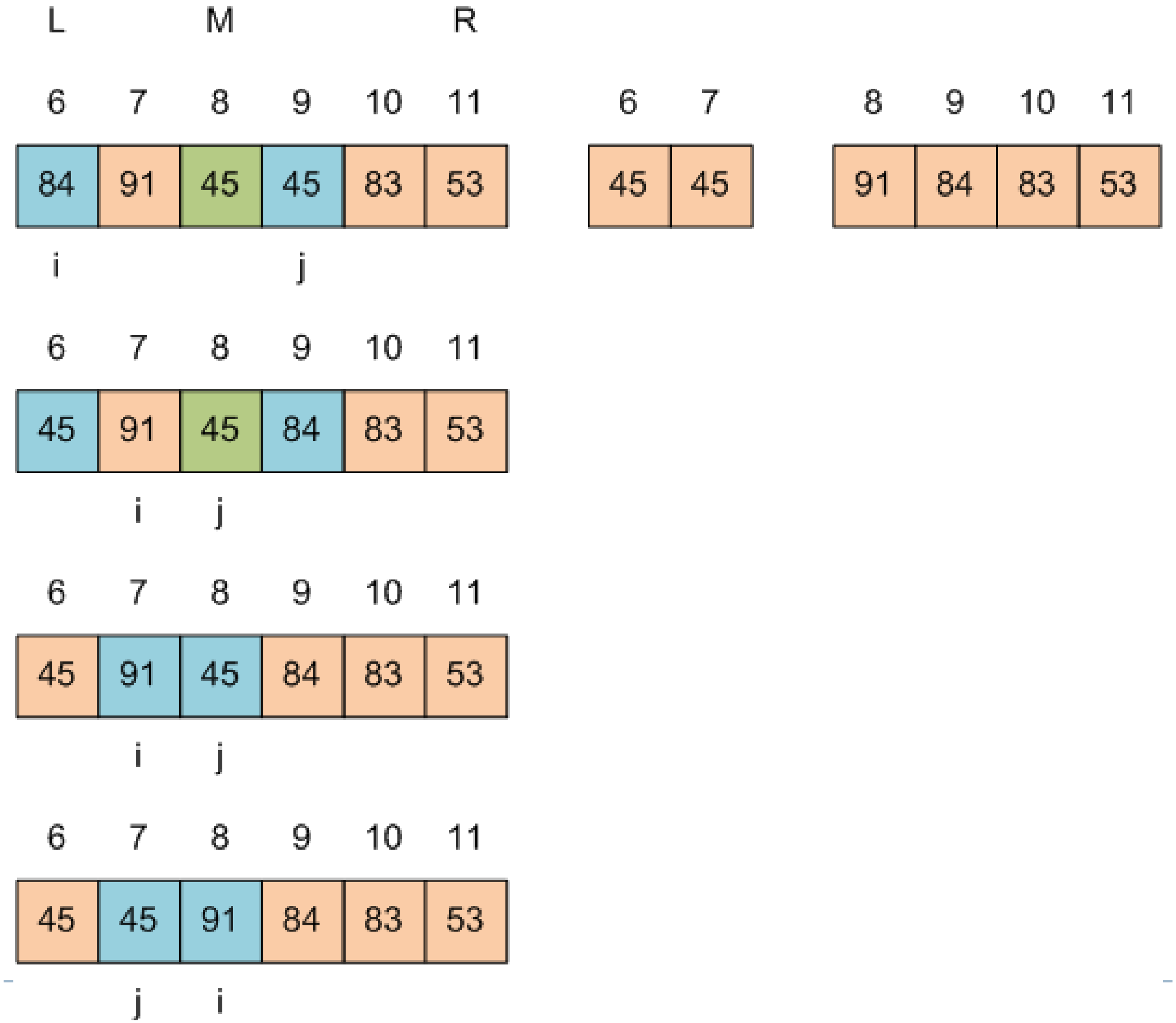
Quicksort (Hoare algorithm)



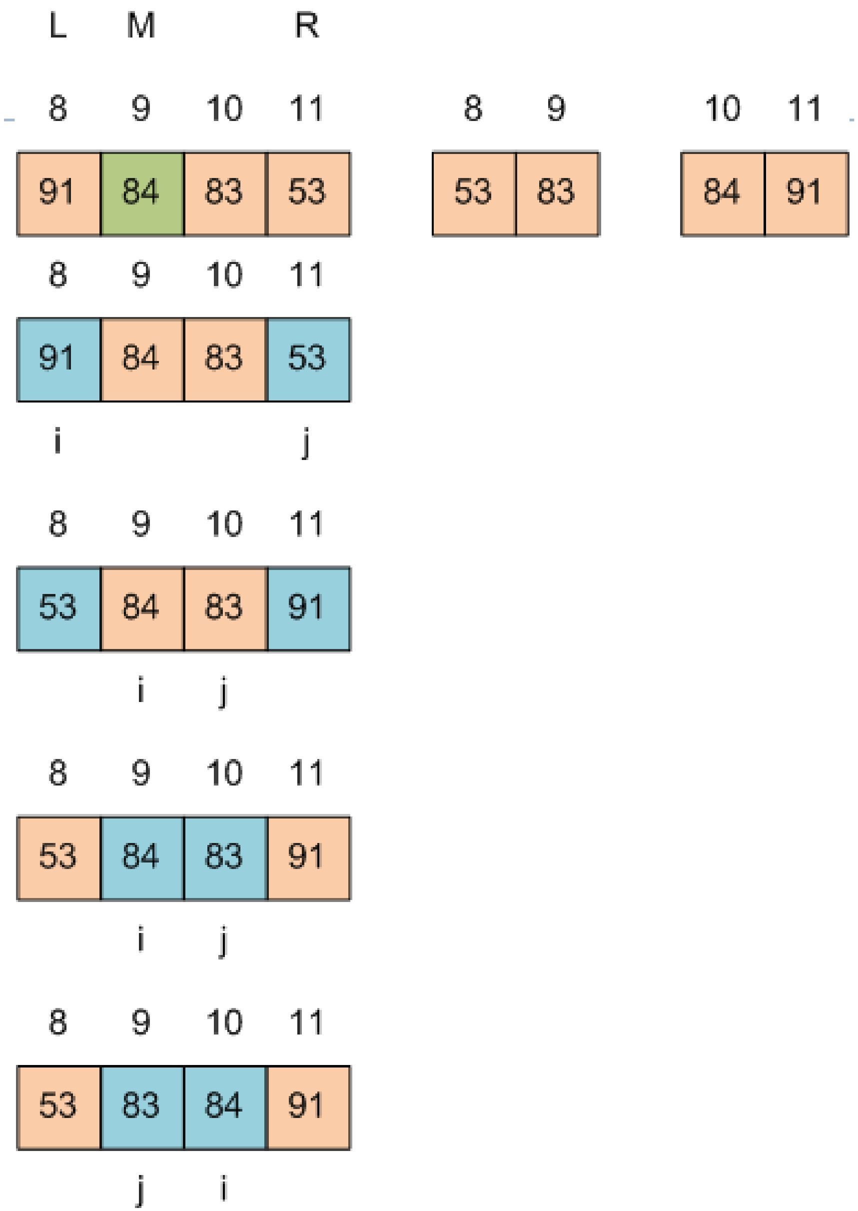
Quicksort (Hoare algorithm)



Quicksort (Hoare algorithm)

