Introduction To Sorting

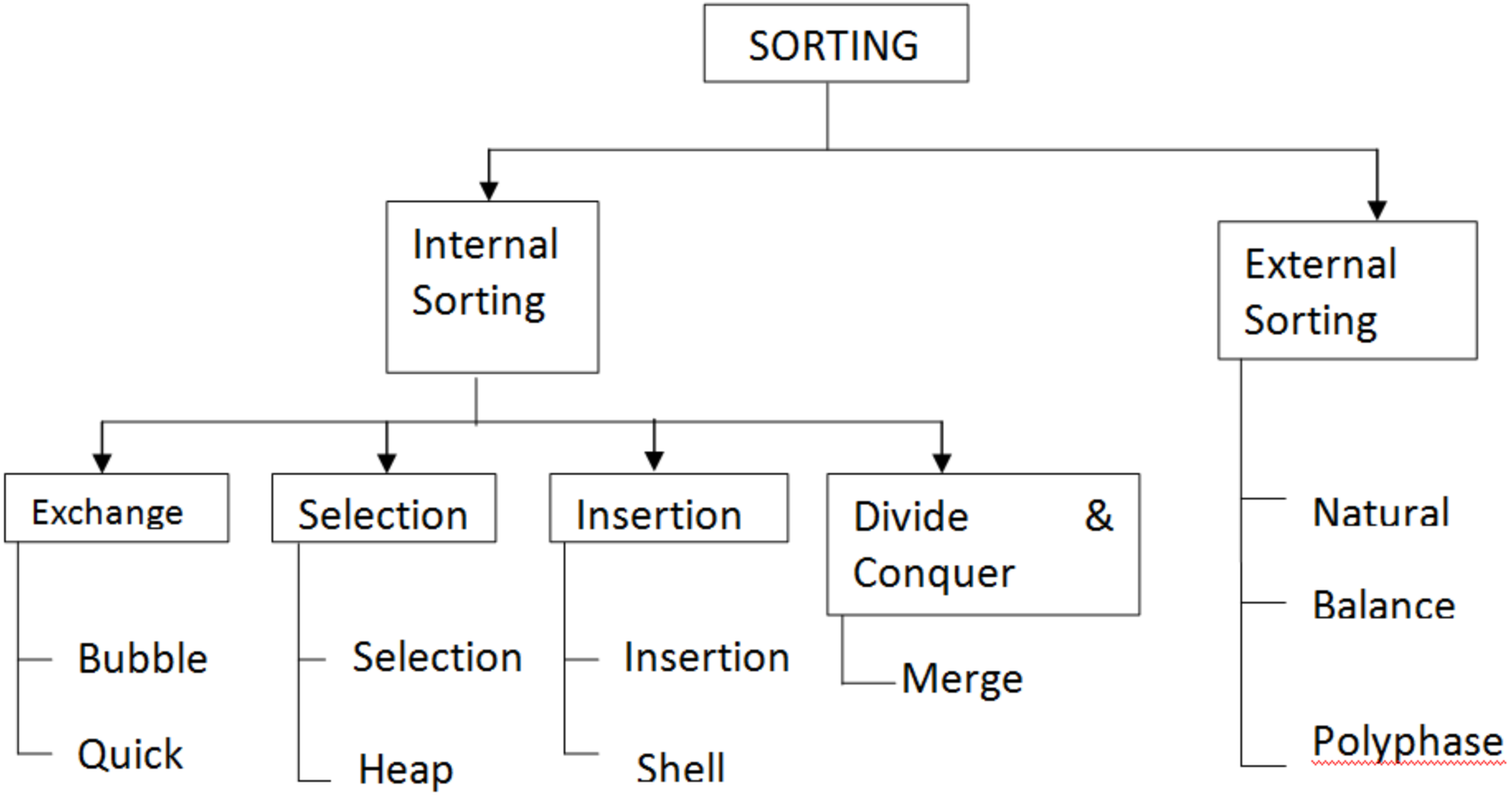
By Yash Gupta

Stable Sort

10	5	10	1	12	10	6
Nilesh	Rishabh	Yash	Karan	Manoj	Arun	Manoj

1	5	6	10	10	10	12
Karan	Rishabh	Manoj	Nilesh	Yash	Arun	Manoj

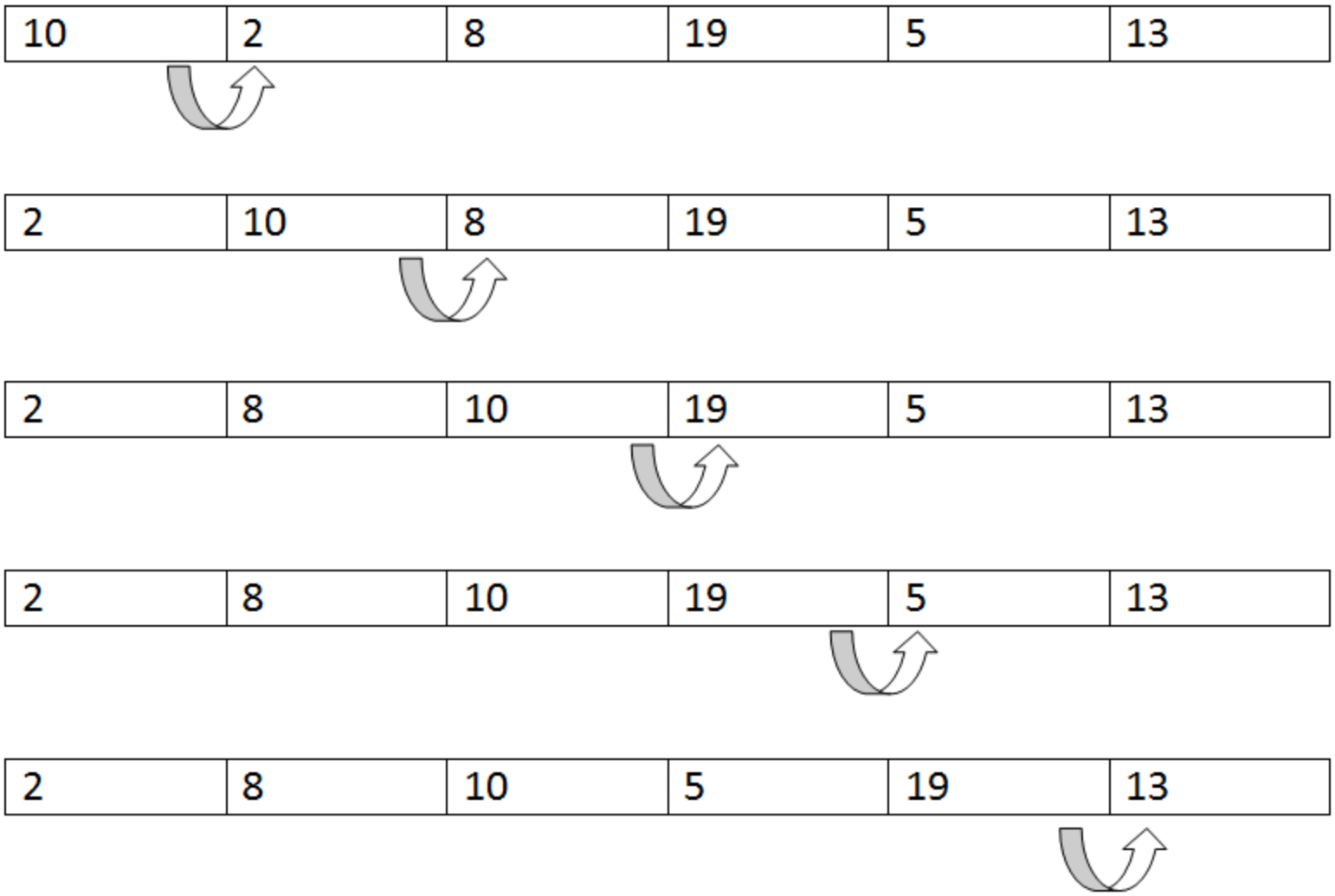
Classification



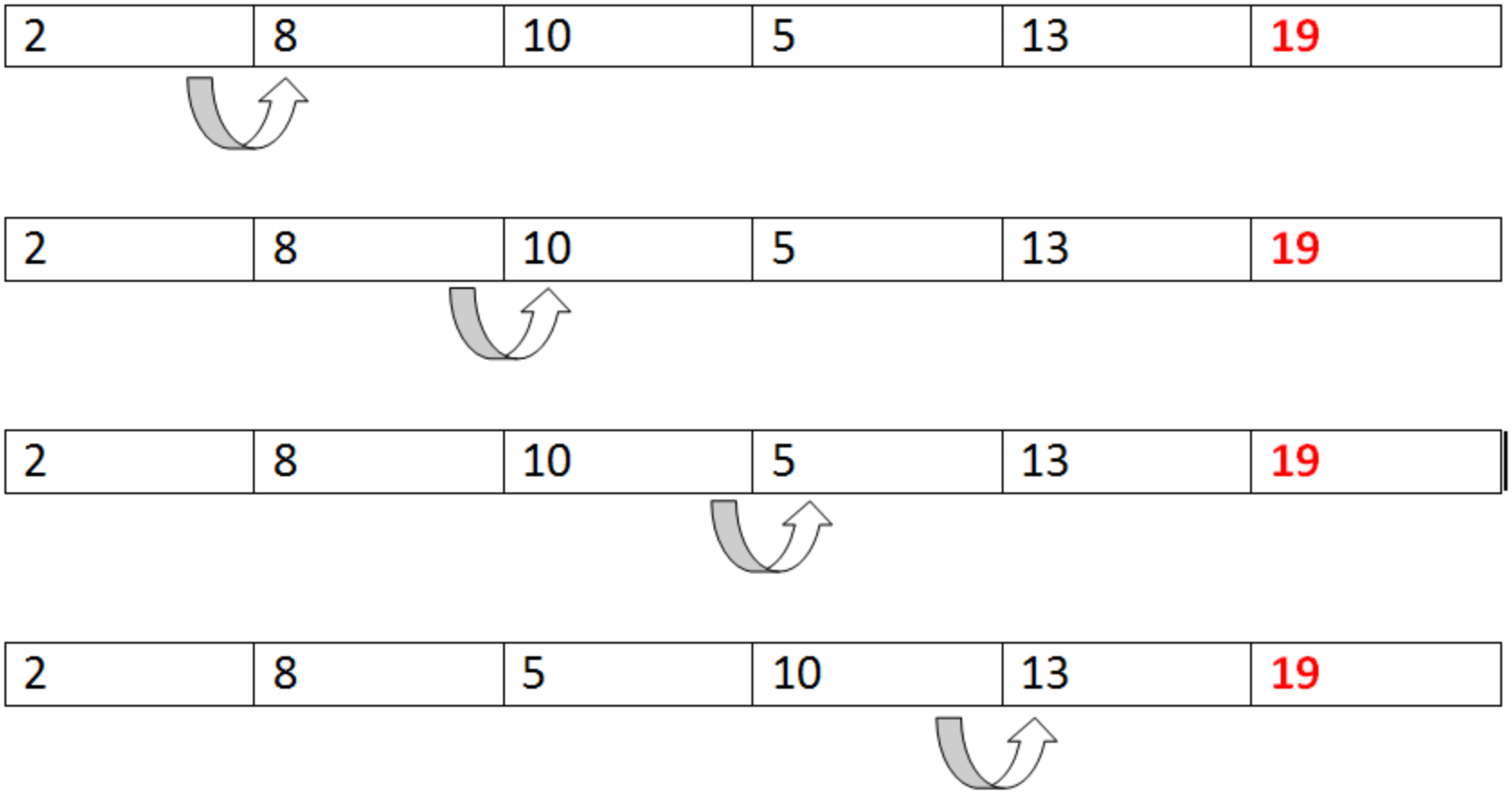
