Chapter eight

8. Advanced Sorting and Searching Algorithms

8.1. *Shell Sort*

Shell sort is an improvement of insertion sort. It is developed by Donald Shell in 1959. Insertion sort works best when the array elements are sorted in a reasonable order. Thus, shell sort first creates this reasonable order.

Algorithm:

1. Choose gap g_k between elements to be partly ordered.
2. Generate a sequence (called increment sequence) g_k, g_{k-1},…., g_2, g_1 where for each sequence g_i, A[j]<=A[j+g_i] for 0<=j<=n-1-g_i and k>=i>=1

It is advisable to choose g_k =n/2 and g_{i-1} = g_i/2 for k>=i>=1. After each sequence g_{k-1} is done and the list is said to be g_i-sorted. Shell sorting is done when the list is *1-sorted* (which is sorted using insertion sort) and A[j]<=A[j+1] for 0<=j<=n-2. Time complexity is $O(n^{3/2})$.

Example: Sort the following list using shell sort algorithm.

5	8	2	4	1	3	9	7	6	0

Choose g_3 =5 (n/2 where n is the number of elements =10)

Sort (5, 3)	3	8	2	4	1	5	9	7	6	0
Sort (8, 9)	3	8	2	4	1	5	9	7	6	0
Sort (2, 7)	3	8	2	4	1	5	9	7	6	0
Sort (4, 6)	3	8	2	4	1	5	9	7	6	0
Sort (1, 0)	3	8	2	4	0	5	9	7	6	1
➔ 5- sorted list	3	8	2	4	0	5	9	7	6	1

Choose g_2 =3

Sort (3, 4, 9, 1)	1	8	2	3	0	5	4	7	6	9
Sort (8, 0, 7)	1	0	2	3	7	5	4	8	6	9
Sort (2, 5, 6)	1	0	2	3	7	5	4	8	6	9
➔ 3- sorted list	1	0	2	3	7	5	4	8	6	9

Choose g_1 =1 (the same as insertion sort algorithm)

Sort (1, 0, 2, 3, 7, 5, 4, 8, 6, 9)	0	1	2	3	4	5	6	7	8	9
➔ 1- sorted (shell sorted) list	0	1	2	3	4	5	6	7	8	9

7.2. *Quick Sort*

Quick sort is the fastest known algorithm. It uses divide and conquer strategy and in the worst case its complexity is O (n2). But its expected complexity is O(nlogn).