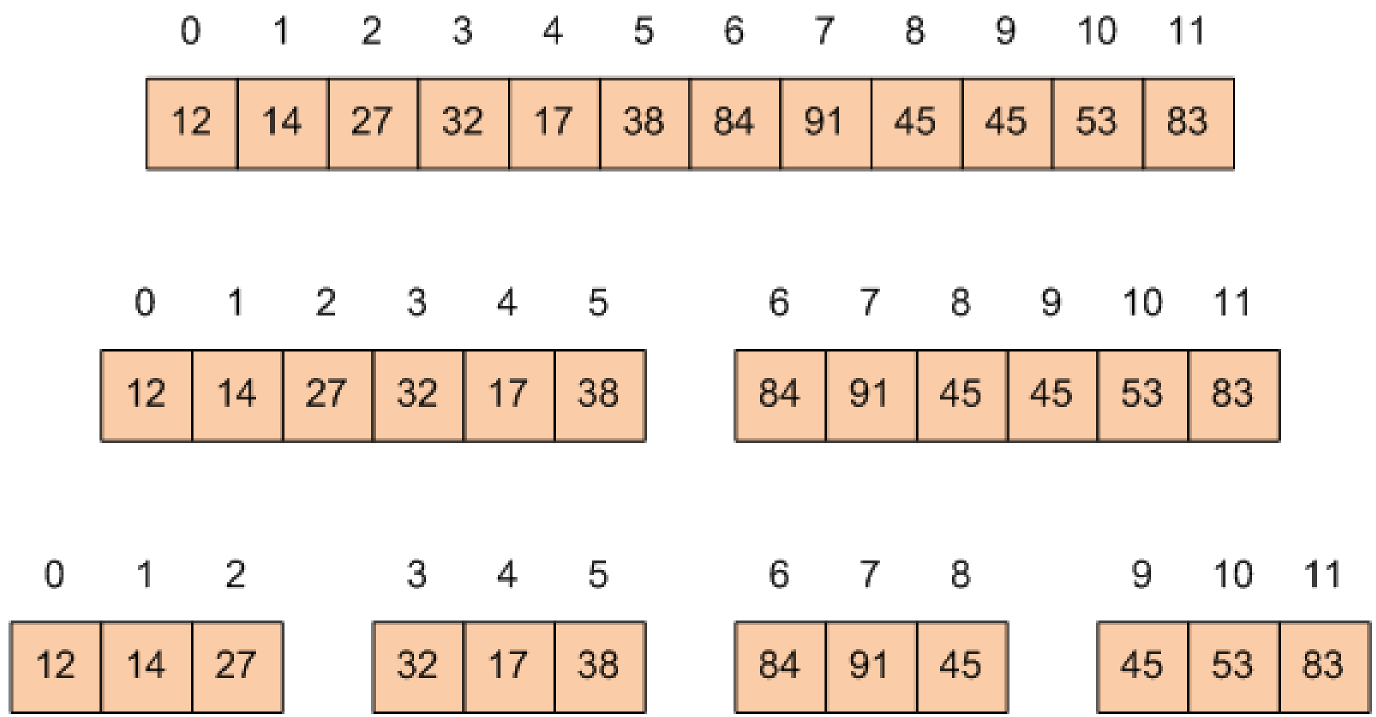


Quicksort
(Hoare algorithm)