Bubble Sort

10	2	8	19	5	13

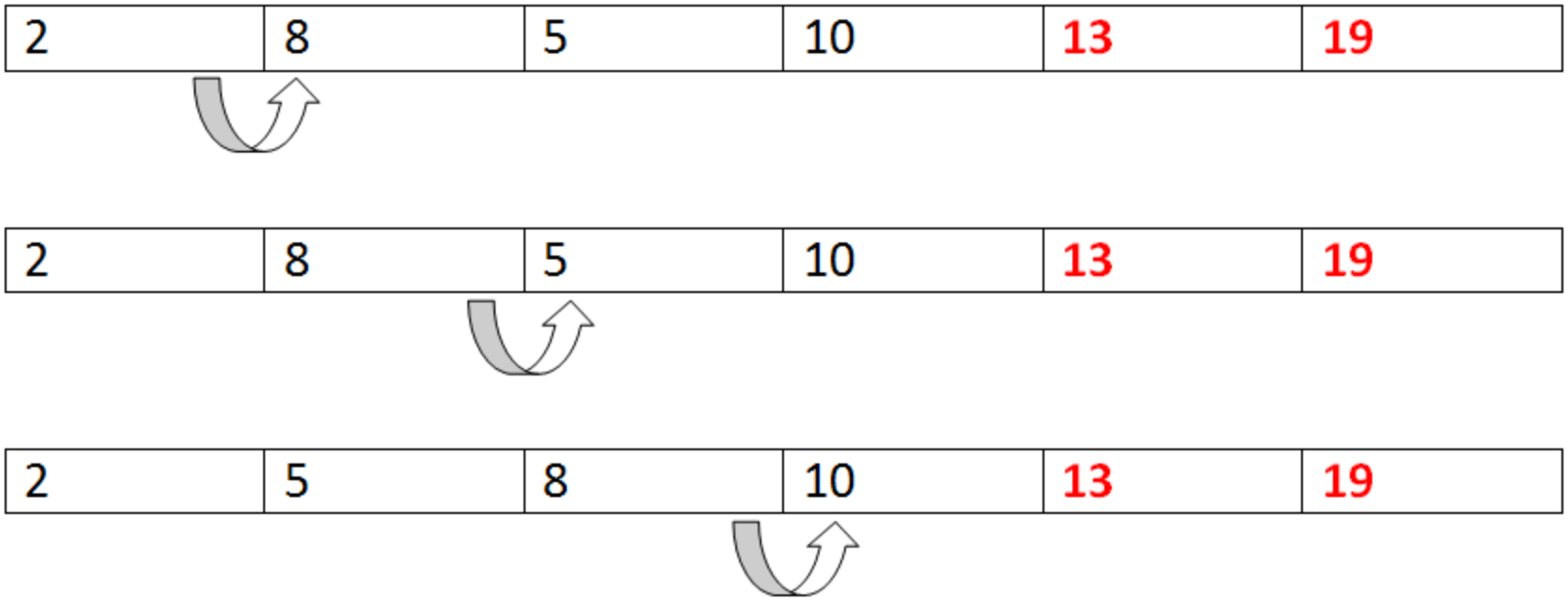
PASS 6



PASS 5



PASS 4



PASS 3 :

2	5	8	10	13	19

2	5	8	10	13	19

PASS 2 :

2	5	8	10	13	19

PASS 1 (IMPLICIT)

2	5	8	10	13	19

```
Algorithm bubbleSort(list[] , n )
Performs bubble sort
Pre     : list contains the elements
        : n is the number of elements in list
Post    : list is sorted

for( pass = 0 ; pass < n - 1 ; pass++)
    for(iteration = 0 ; iteration < (n-1)-pass ; iteration++)
        if( list[iteration] > list[iteration+1] )
            swap(list , iteration , iteration+1 )
        end if
    end for
end for
```

Insertion Sort

PASS 1 (IMPLICIT)