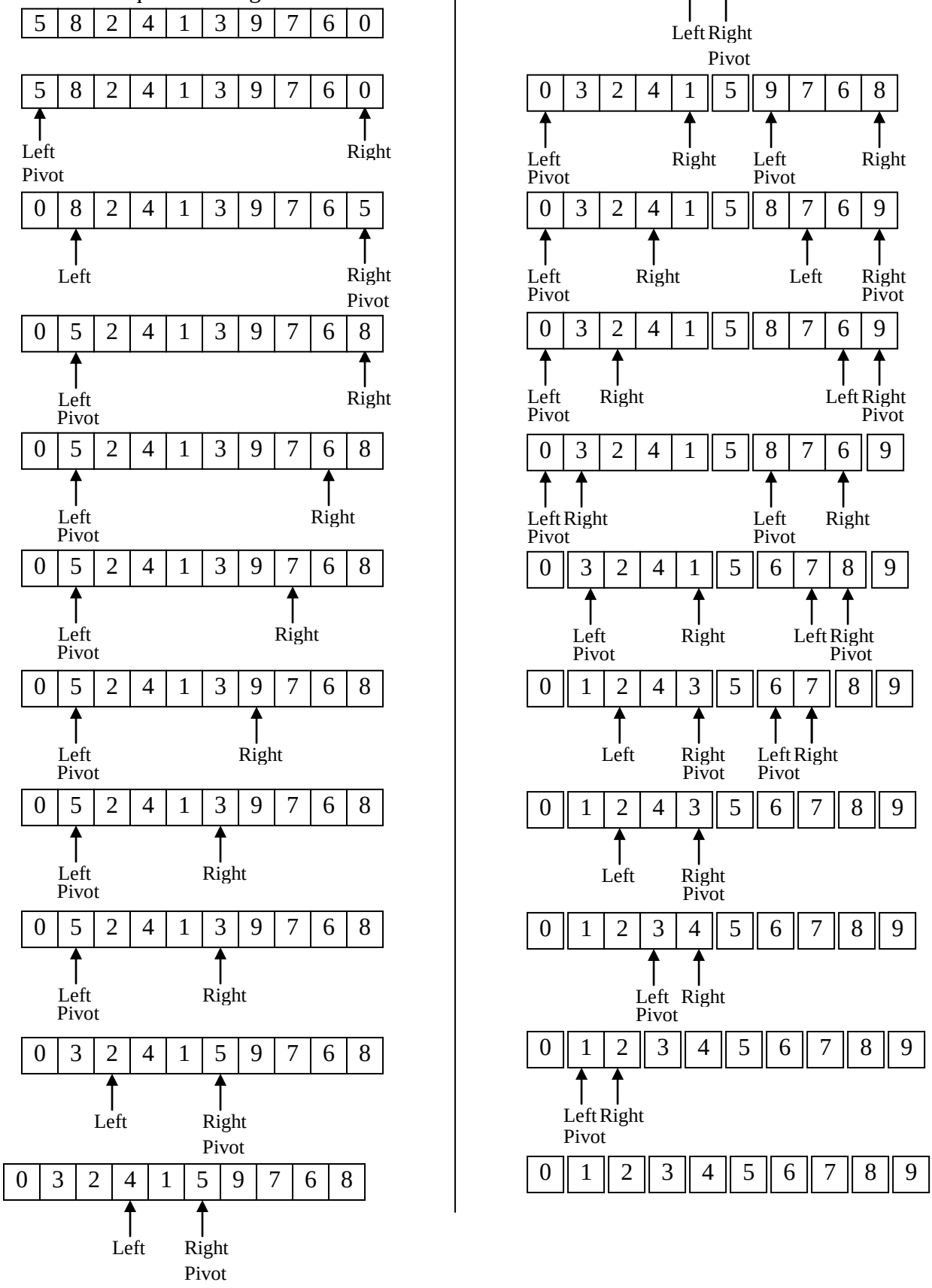
Algorithm:

1. Choose a pivot value (mostly the first element is taken as the pivot value)
2. Position the pivot element and partition the list so that:
   - the left part has items less than or equal to the pivot value
   - the right part has items greater than or equal to the pivot value
3. Recursively sort the left part
4. Recursively sort the right part

The following algorithm can be used to position a pivot value and create partition.

```
Left=0;
Right=n-1; // n is the total number of elements in the list
PivotPos=Left;
while(Left<Right)
{
        if(PivotPos==Left)
        {
                if(Data[Left]>Data[Right])
                {
                        swap(data[Left], Data[Right]);
                        PivotPos=Right;
                        Left++;
                }
                else
                        Right--;
        }
        else
        {
                if(Data[Left]>Data[Right])
                {
                        swap(data[Left], Data[Right]);
                        PivotPos=Left;
                        Right--;
                }
                else
                        Left++;
        }
}
```

Example: Sort the following list using quick sort algorithm.



8.3. *Heap Sort*

Heap sort operates by first converting the list in to a heap tree. Heap tree is a binary tree in which each node has a value greater than both its children (if any). It uses a process called "adjust to accomplish its task (building a heap tree) whenever a value is larger than its parent. The time complexity of heap sort is O(nlogn).