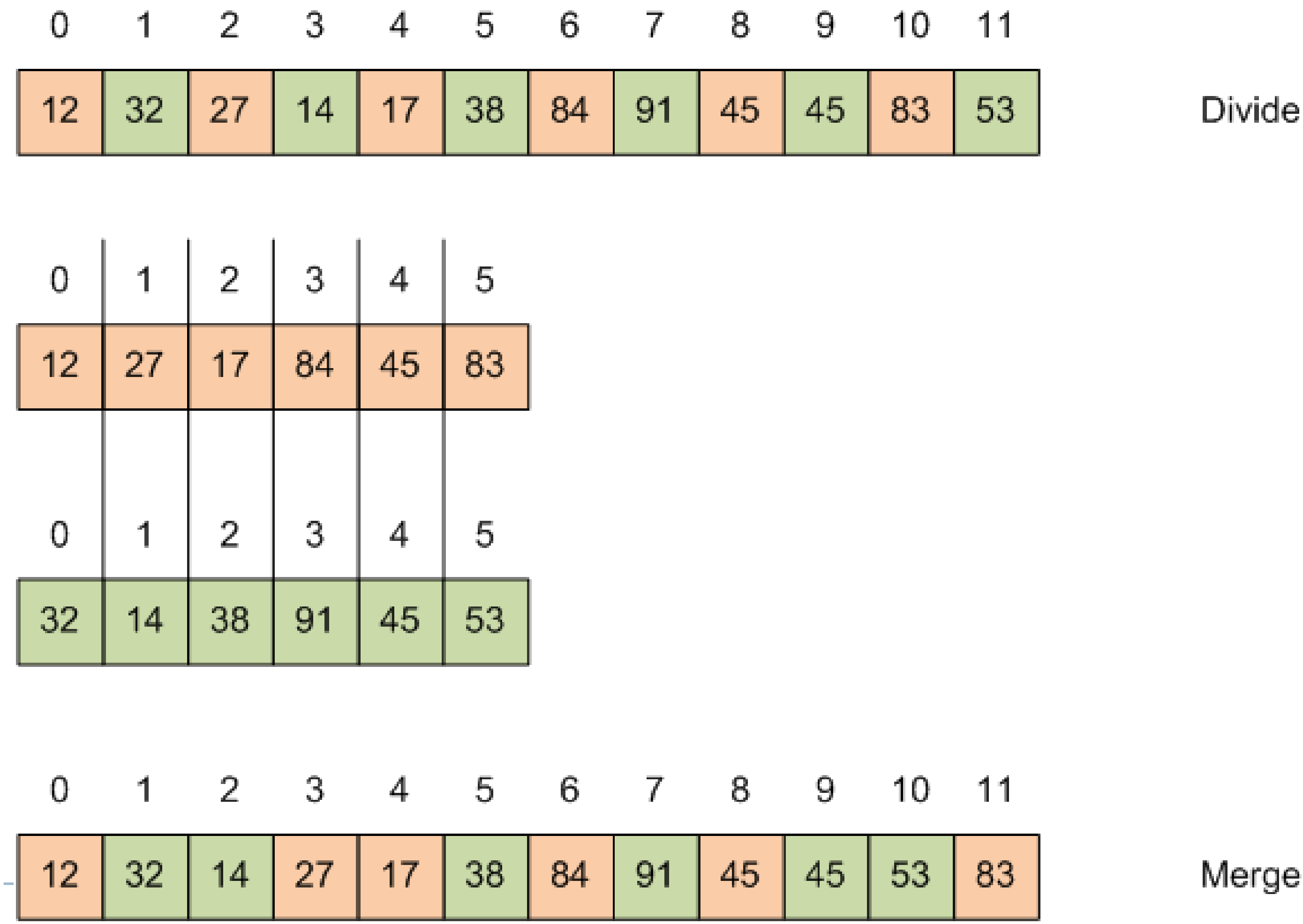
Merge sorting

- Divide-and-conquer



Merge sorting

- Direct merge sorting



Merge sorting

0	1	2	3	4	5	6	7	8	9	10	11
12	32	14	27	17	38	84	91	45	45	53	83

0	1	2	3	4	5
12	32	17	38	45	45

0	1	2	3	4	5
14	27	84	91	53	83

0	1	2	3	4	5	6	7	8	9	10	11
12	14	27	32	17	38	84	91	45	45	53	83

Merge sorting

0	1	2	3	4	5	6	7	8	9	10	11
12	14	27	32	17	38	84	91	45	45	53	83

0	1	2	3	4	5	6	7
12	14	27	32	45	45	53	83

0	1	2	3
17	38	84	91

0	1	2	3	4	5	6	7	8	9	10	11
12	14	17	27	32	38	84	91	45	45	53	83

Merge sorting

0	1	2	3	4	5	6	7	8	9	10	11
12	14	17	27	32	38	84	91	45	45	53	83

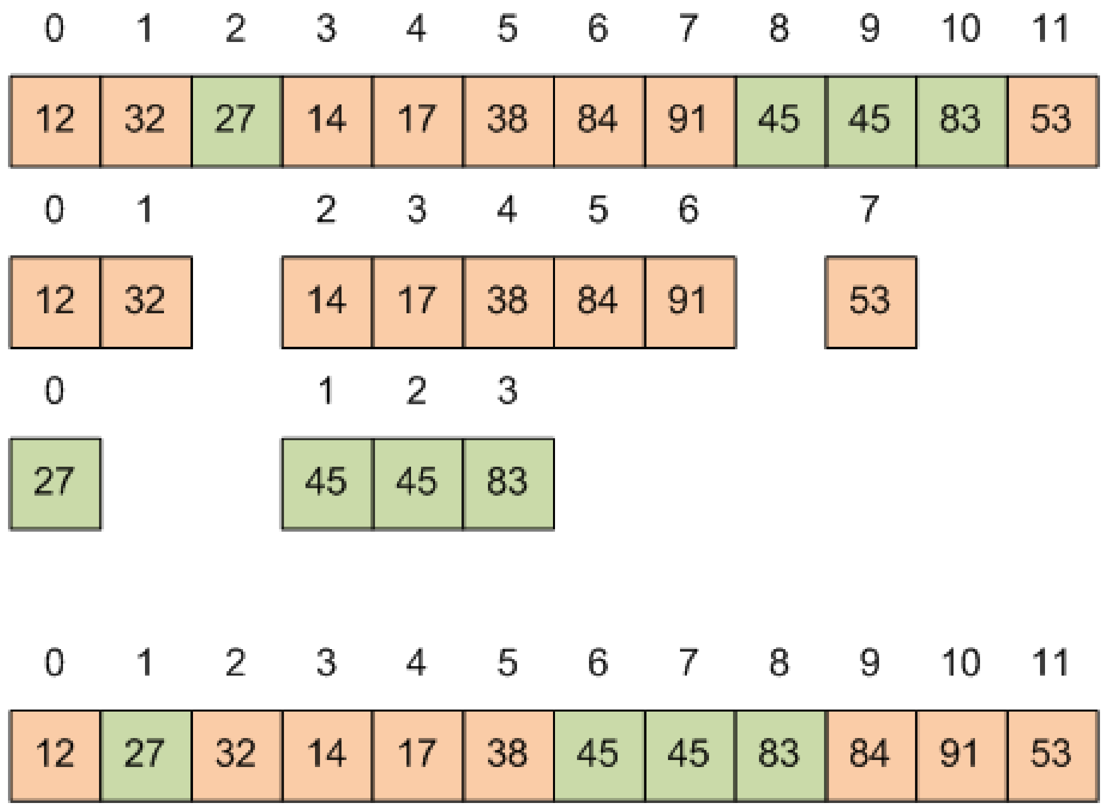
0	1	2	3	4	5	6	7
12	14	17	27	32	38	84	91

0	1	2	3
45	45	53	83

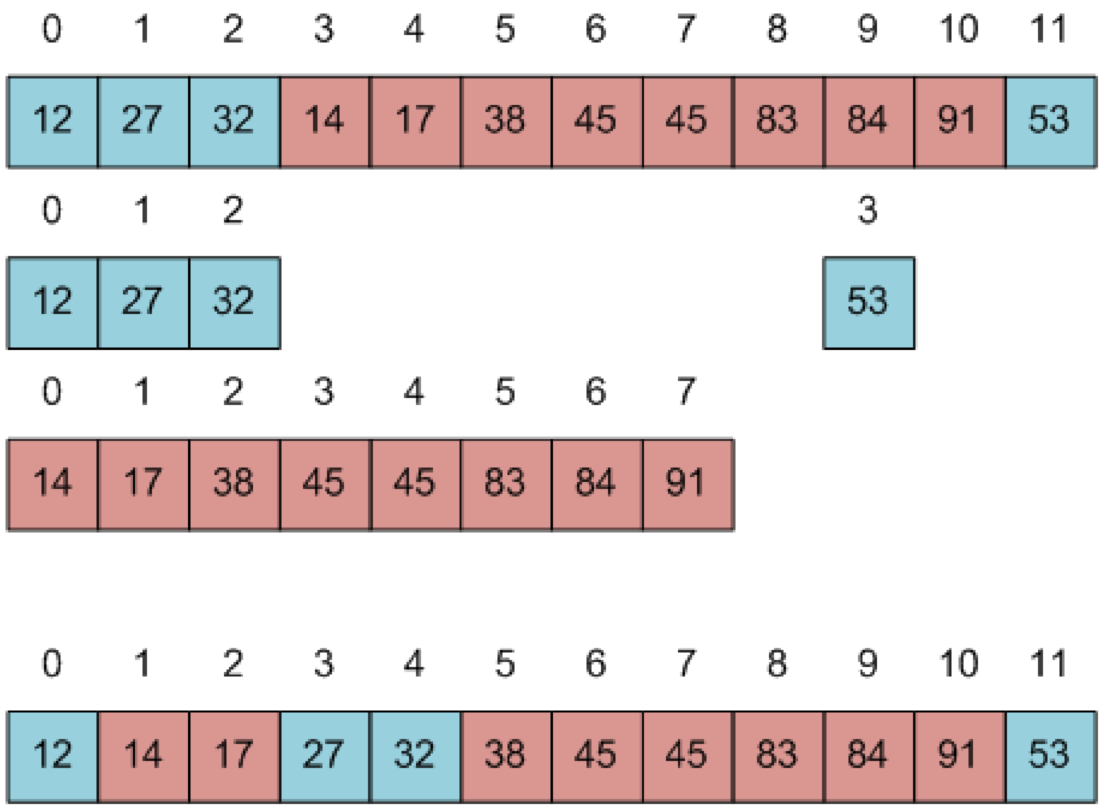
0	1	2	3	4	5	6	7	8	9	10	11
12	14	17	27	32	38	45	45	53	83	84	91