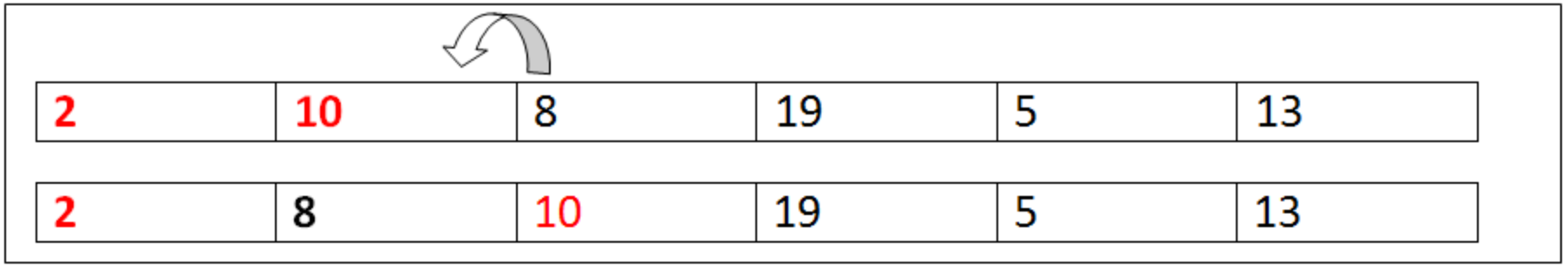
10	2	8	19	5	13

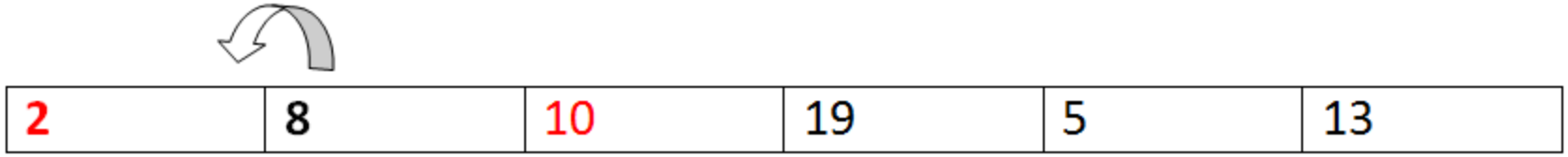
PASS 2

10	2	8	19	5	13

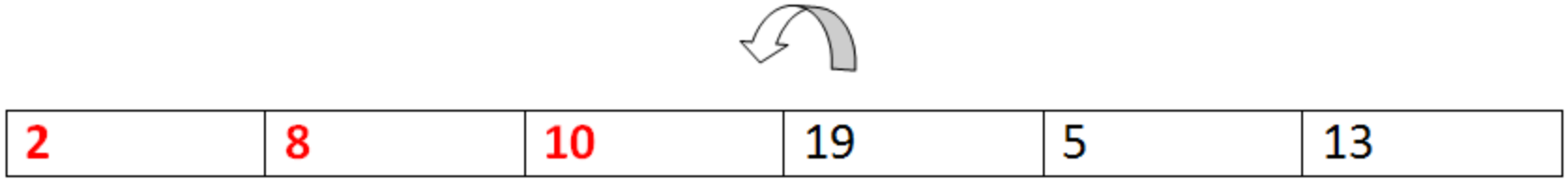
2	10	8	19	5	13

PASS 3

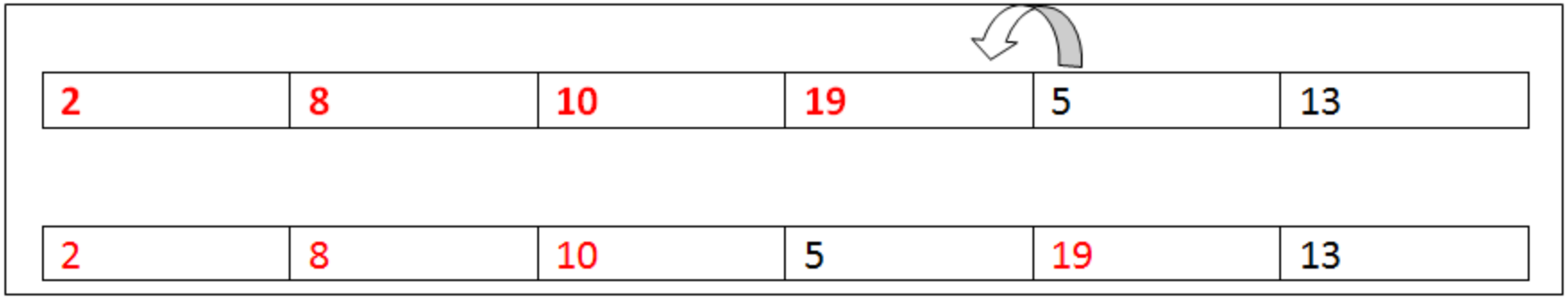


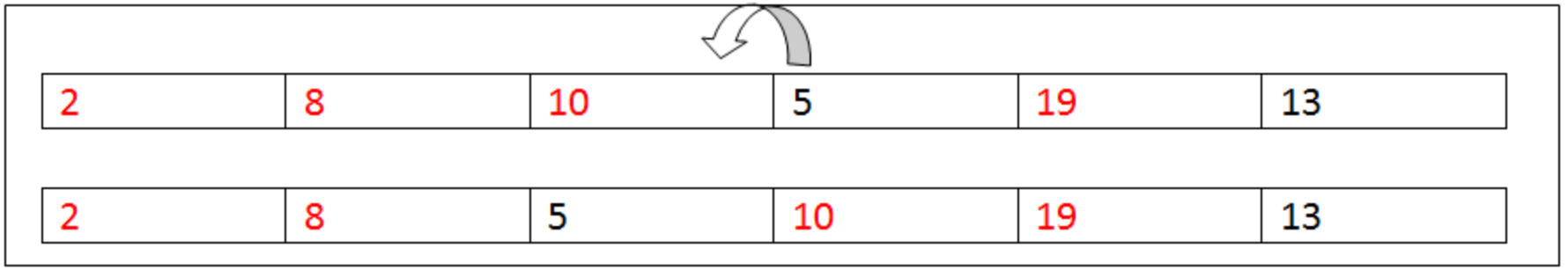


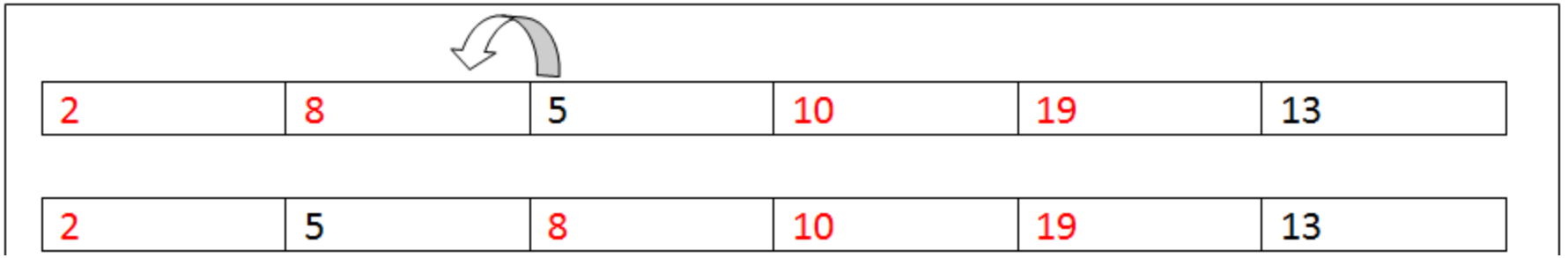
