Heap Sort & Radix Sort

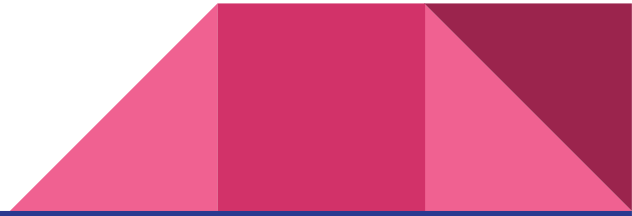
By Sharishma Rongala

HEAP SORT

What is a Heap?

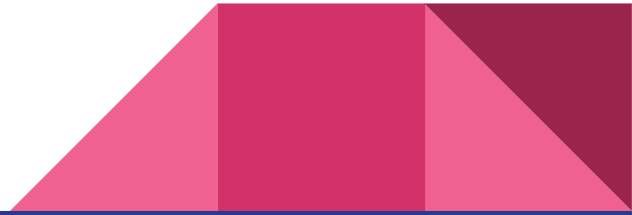
A Heap is an ordered binary tree. It has the following properties:

1) Complete Binary Tree
2) Heap Order



Complete Binary Tree

- Each node can have only two children.
- Except for the lowest level, the tree is completely filled.
- The tree is always left-justified.



Heap Order

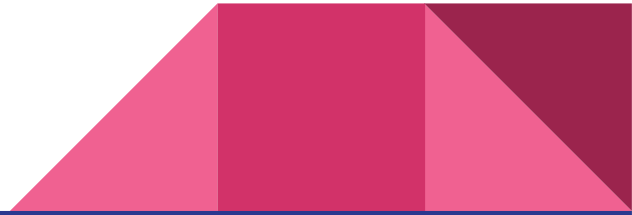
For every node *v*, other than the root, the key stored in *v* follows a particular order.

Max Heap:

- The key follows the order of A[Parent(i)] ≥ A[i]
- To sort the elements in the increasing order, use a max heap

Min Heap:

- The key follows the order of A[Parent(i)] ≤ A[i]
- To sort the elements in the decreasing order, use a min heap

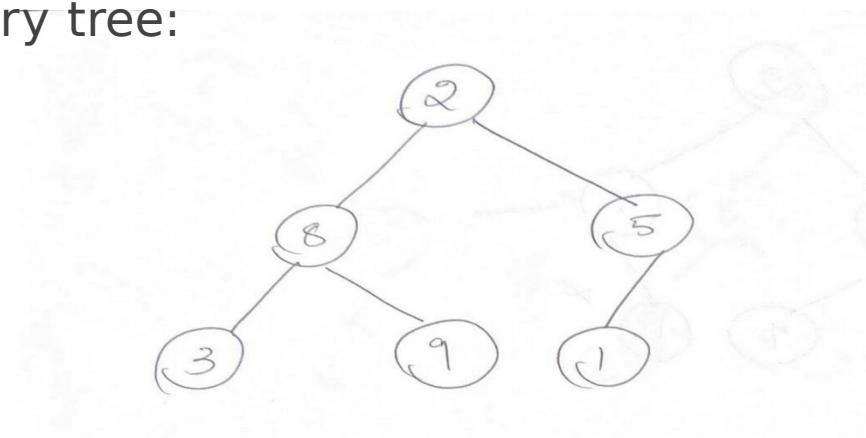


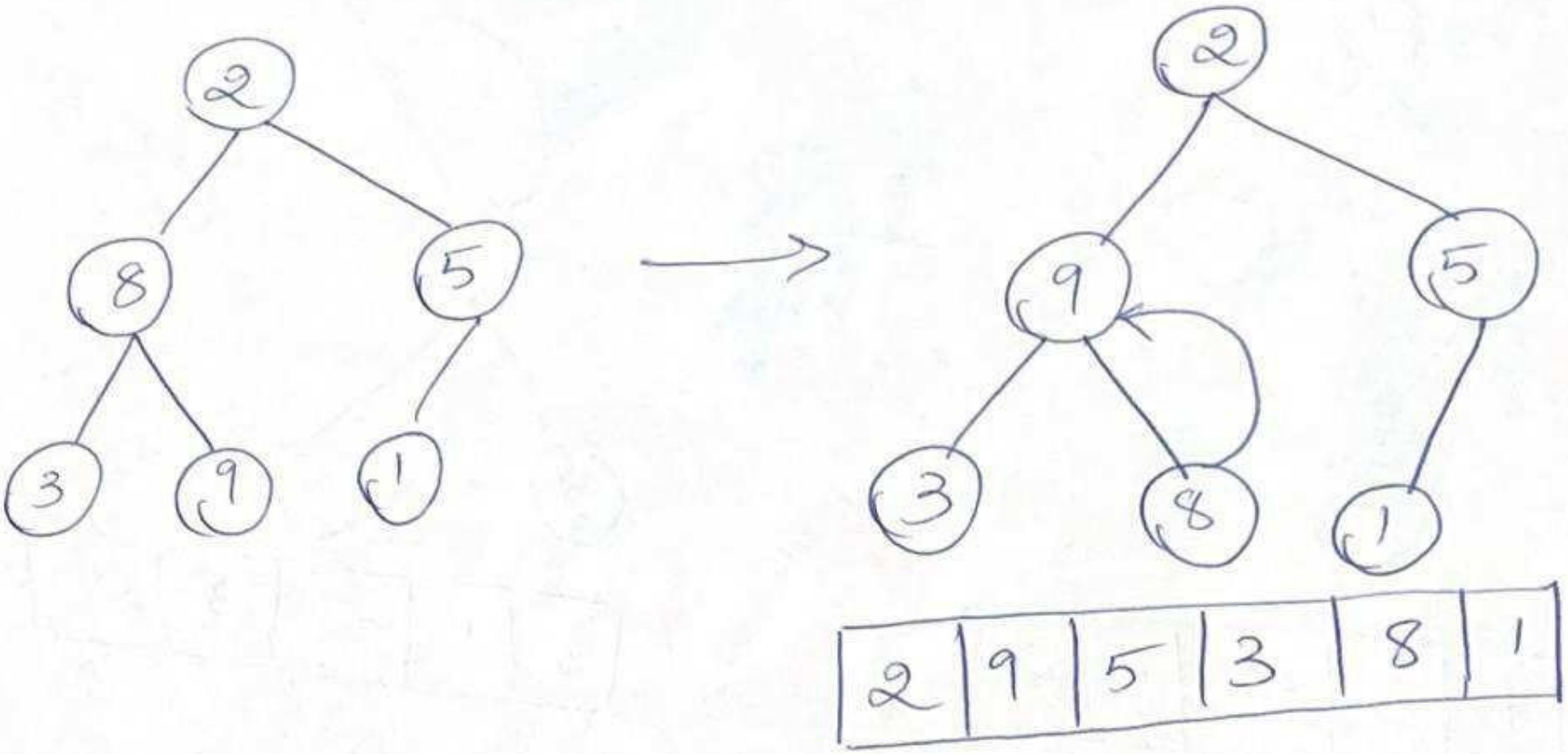
Exampl
e:

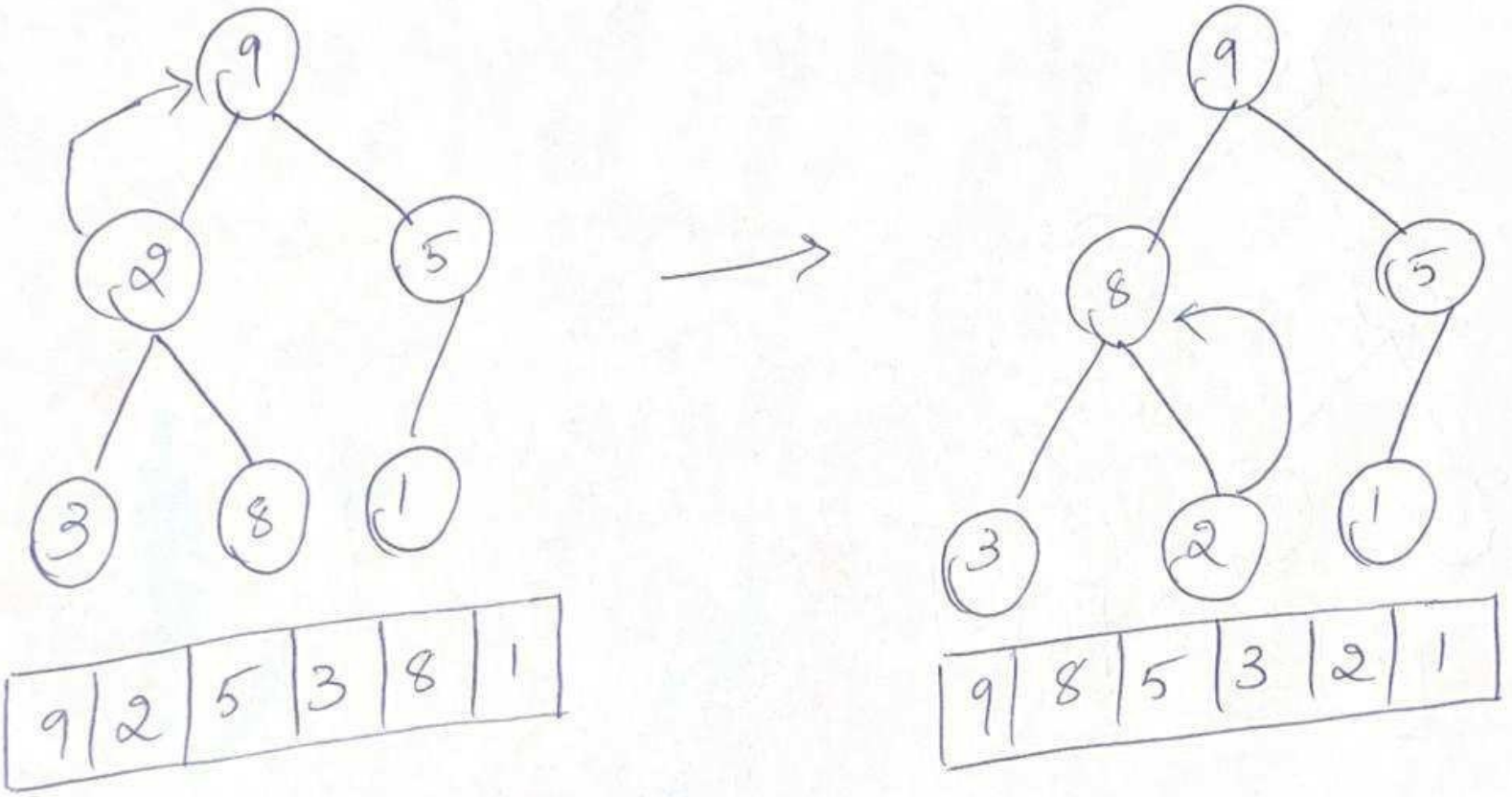
Let’s convert the following array to a heap

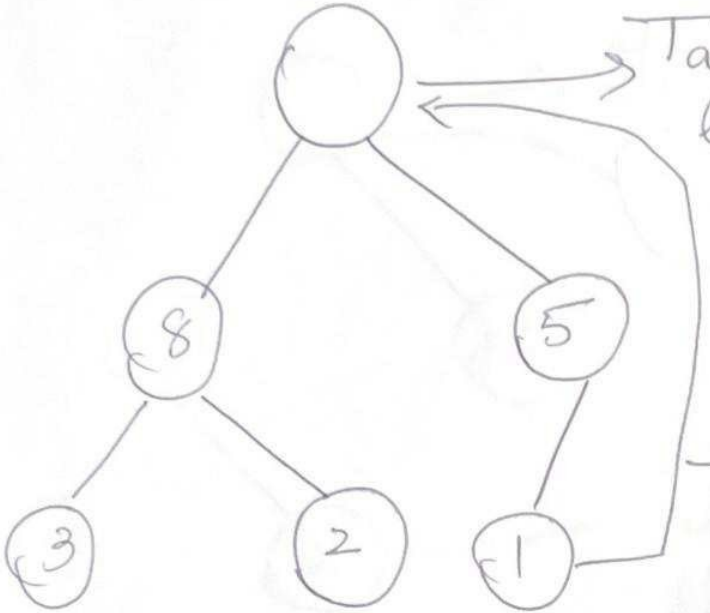
2	8	5	3	9	1

Now, we arrange the array into a complete binary tree:




2
8
5
3
9
1
2
9
5
3
8
1
2 | 9 | 5 | 3 | 8 | 1


9
2
5
3
8
1
9 | 2 | 5 | 3 | 8 | 1
9
8
5
3
2
1
9 | 8 | 5 | 3 | 2 | 1



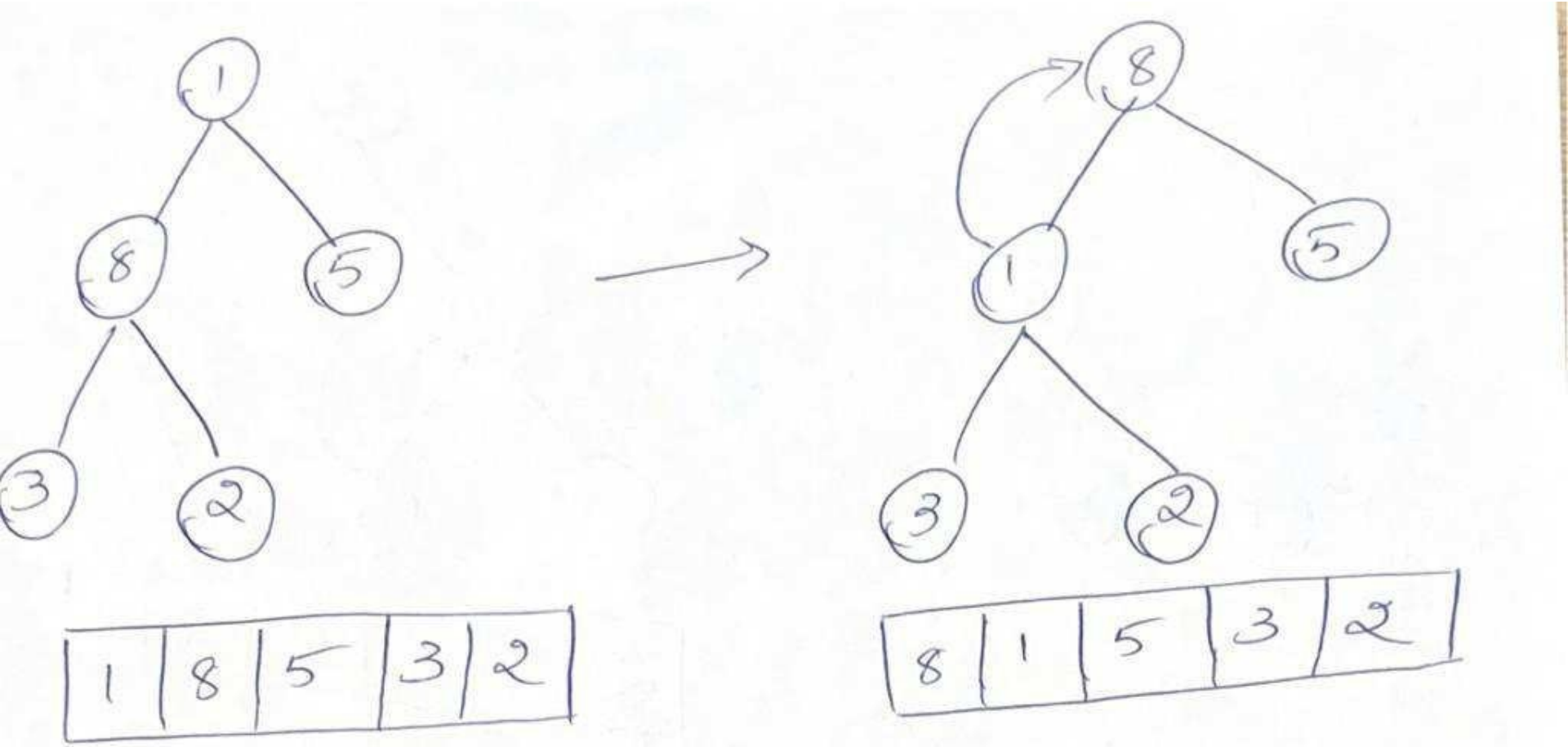
Take out the largest element.