Merge sorting

- Natural merge: use decreasing subarrays



Merge sorting



Merge sorting

0	1	2	3	4	5	6	7	8	9	10	11
12	14	17	27	32	38	45	45	83	84	91	53

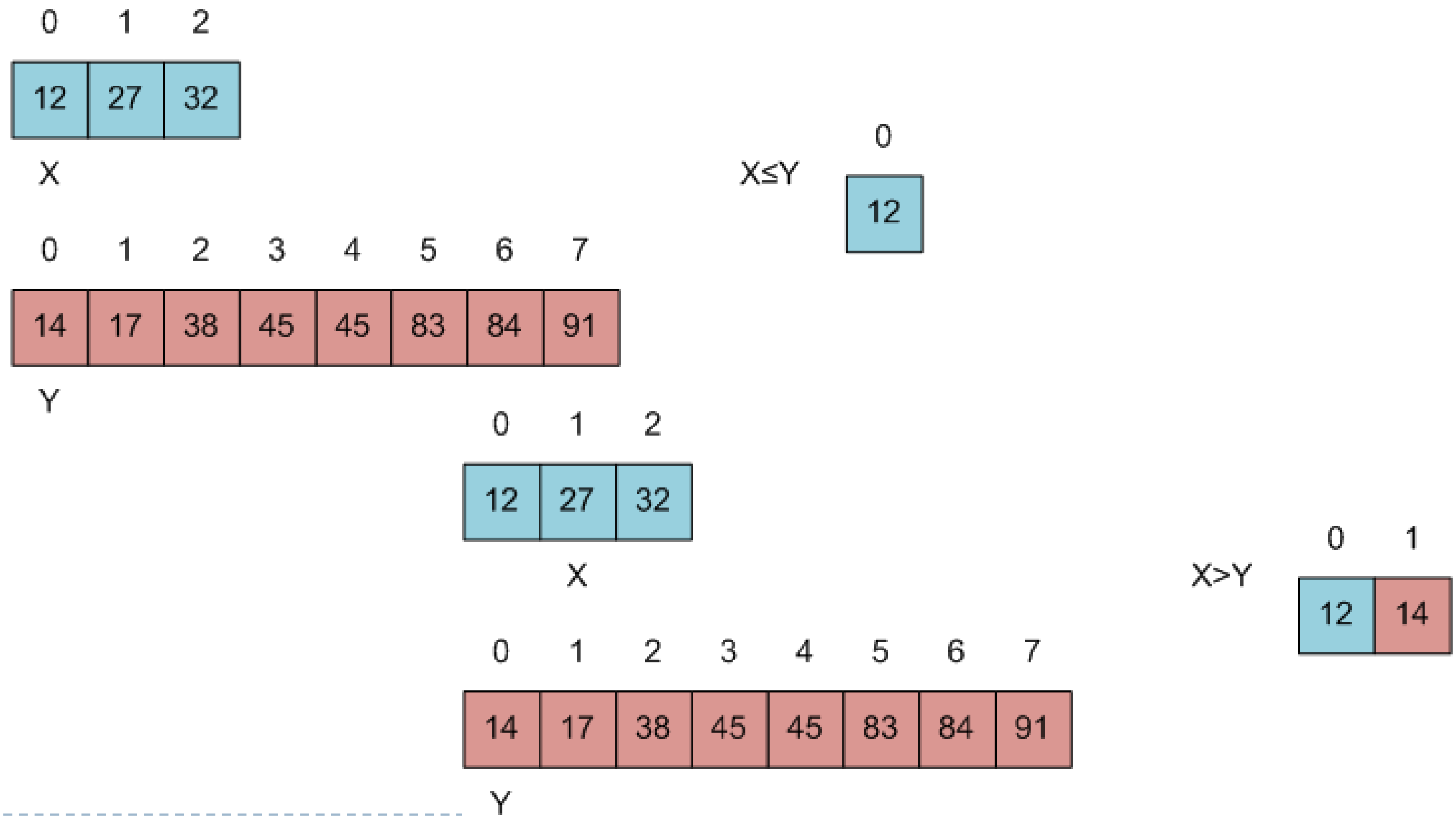
0	1	2	3	4	5	6	7	8	9	10
12	14	17	27	32	38	45	45	83	84	91