PASS 4



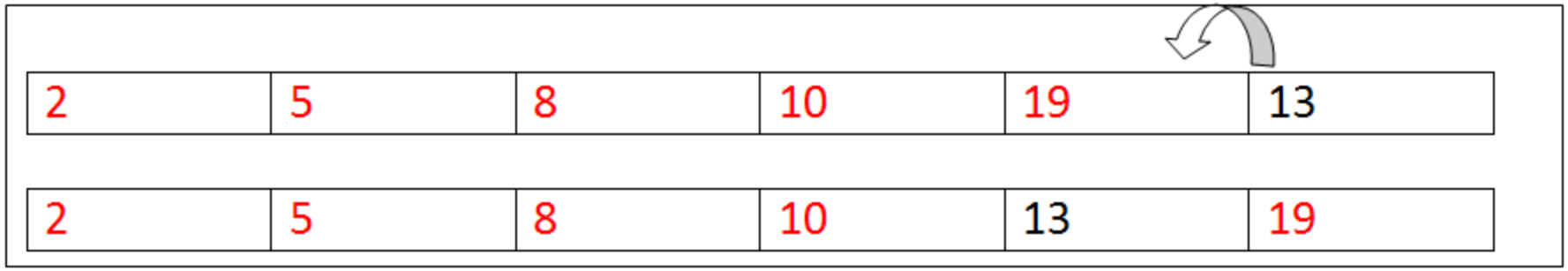
PASS 5

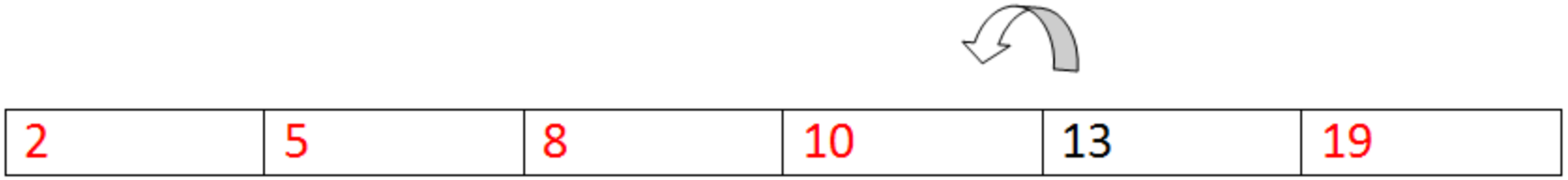

2 8 10 19 5 13
2 8 10 5 19 13


2 8 10 5 19 13
2 8 5 10 19 13


2 8 5 10 19 13
2 5 8 10 19 13

PASS 6





Finally after insertion sort -

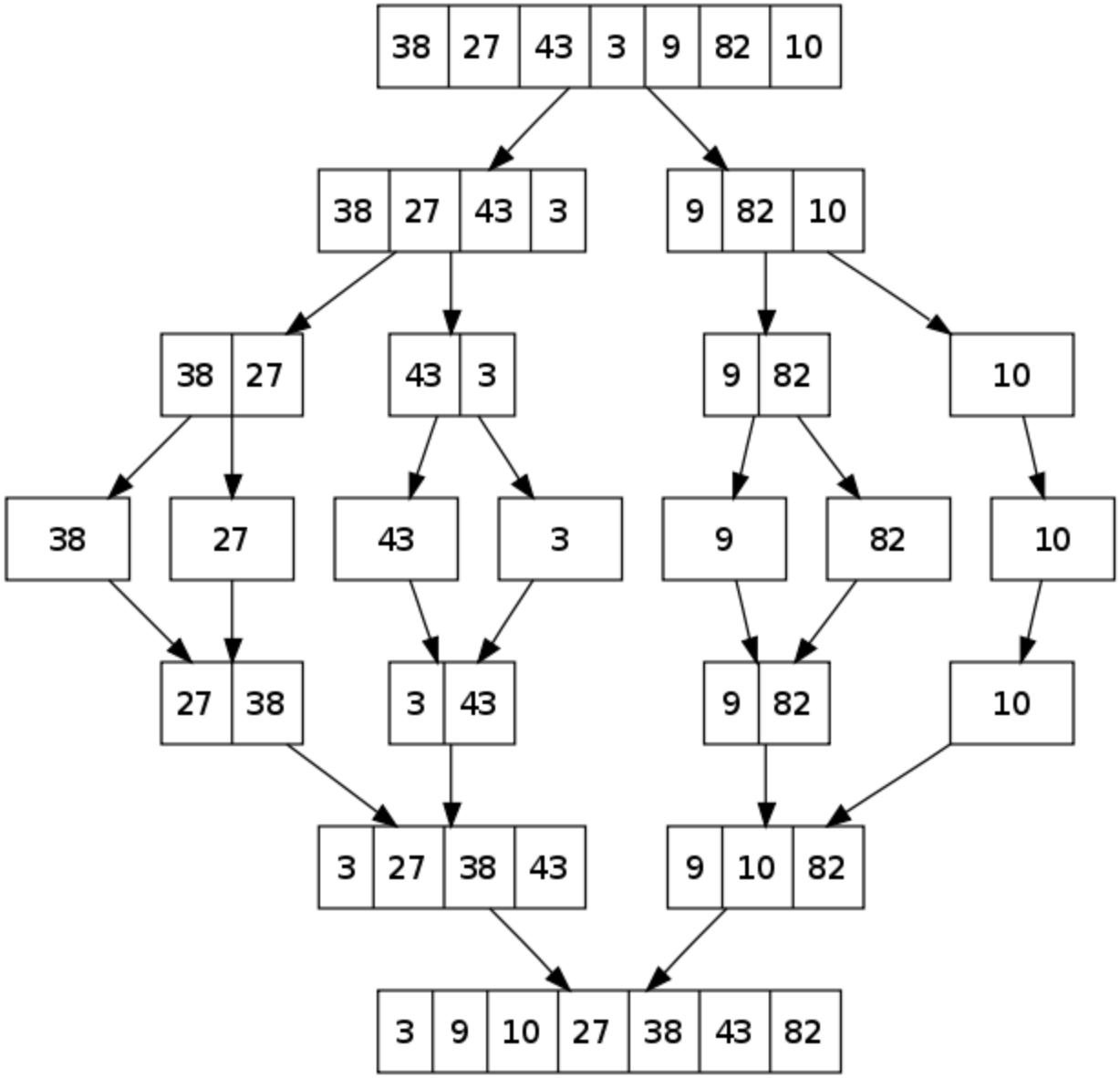
```
Algorithm insertionSort(list ,  n )
Performs insertion sort
Pre     : list contains the elements
        : n is the number of elements in list
Post    : list is sorted

for( pass =1 ; pass < n ; pass++)
    j=pass-1
    while( j >= 0 &&  list[ j ] >  list [ j + 1 ] ) )
        swap(list , j,  j+1)
        j--
    end while
end for
```