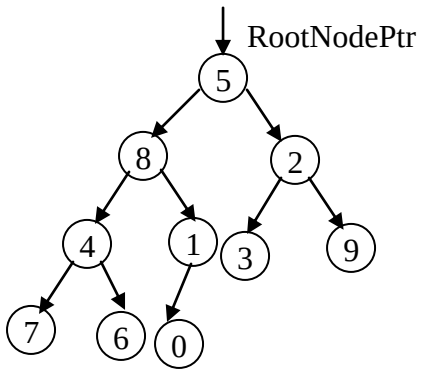
Algorithm:

1. Construct a binary tree
   - The root node corresponds to Data[0].
   - If we consider the index associated with a particular node to be *i*, then the left child of this node corresponds to the element with index 2**i*+1 and the right child corresponds to the element with index 2**i*+2. If any or both of these elements do not exist in the array, then the corresponding child node does not exist either.
2. Construct the heap tree from initial binary tree using "adjust" process.
3. Sort by swapping the root value with the lowest, right most value and deleting the lowest, right most value and inserting the deleted value in the array in it proper position.

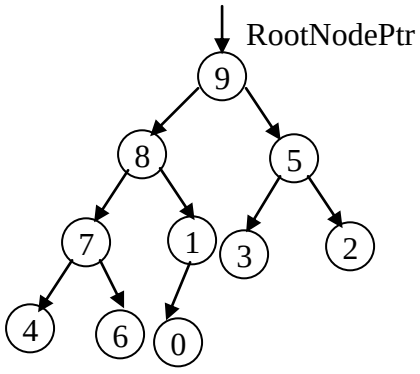
Example: Sort the following list using heap sort algorithm.