1
53

0	1	2	3	4	5	6	7	8	9	10	11
12	14	17	27	32	38	45	45	53	83	84	91

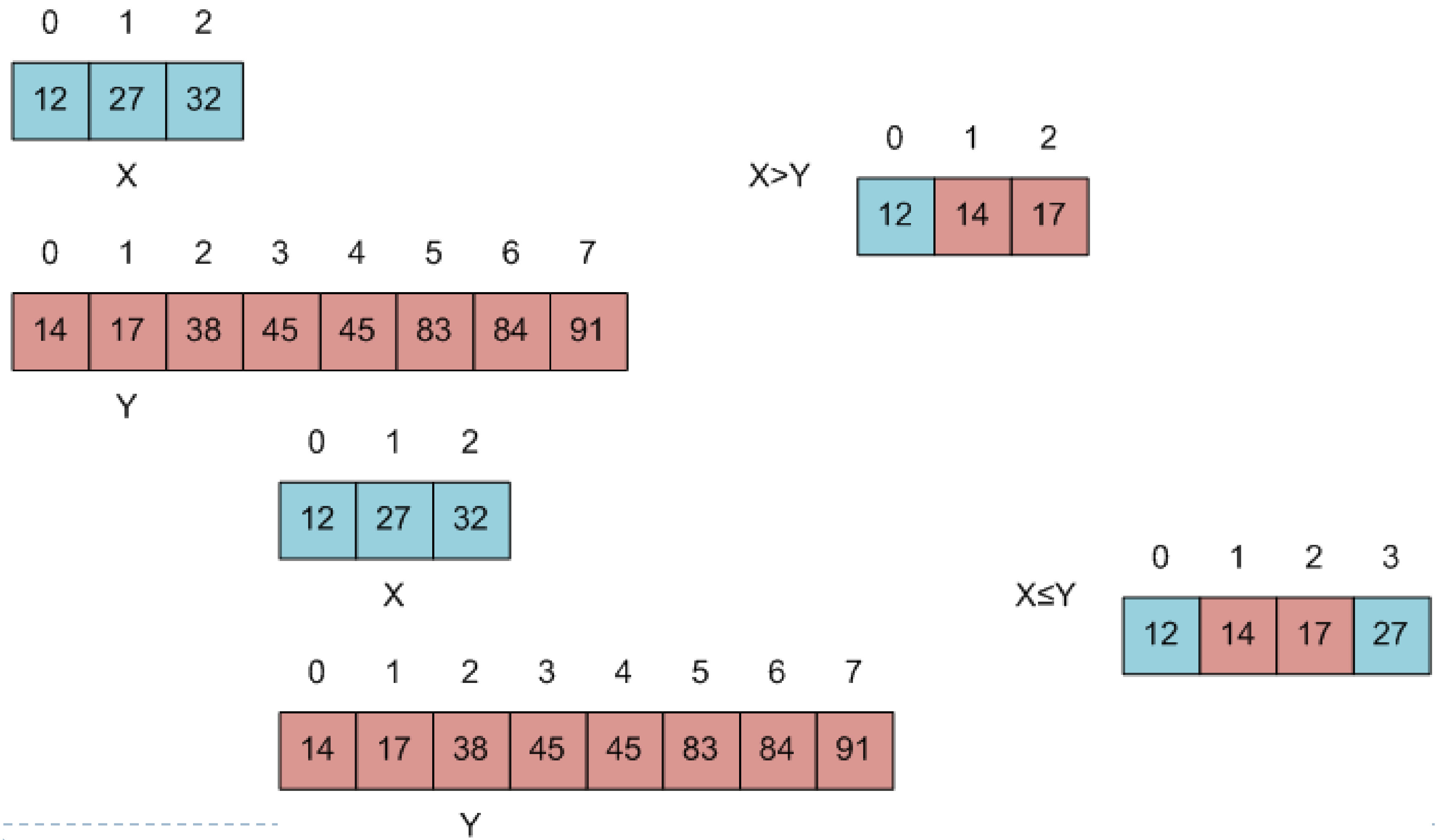
Merge sorting

- Merge phase



Merge sorting

- Merge phase



Merge sorting

- Merge phase

0	1	2
12	27	32

X

0	1	2	3	4	5	6	7
14	17	38	45	45	83	84	91

Y

X≤Y

0	1	2	3	4
12	14	17	27	32

0	1	2
12	27	32

X

The end of the array

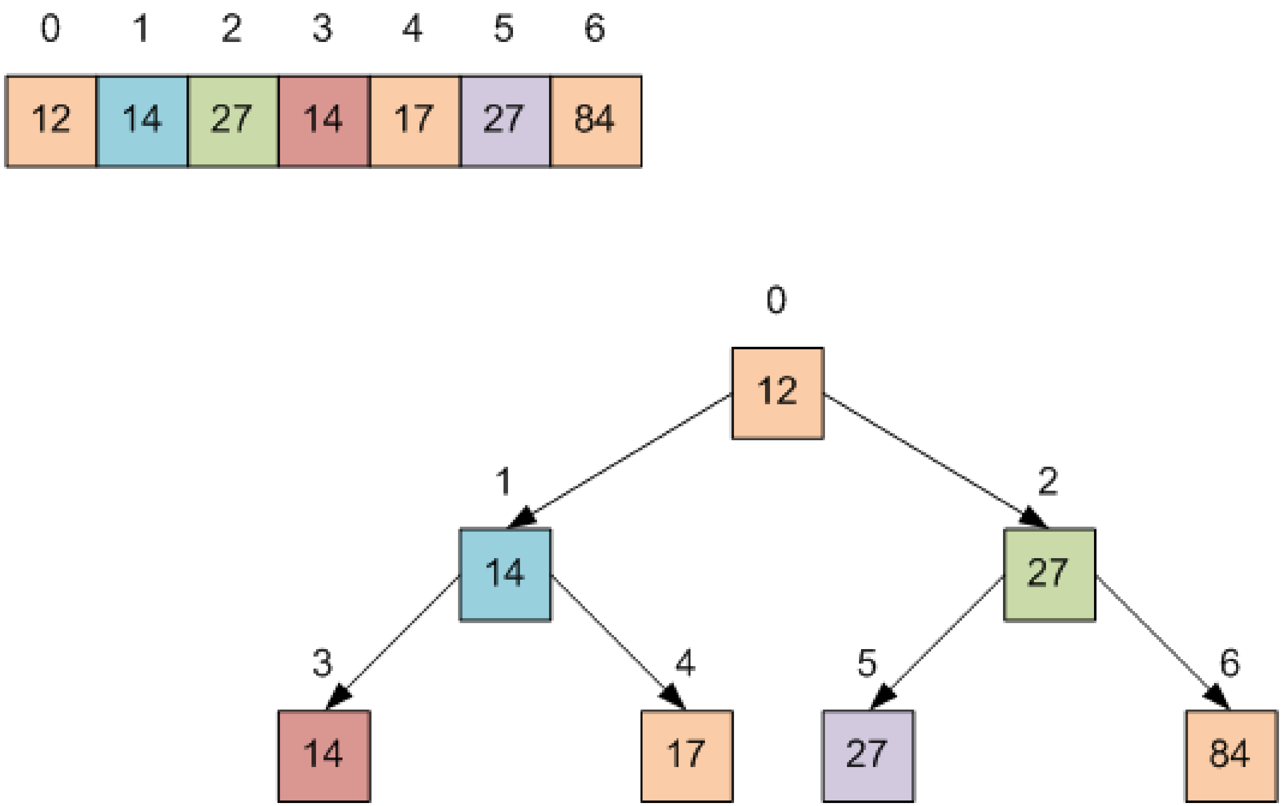
0	1	2	3	4	5	6	7
14	17	38	45	45	83	84	91

Y

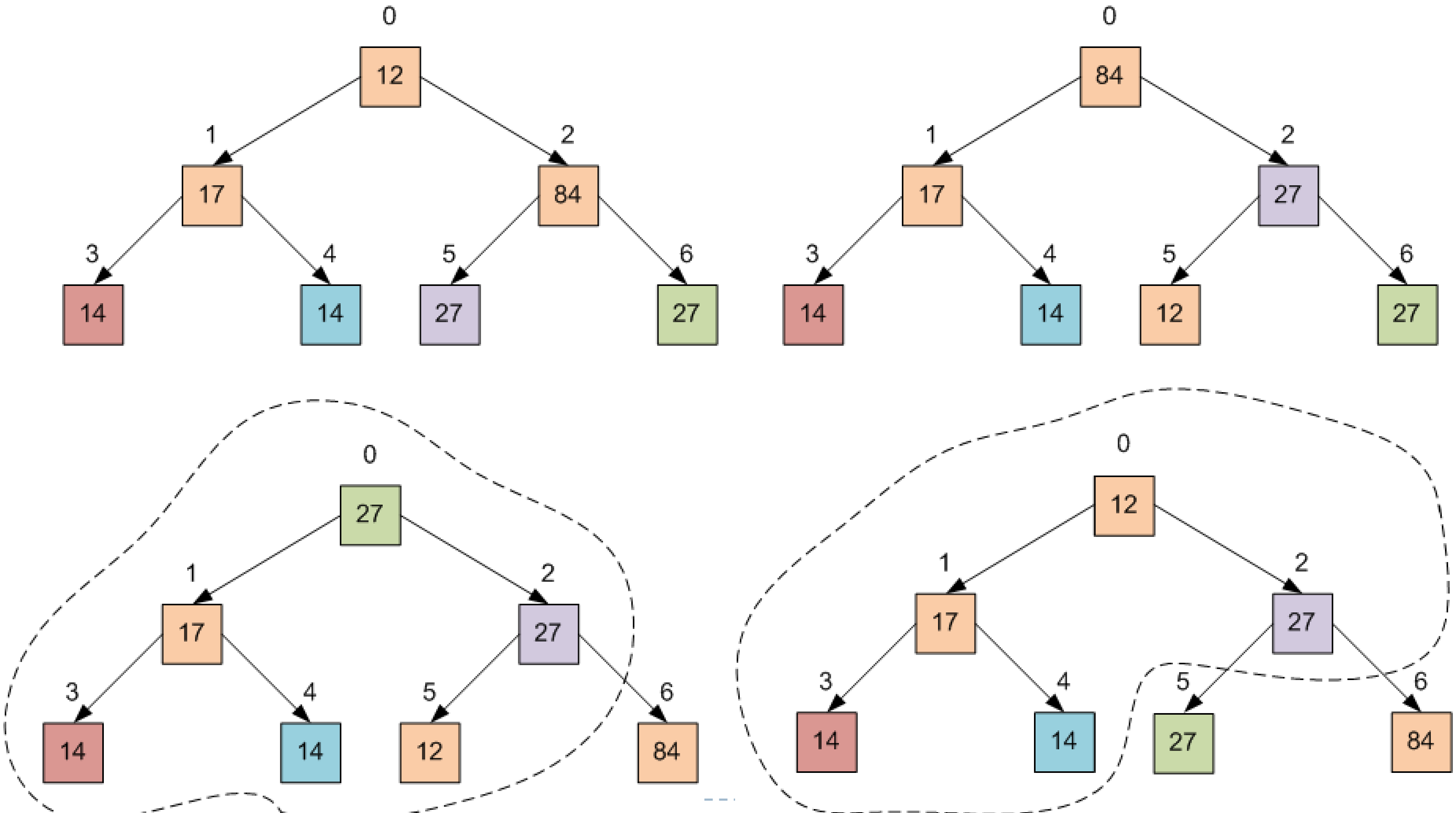
0	1	2	3	4	5	6	7	8	9	10
12	14	17	27	32	38	45	45	83	84	91