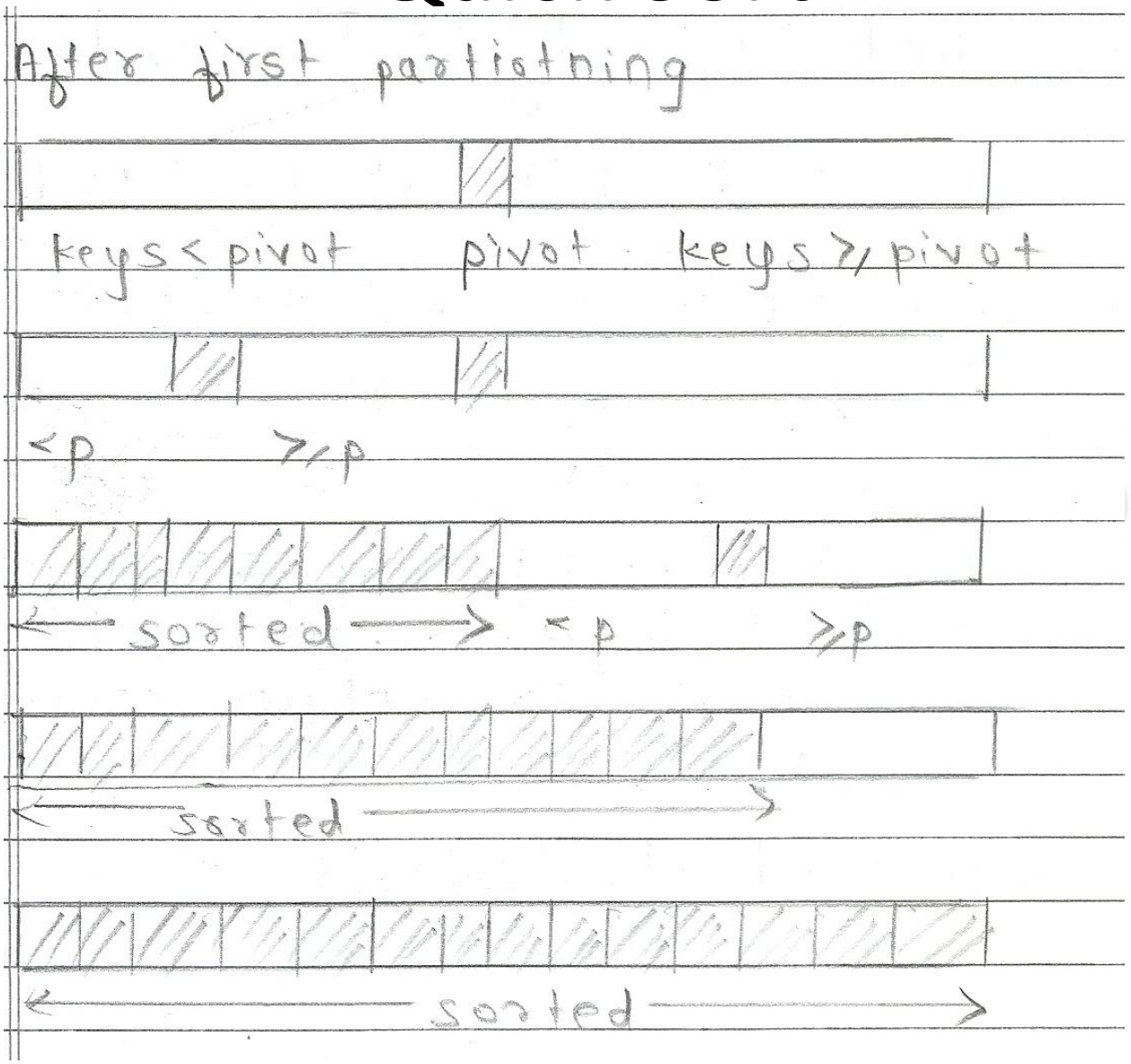
Selection Sort

```
Algorithm selectionSort(list , n  )
Performs selection sort
Pre        : list contains the elements
           : n is the number of elements in list
Post       : list is sorted
for ( i=0 ; i < n-1 ; i++ )
     min= list[i]
     for (j= i + 1 ; j < n ; j++ )
          if ( list[j] < min )
               min = list[j]
               pos=j
          end if
     end for
     swap(list, i, j )
end for
 end selectionSort
```

Merge Sort


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

Quick Sort


After first partioning
keys < pivot
pivot
keys ≥ pivot
< p
≥ p
sorted
< p
≥ p
sorted
sorted

```
Algorithm medianleft( list[] , left , right )
Sorts the left,right,mid elements and swaps median and left element
    Pre    : list contains the elements
           :left and right are the index
    Post   : median is shifted to left position
mid = (left + right ) / 2
if( list[ left ] > list[ mid ] )
    swap(list , left , mid )
end if
if( list[left] > list[ right ] )
    swap(list , left , right )
end if
if( list[ mid ] > list [ right ] )
    swap(list , mid , right )
end if
swap( list , left , mid )
end medianleft
```

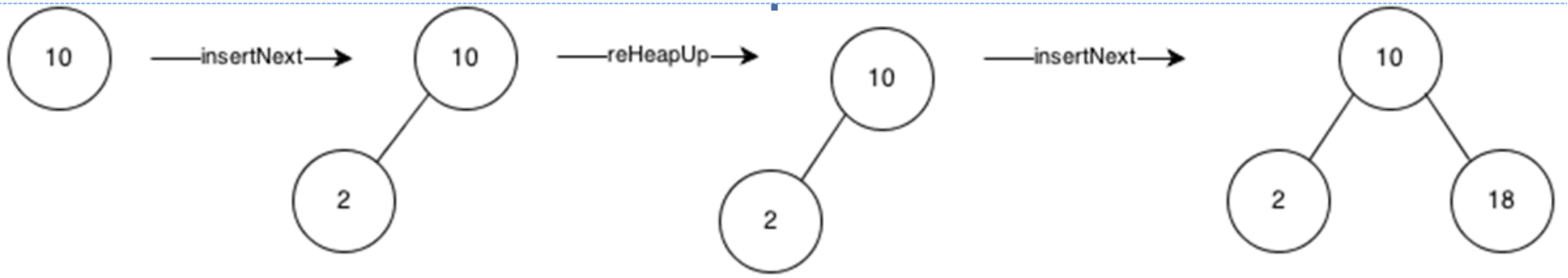
```
Algorithm quickSort(int list[] , int left , int right )
Performs quick sort
    Pre  : list contains the elements
         : left and right are first and last index
         : n is the number of elements in list
    Post: list is sorted
if( (right - left) > minsize )
    partition =  placePivot(list, left, right )
    if( left < partition-1 )
         quicksort(list , left , partition-1 )
    if( partition+1 < right )
         quicksort(list , partition+1, right )
else
    quickInsertion(list , left , right )
end quickSort
```