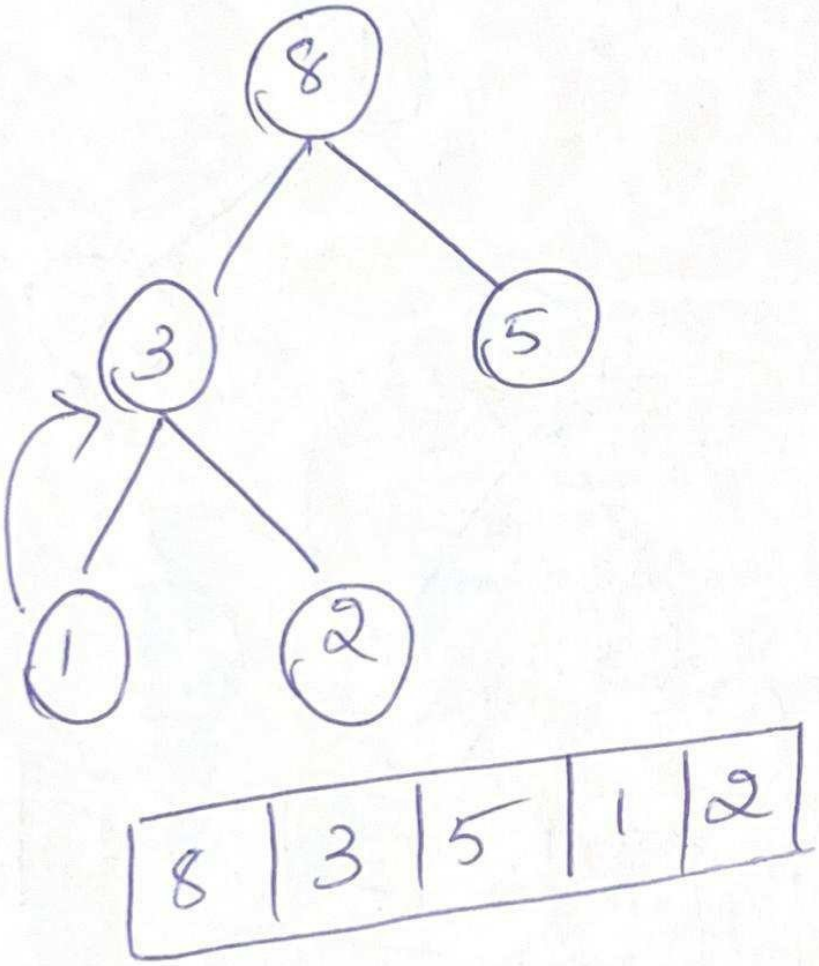
Now, the last element becomes the root.

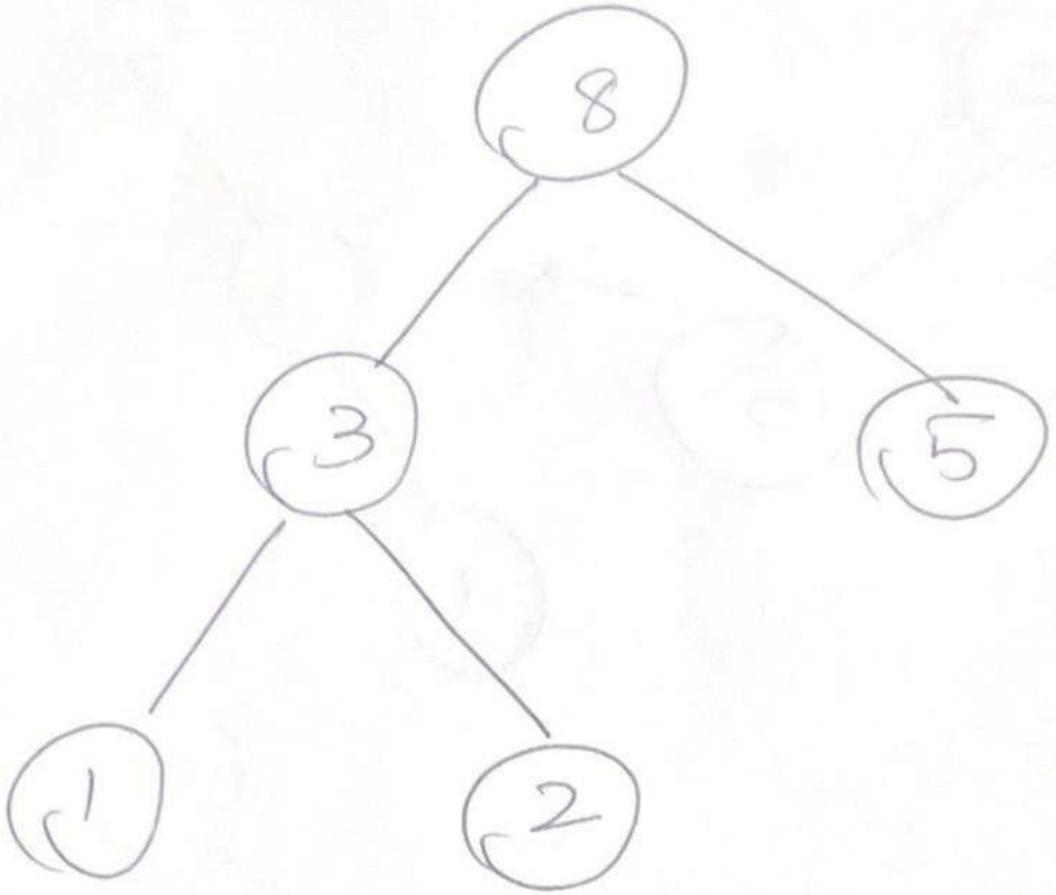
array:

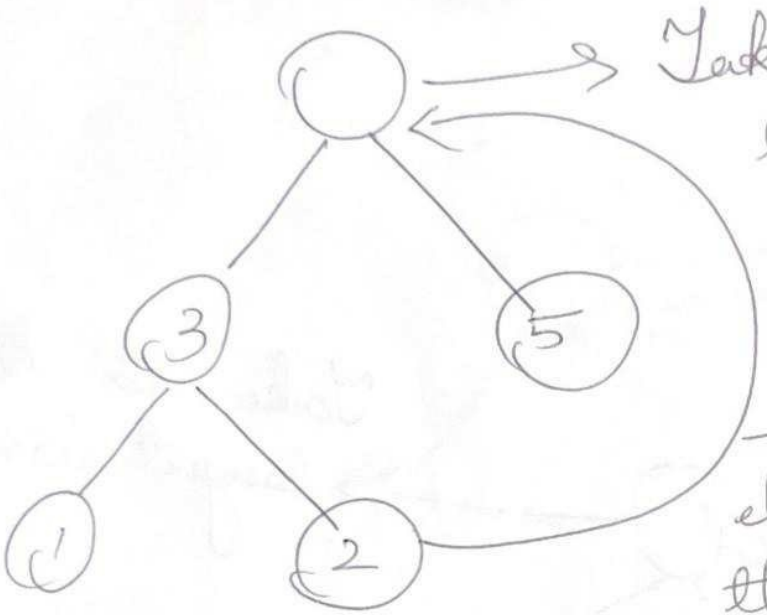
8	5	3	2	1

9


1
8
5
3
2
1 | 8 | 5 | 3 | 2
8
1
5
3
2
8 | 1 | 5 | 3 | 2


8
3
5
1
2
8
3
5
1
2


8
3
5
1
2

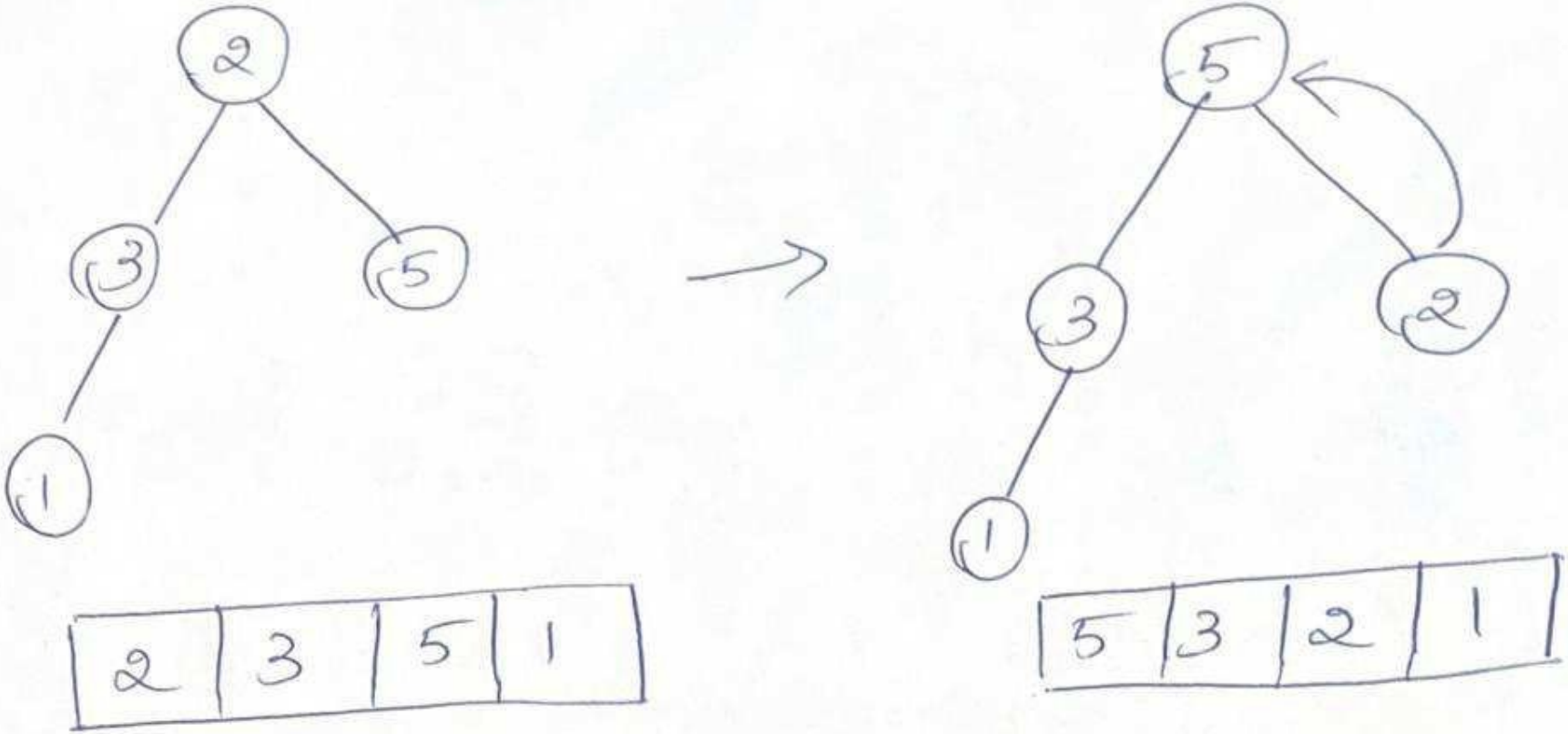