Stability

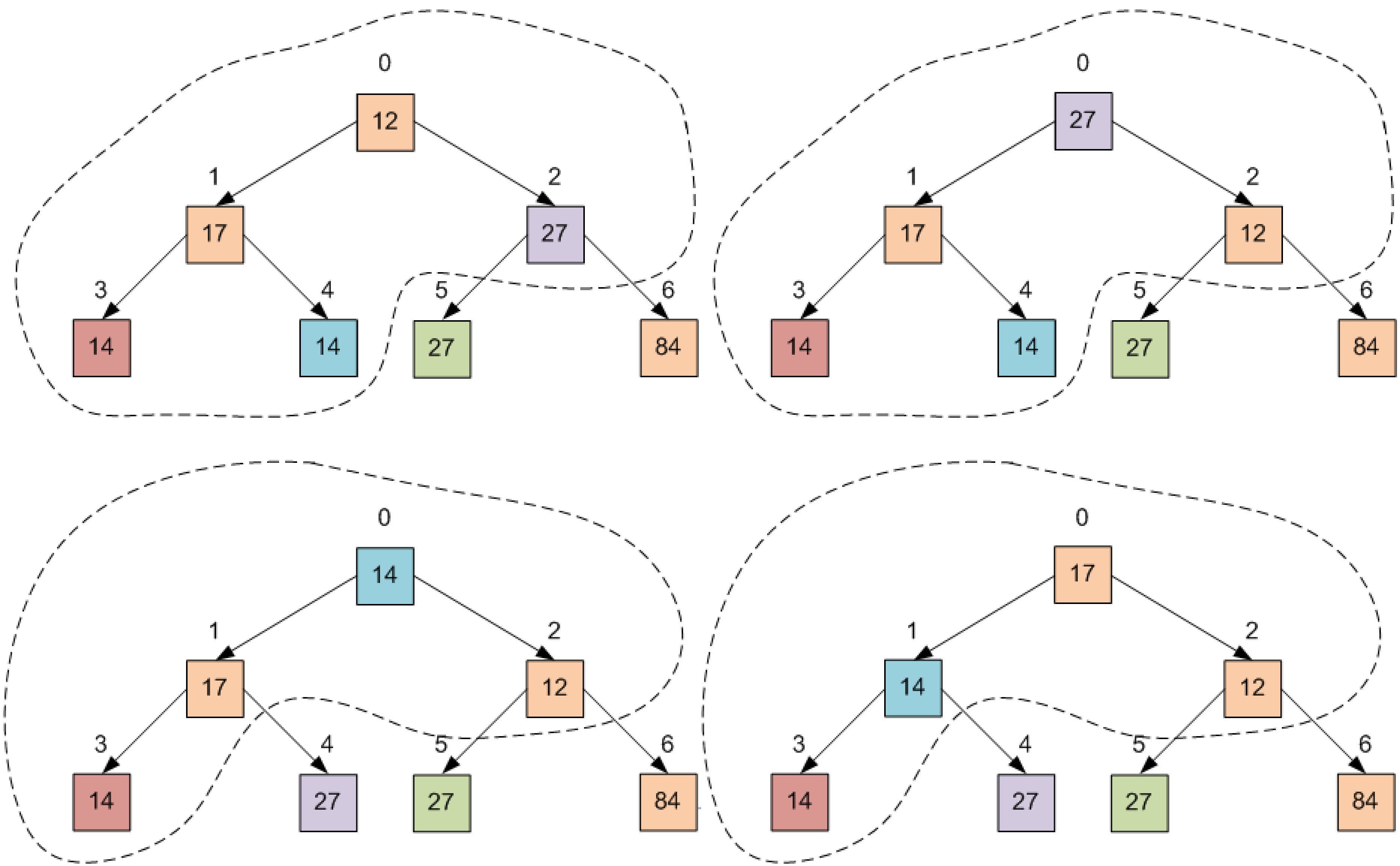
- Equal elements keep their initial order in a sorted array



Stability

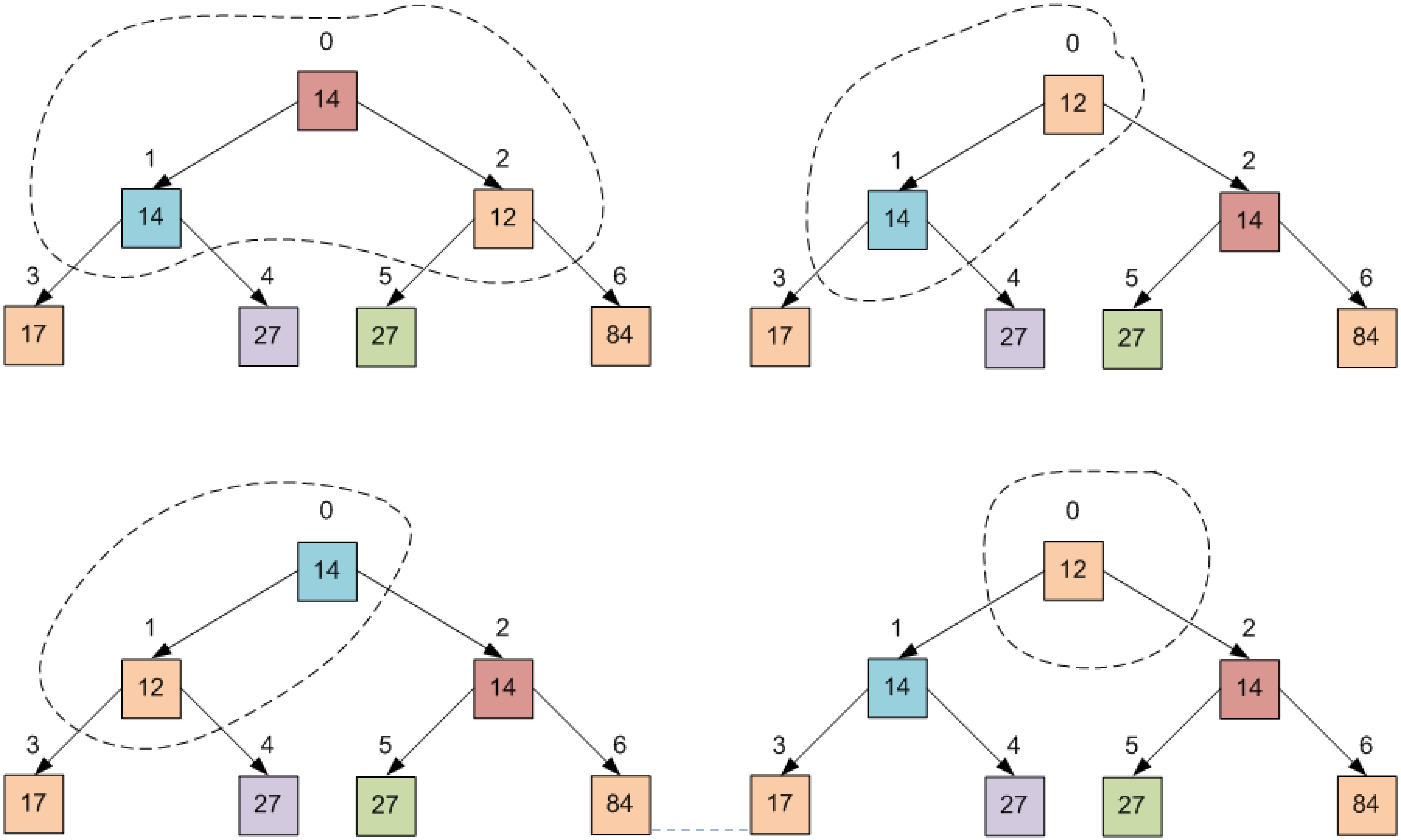

0
12
1
17
2
84
3
14
4
14
5
27
6
27
0
84
1
17
2
27
3
14
4
14
5
12
6
27
0
27
1
17
2
27
3
14
4
14
5
12
6
84
0
12
1
17
2
27
3
14
4
14
5
27
6
84

Stability



0
12
1
17
2
27
3
14
4
14
5
27
6
84
0
27
1
17
2
12
3
14
4
14
5
27
6
84
0
14
1
17
2
12
3
14
4
27
5
27
6
84
0
17
1
14
2
12
3
14
4
27
5
27
6
84