5	8	2	4	1	3	9	7	6	0

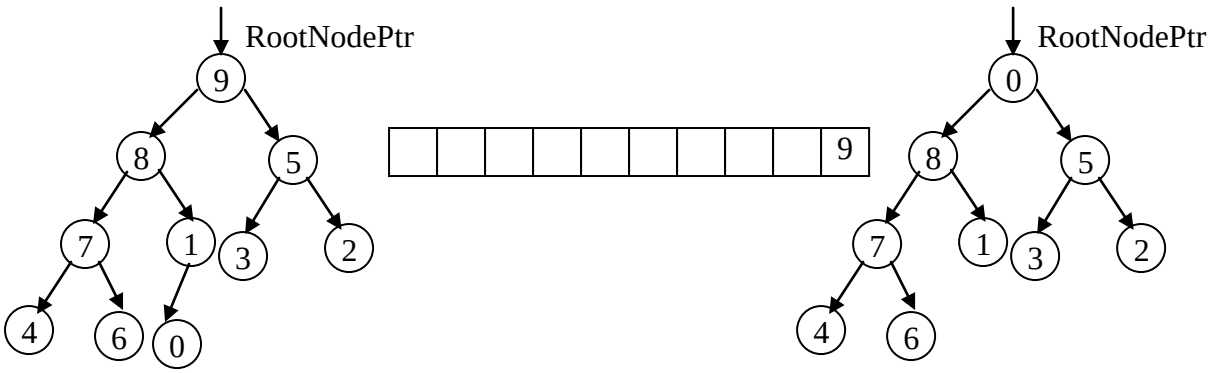
Construct the initial binary tree

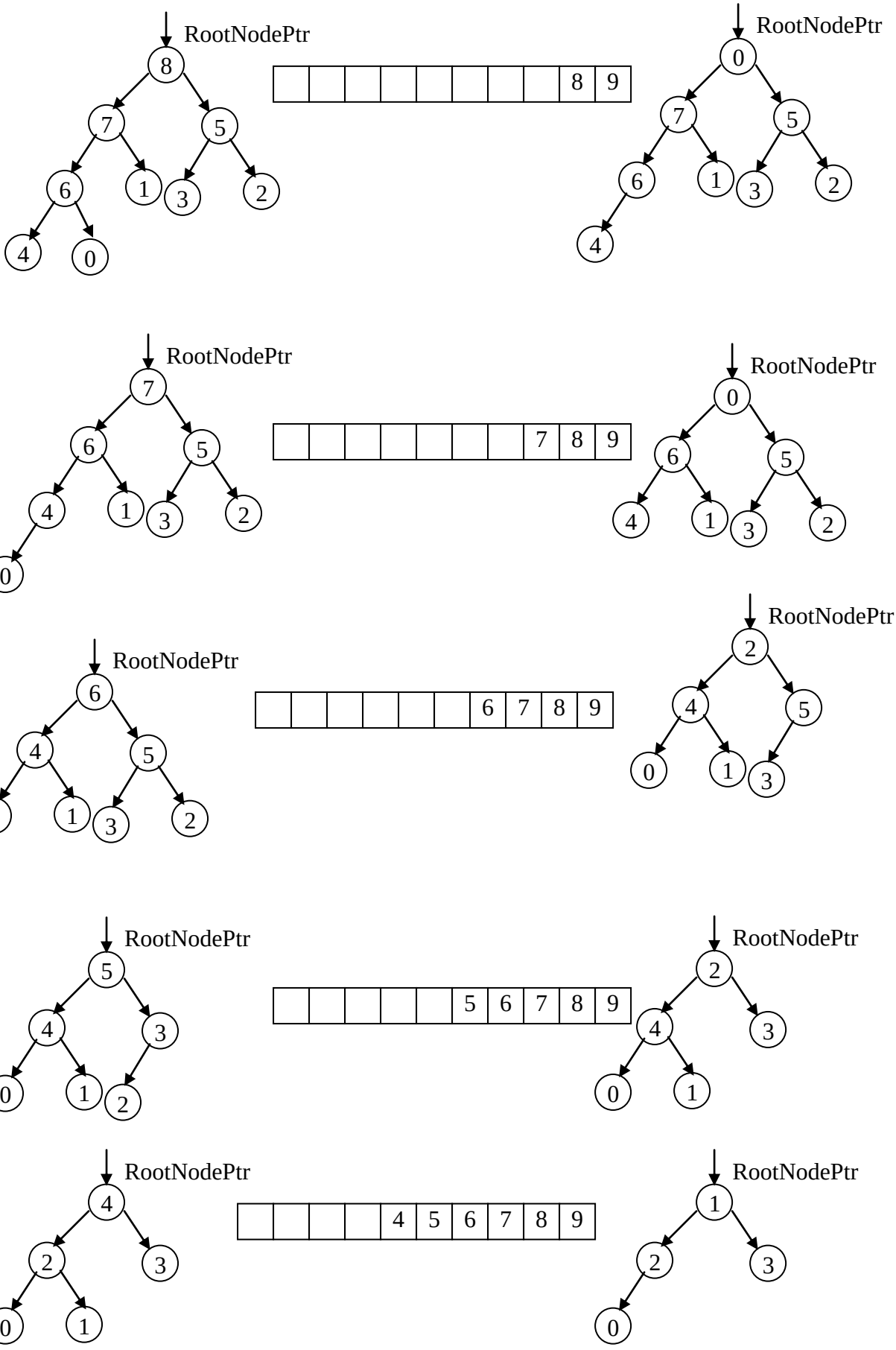


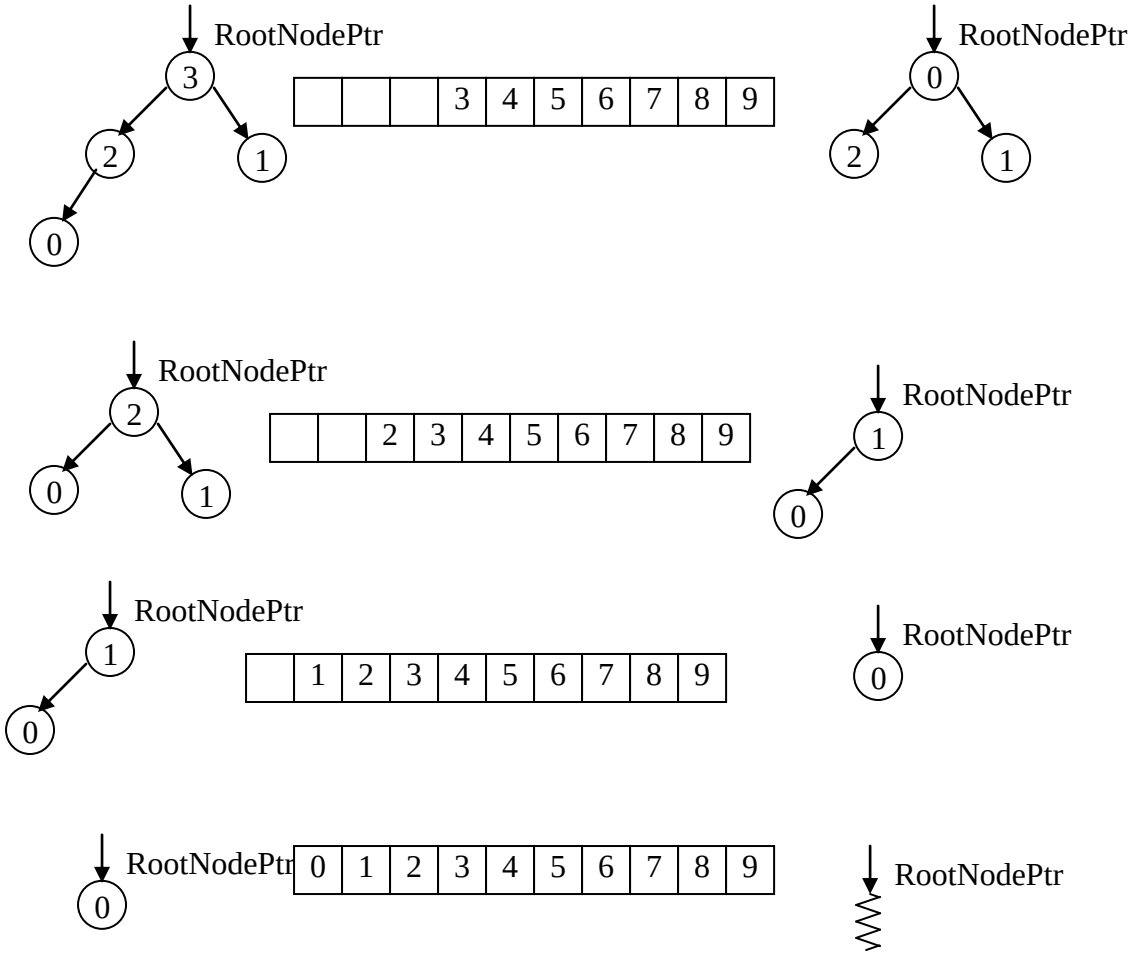
Construct the heap tree



Swap the root node with the lowest, right most node and delete the lowest, right most value; insert the deleted value in the array in its proper position; adjust the heap tree; and repeat this process until the tree is empty.




RootNodePtr
8
7
5
6
1
3
2
4
0
8
9
RootNodePtr
0
7
5
6
1
3
2
4
RootNodePtr
7
6
5
4
1
3
2
0
7
8
9
RootNodePtr
0
6
5
4
1
3
2
RootNodePtr
6
4
5
0
1
3
2
6
7
8
9
RootNodePtr
2
4
5
0
1
3
RootNodePtr
5
4
3
0
1
2
5
6
7
8
9
RootNodePtr
2
4
3
0
1
RootNodePtr
4
2
3
0
1
4
5
6
7
8
9
RootNodePtr
1
2
3
0


RootNodePtr
3
2
1
0
3 4 5 6 7 8 9
RootNodePtr
0
2
1
RootNodePtr
2
0
1
2 3 4 5 6 7 8 9
RootNodePtr
1
0
RootNodePtr
1
0
1 2 3 4 5 6 7 8 9
RootNodePtr
0
RootNodePtr
0
0 1 2 3 4 5 6 7 8 9
RootNodePtr

8.4. *Merge Sort*

Like quick sort, merge sort uses divide and conquer strategy and its time complexity is O(nlogn).

Algorithm:

1. Divide the array in to two halves.
2. Recursively sort the first n/2 items.
3. Recursively sort the last n/2 items.
4. Merge sorted items (using an auxiliary array).

Example: Sort the following list using merge sort algorithm.