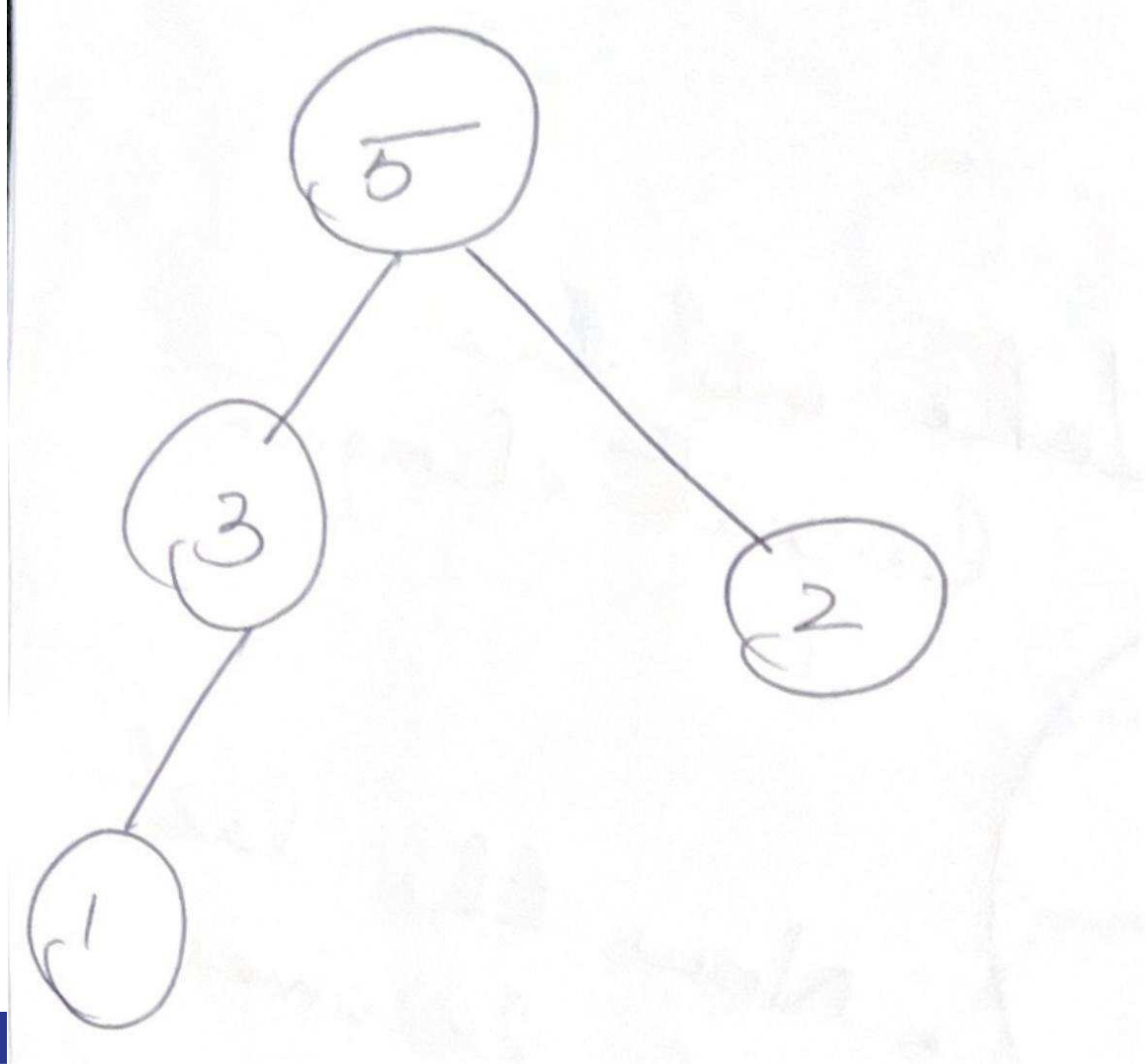
Take out the largest element

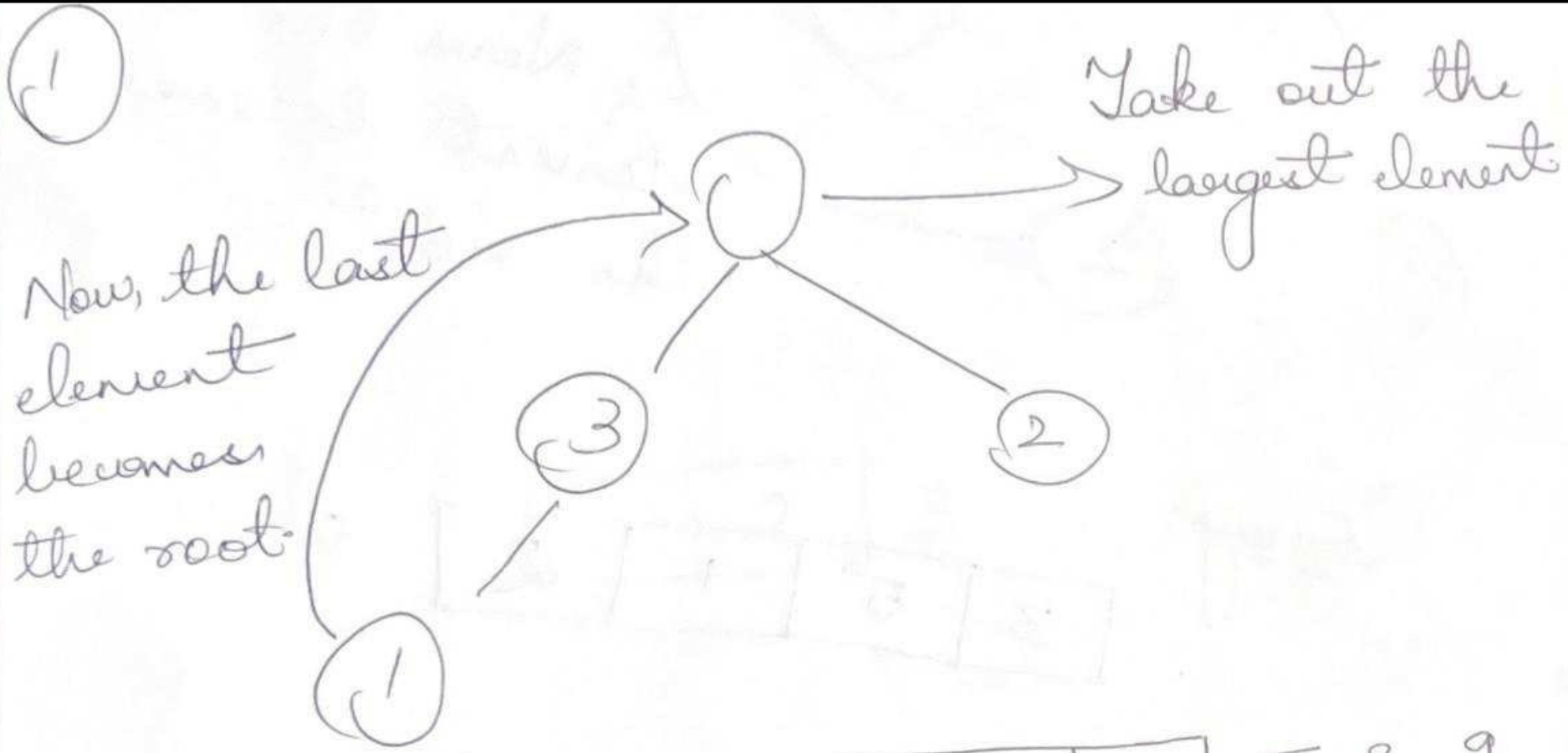
Now, the last element becomes the root.

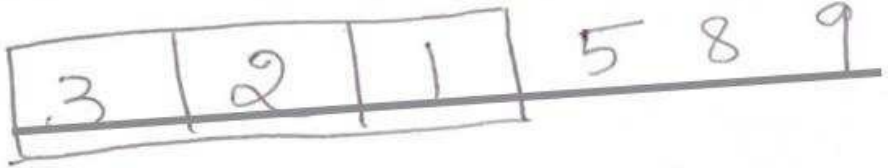
Array:

3	5	1	2	8	9

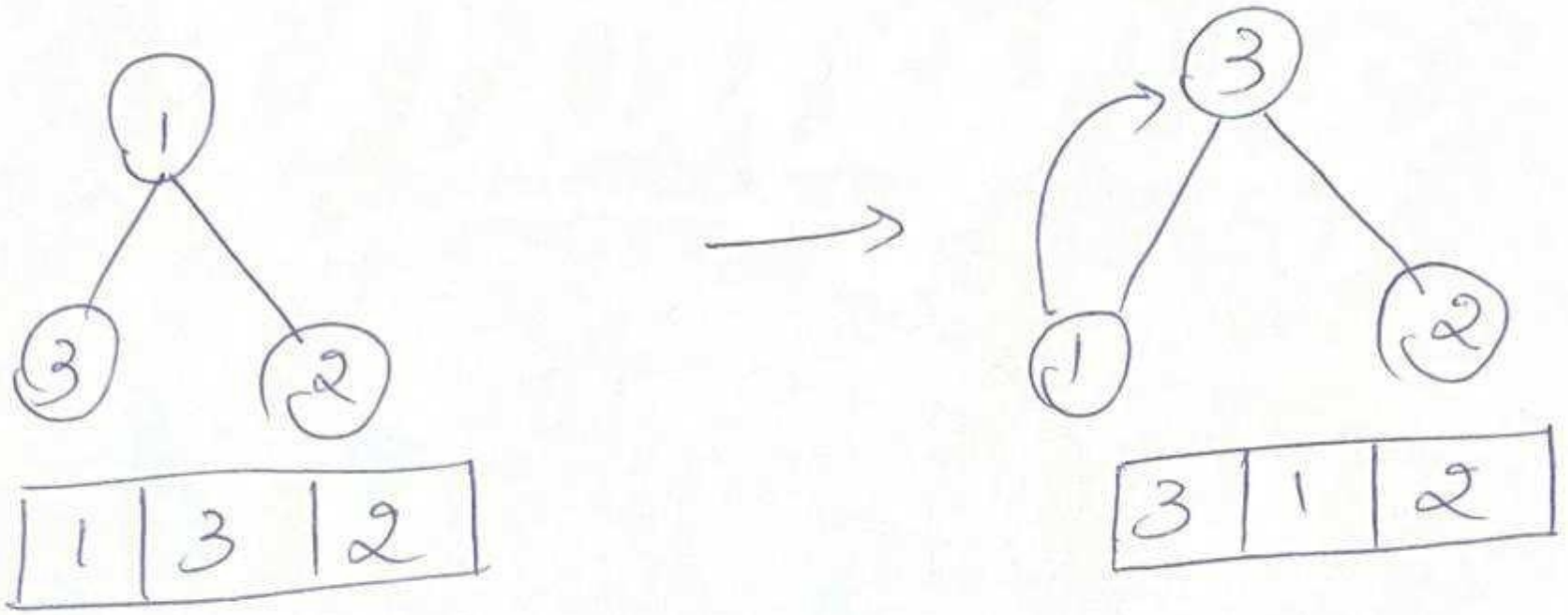

2
3
5
1
2 3 5 1
5
3
2
1
5 3 2 1

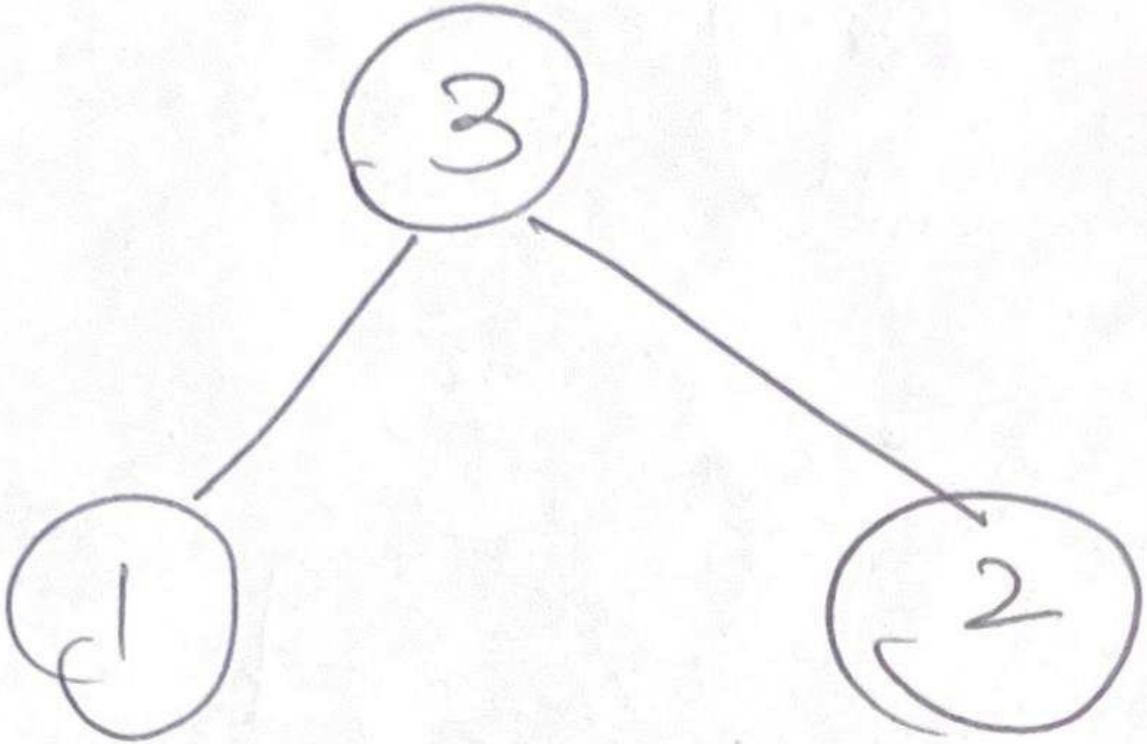

5
3
2
1

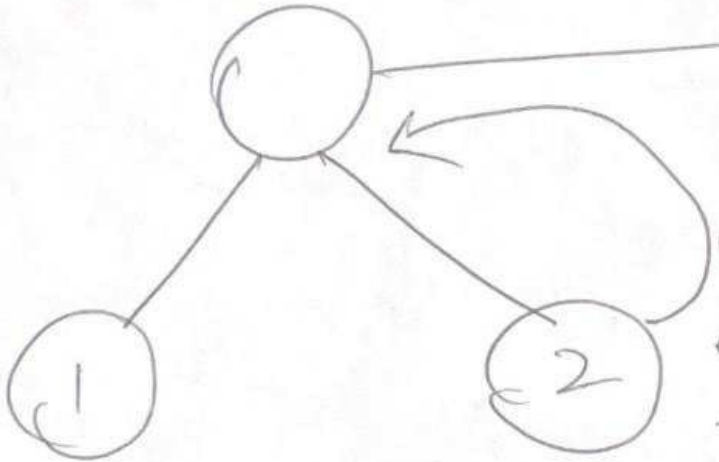

1
Take out the largest element
Now, the last element becomes the root
3
2
1


3
2
1
5
8
9

deray:

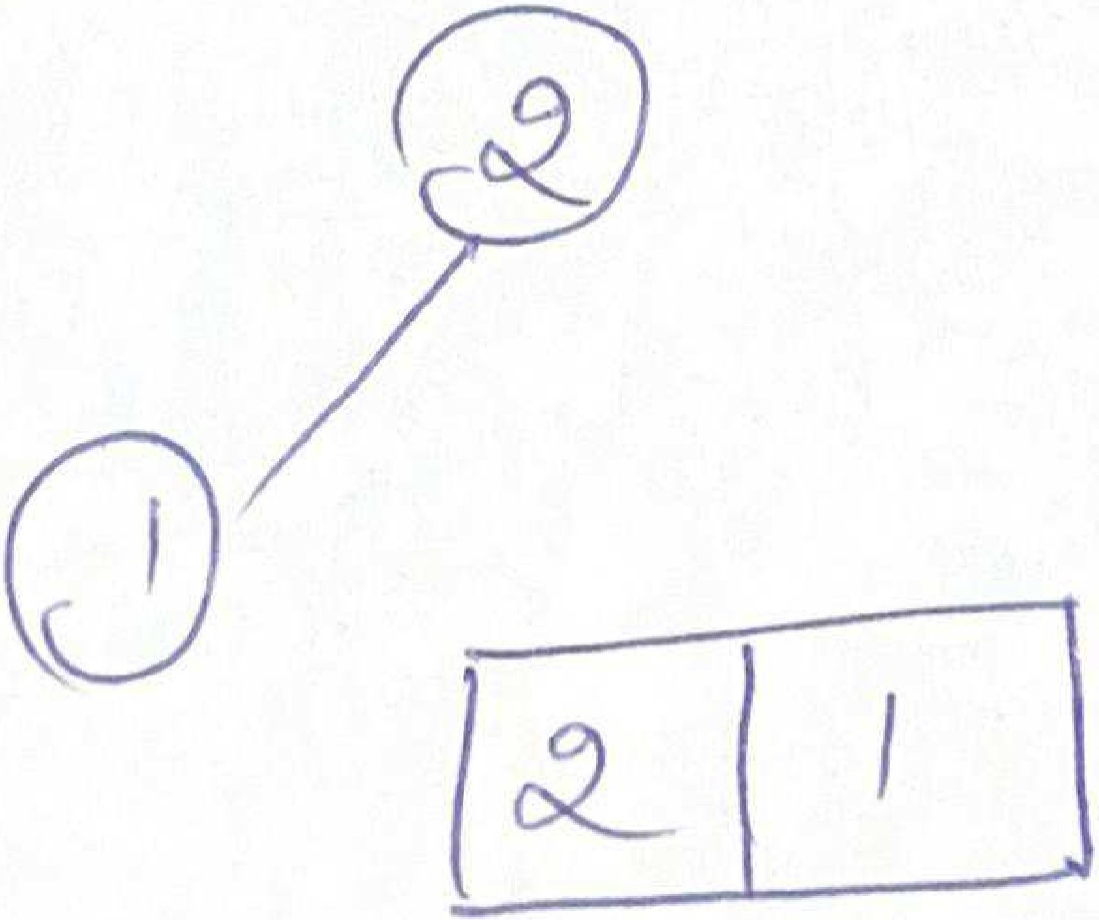

1
3
2
1 | 3 | 2
3
1
2
3 | 1 | 2


3
1
2

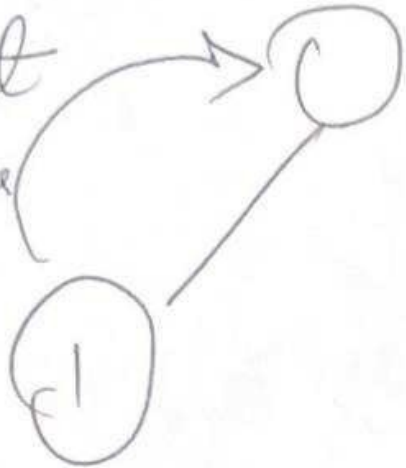


Array:

1	2	3	5	8	9


2
1
2
1

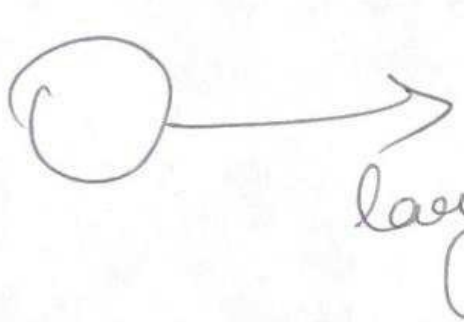
Now, the last element becomes the root



Take out the largest element

Array:

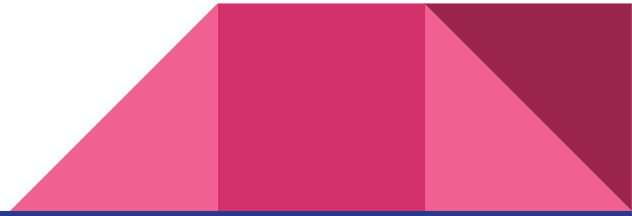
1	2	3	5	8	9



Array: 1 2 3 5 8 9

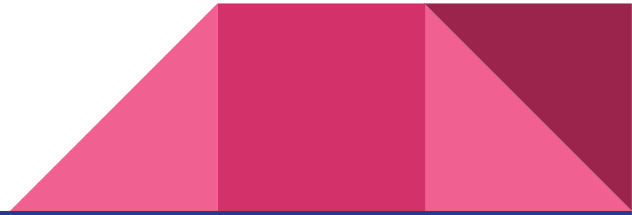
The sorted array is:

1	2	3	5	8	9



ALGORITHM

- **Step 1 -** Create a **Complete Binary Tree** from the elements in a given array.
- **Step 2 -** Convert the Binary Tree to a **Max Heap.**
- **Step 3 - Swap** the root element and the last element from Max Heap.
- **Step 4 -** Repeat this process until Max Heap is empty.
- **Step 5 -** Display the sorted list.



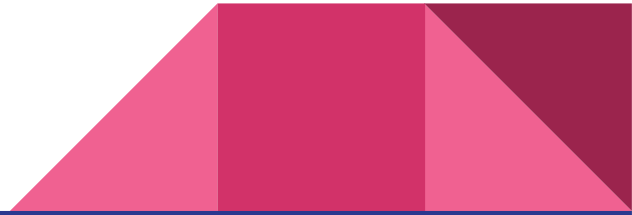
Java Code

```
public void sort(int arr[])
    {
        int n = arr.length;

        // Build heap (rearrange array)
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);

    // One by one extract an element
        from heap
     for (int i = n - 1; i > 0; i--) {
        // Move current root to end
            int temp = arr[0];
             arr[0] = arr[i];
            arr[i] = temp;

            // Call max heapify on the reduced heap
            heapify(arr, i, 0);
        }
    }
// To heapify a subtree rooted with node i which is
    // an index in arr[]. n is size of heap
    void heapify(int arr[], int n, int i)
    {
        int largest = i; // Initialize largest as root
        int l = 2 * i + 1; // left = 2*i + 1
        int r = 2 * i + 2; // right = 2*i + 2

        // If left child is larger than root
        if (l < n && arr[l] > arr[largest])
            largest = l;

    // If right child is larger than largest
        so far
        if (r < n && arr[r] > arr[largest])
            largest = r;

    // If largest is not root
    if (largest != i) {
            int swap = arr[i];
            arr[i] = arr[largest];
             arr[largest] = swap;

            // Recursively heapify the affected sub-tree
            heapify(arr, n, largest);
        }
    }
```

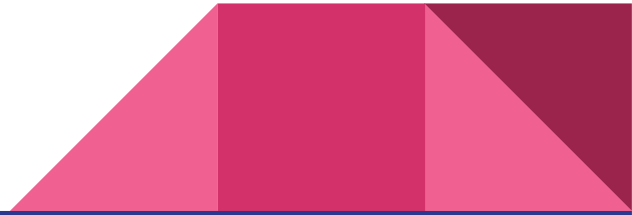
Time Complexity Analysis

- In the heapify() function, we walk through the tree from top to bottom. The height of a binary tree (the root not being counted) of size n is log2 n at most, i.e., if the number of elements doubles, the tree becomes only one level deeper.

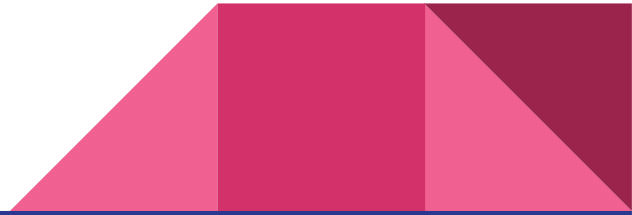
- Time complexity of heapify is O(Logn).