Stability

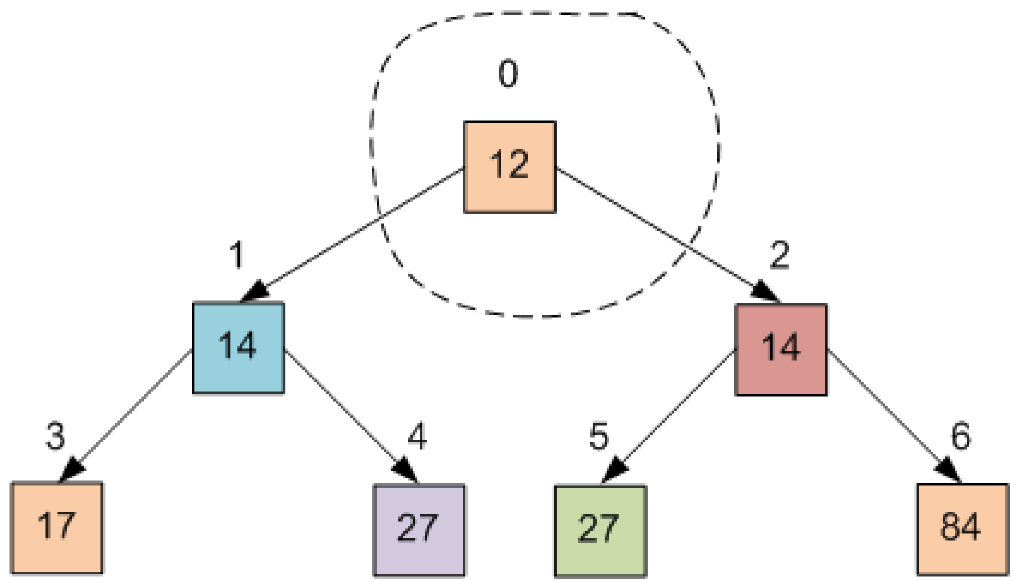


0
14
1
14
2
12
3
4
5
6
17
27
27
84
0
12
1
14
2
14
3
4
5
6
17
27
27
84
0
14
1
12
2
14
3
4
5
6
17
27
27
84
0
12
1
14
2
14
3
4
5
6
17
27
27
84

Stability

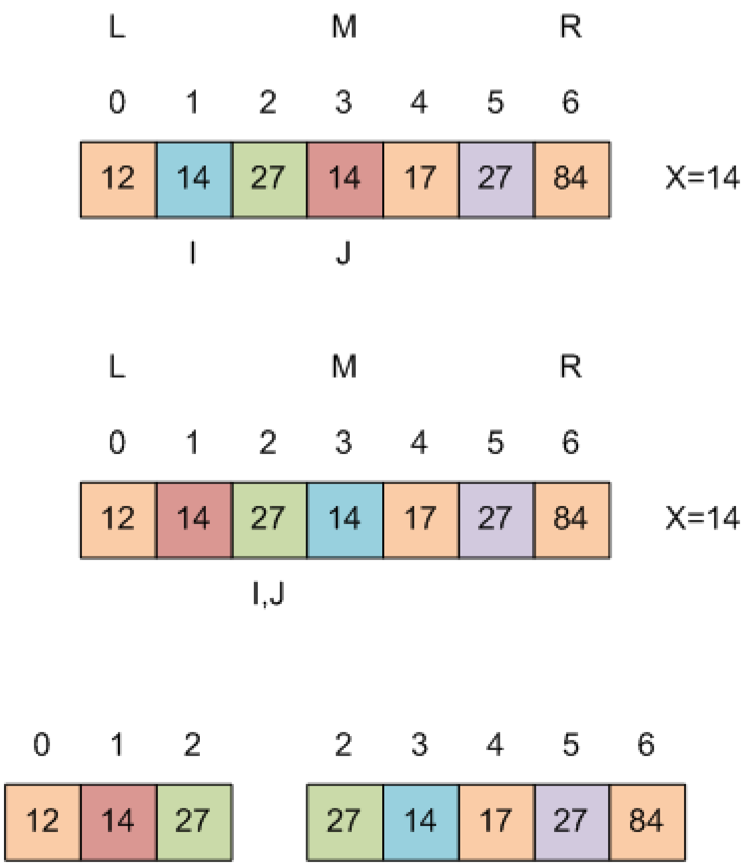
- **Heapsort is unstable**

0	1	2	3	4	5	6
12	14	27	14	17	27	84

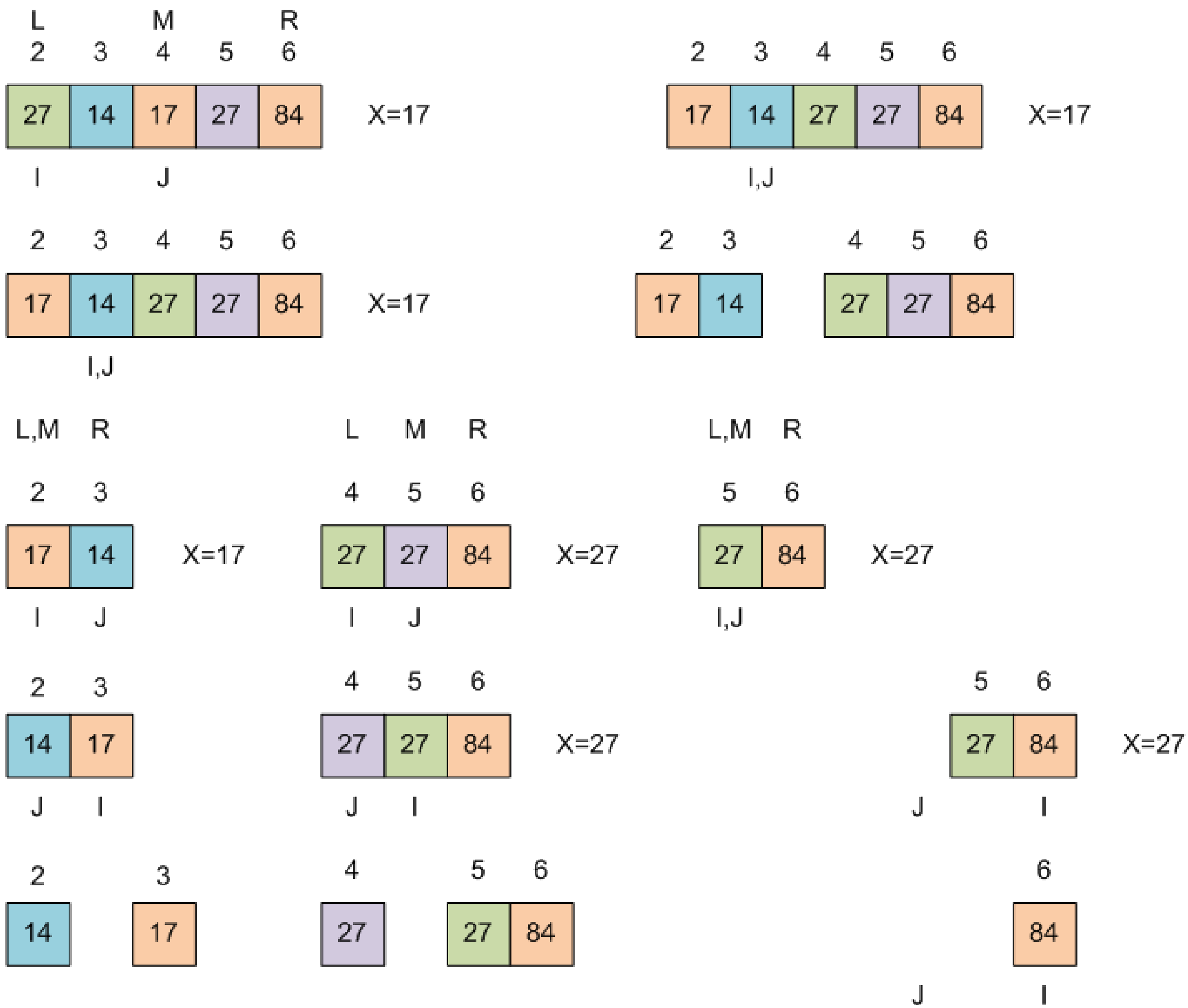


Stability

- **Quicksort**



Stability


L M R
2 3 4 5 6
27 14 17 27 84
X=17
I J
2 3 4 5 6
17 14 27 27 84
X=17
I,J
2 3 4 5 6
17 14 27 27 84
X=17
I,J
2 3
17 14
4 5 6
27 27 84
L,M R
2 3
17 14
X=17
I J
L M R
4 5 6
27 27 84
X=27
I J
L,M R
5 6
27 84
X=27
I,J
2 3
14 17
J I
4 5 6
27 27 84
X=27
J I
5 6
27 84
X=27
J I
2 3
14 17
4 5 6
27 27 84
6
84
J I

Stability

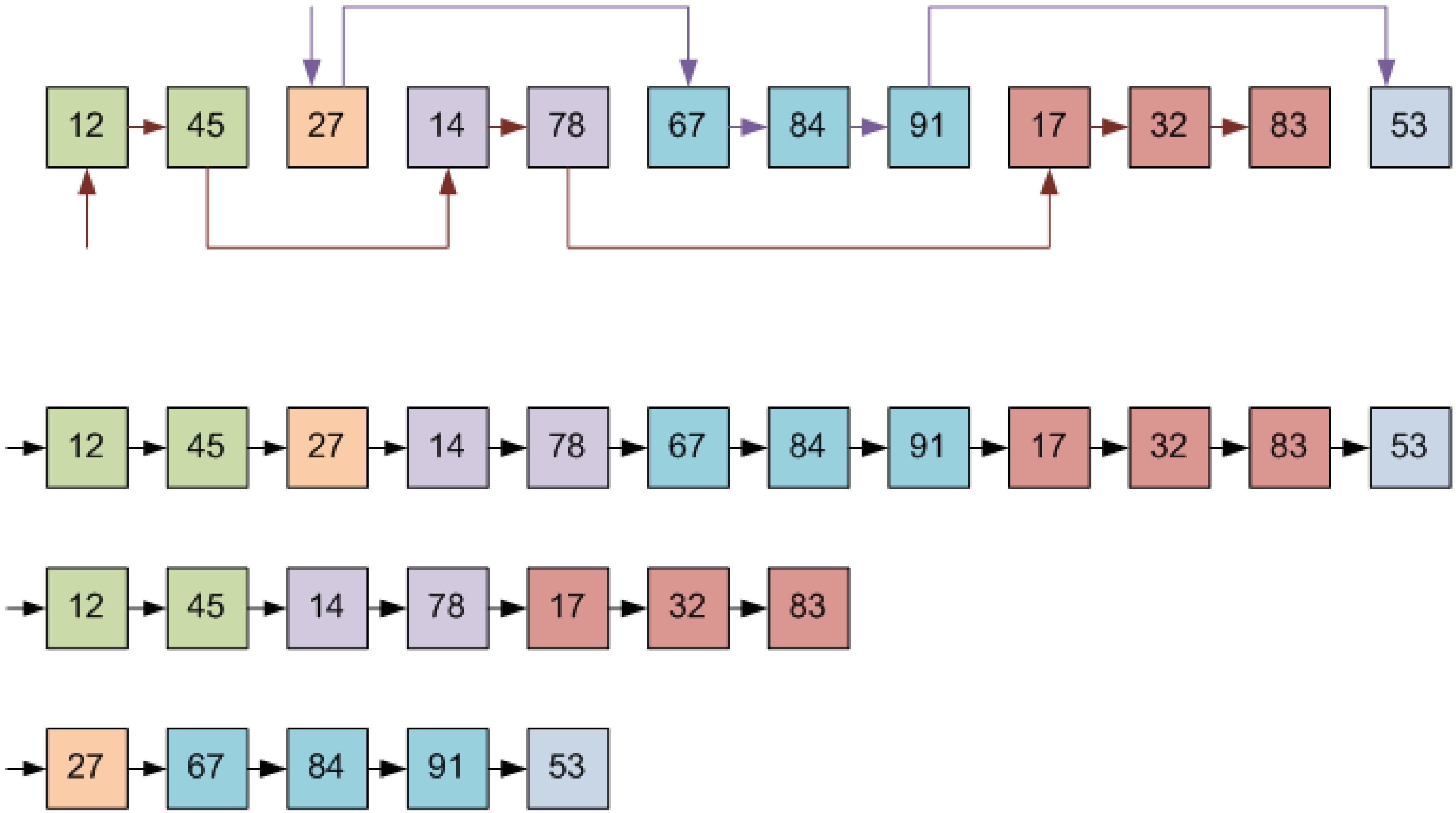
- **Quicksort is unstable**

0	1	2	3	4	5	6
12	14	27	14	17	27	84

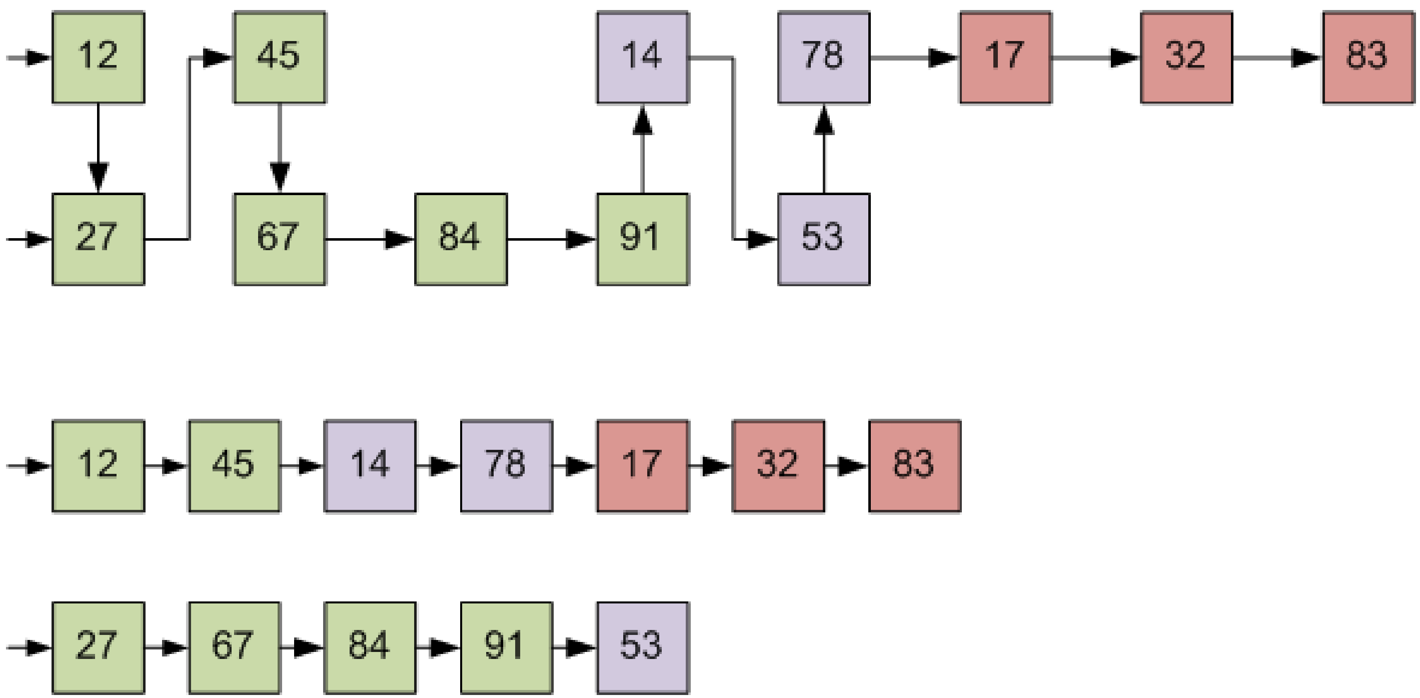
0	1	2	3	4	5	6
12	14	14	17	27	27	84

- **Mergesort is stable**

Merge sorting for linked lists



Merge sorting for linked lists



Comparison

	Worst case	Average case	Data structures	Stability
Shell	N^2		Random access	No
Hepsort	N logN	N logN	Random access	No
Quicksort	N^2 (hardly ever possible)	N logN	Random access	No
Mergesort	N logN	N logN	Random access Sequential access	Yes