- To initially build the heap, the heapify() method is called for each parent node – backward, starting with the last node and ending at the tree root.
- A heap of size n has n/2 (rounded down) parent nodes.
- Since the complexity of the heapify() method is *O(log n)* as shown above, the complexity for the buildHeap() method is, therefore, maximum* O(n log n).

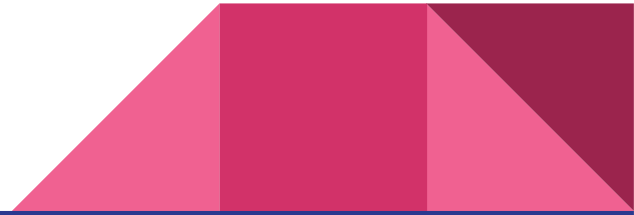


- The heapify() method is called n-1 times. So the total complexity for repairing the heap is also O(n log n).
- Both sub-algorithms, therefore, have the same time complexity. Hence, the time complexity of Heapsort is: O(n log n)

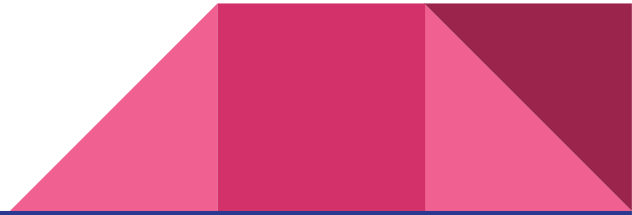


Radix Sort

- This algorithm is only used to sort numbers.
- We sort the numbers from Least Significant Digit(LSD) to Most Significant Digit(MSD).
- We use Counting Sort as a subroutine to sort.



- The data is sorted using the radix sort method, which divides elements into buckets based on their radix.
- For elements with more than one digit, this bucketing process is repeated for each digit, while preserving the ordering of the prior step, until all digits have been considered. For this reason, **radix sort** is also called **bucket sort** and **digital sort**.
- We consider the Radix as the base of a number system.
- For Example, The Decimal number system which has 10 digits from 0 to 9. So, the Radix is 10 and the Radix of binary number system is 2.



Example :

Input List(LSD):

53	89	150	36	633	233

Input List:

53	89	150	36	633	233

Sort on Unit's place:

53	89	150	36	633	233

0	1	2	3	4	5	6	7	8	9
150			53 633 233			36			89

After Sorting:

150	53	633	233	36	89

Sort on ten's place :

150	53	633	233	36	89

0	1	2	3	4	5	6	7	8	9
			633 233 36		150 53			89	

After Sorting :

633	233	36	150	53	89

Sort on hundred's place:

633	233	36	150	53	89

0	1	2	3	4	5	6	7	8	9
36 53 89	150	233				633			

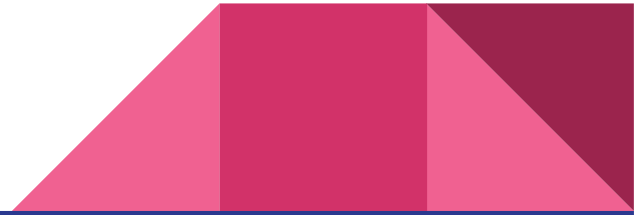
After Sorting:

36	53	89	150	233	633

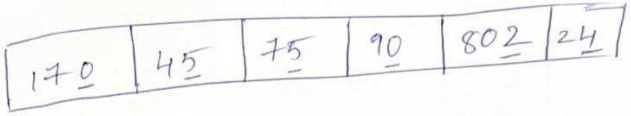
The numbers are now sorted.

Can you solve this input array using Radix Sort?

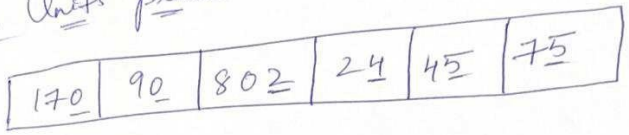
170	45	75	90	802	24



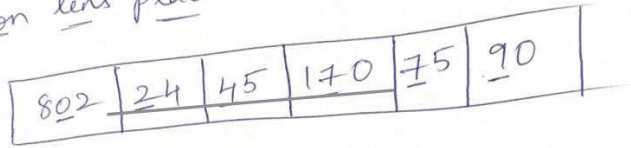
Input List:-



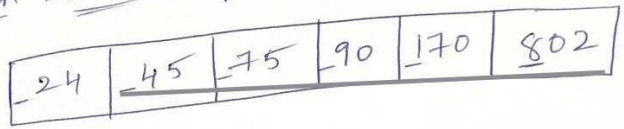
Sort on Unit's place:



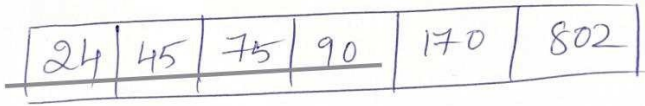
Sort on ten's place:



Sort on hundred's place:

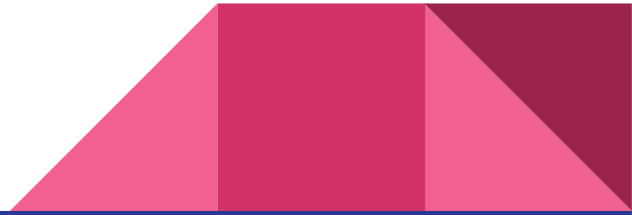


The Sorted array is:-



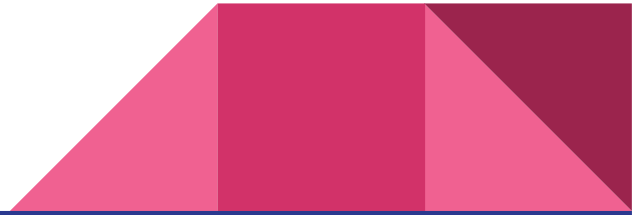
Time Complexity Analysis

- Each pass over 'n' d-digit numbers and ‘b' base keys then it takes time O(n+b).
- There are 'd' passes, so the total time for radix sort is O(d (n+b)).
- When d is a constant and b is much smaller than n, then total run time = O(n).



ALGORITHM

- Create an array a[0…..n-1] elements.
- Call bucket sort repeatedly on least to most significant digit of each element.
- Return the sorted array.



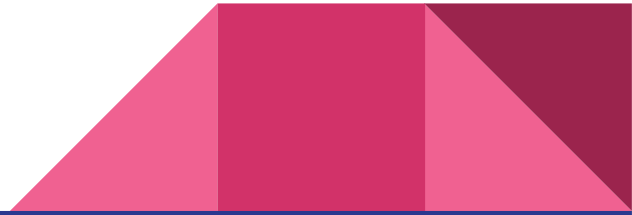
Java Code

```
Void radixsort(int arr[],int n)
{
  Int m = getMax(arr,n); Find the max no. to know no. of digits.
   For (int exp = 1;m/exp > 0;exp *= 10)
       countSort(arr,n,exp);
}
```

The loop applies the Countsort to the nth digit of the elements.

APPLICATIONS

- Mostly used in parallel computing, we divide the input into several buckets, enabling us to sort the buckets in parallel, as they are independent of each other.



THANK YOU