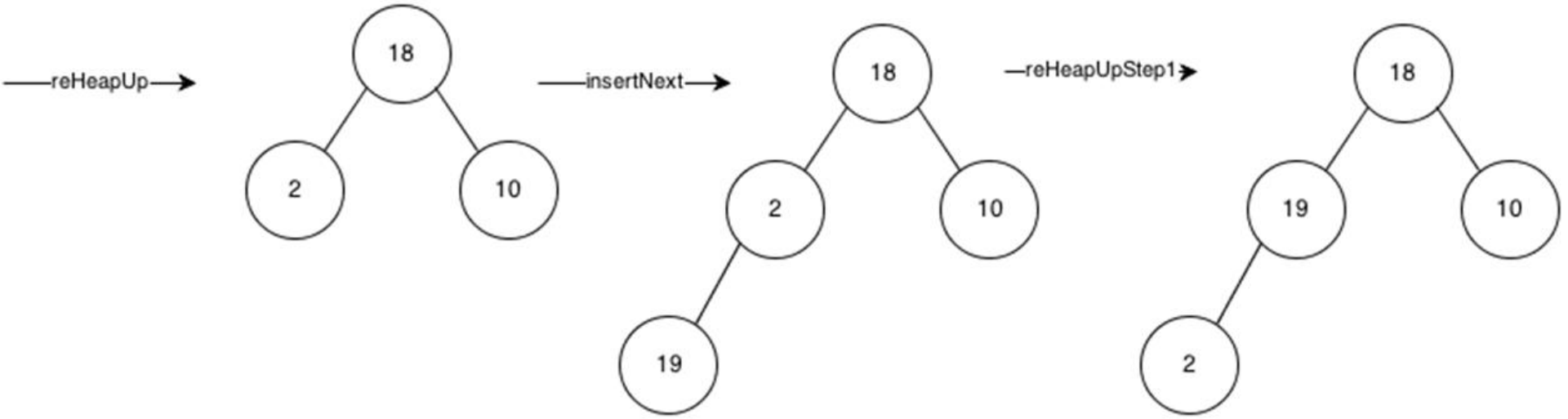
```
Algorithm placePivot(list, left, right )
Performs sorting by placing pivot at its correct position
Performs quick sort
        Pre     : list contains the elements
                : left and right are first and last index
                : n is the number of elements in list
        Post    : list is sorted
medianleft(list , left , right )
pivot = list[left]
sortleft = left + 1
sortright = right
while( sortleft <= sortright )
        while( list[sortleft] < pivot )
                sortleft++
        end while
        while( list[sortright] > pivot )
                sortright—
        end while
        if( sortleft < sortright )
                swap(list , sortleft , sortright );
                sortleft++
                sortright—
        end if
swap(list , left , sortright )
return sortright
end placePivot
```

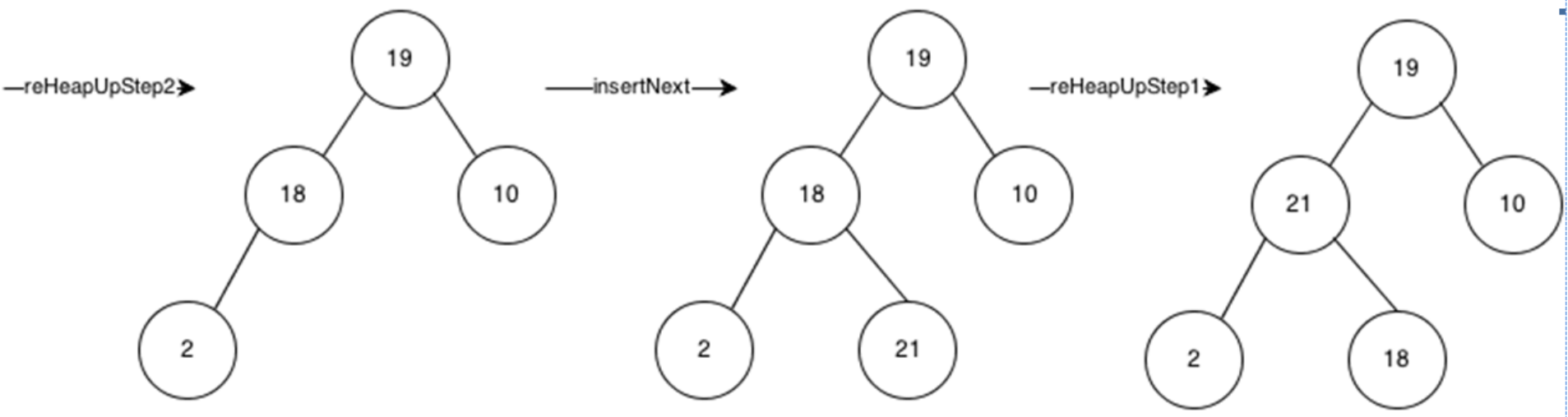
```
Algorithm quickInsertion(list[] , start , last )
Modified insertion sort for quicksort
Pre      : list contains the elements
    : start and last are index
Post     : list is sorted
for(i = start ; i <= last ; i++)
    curr = i - 1
    while( curr >= start && list[curr] > list[curr+1] )
        swap(list , curr , curr+1 )
        curr--
    end while
end for
end quickInsertion
```

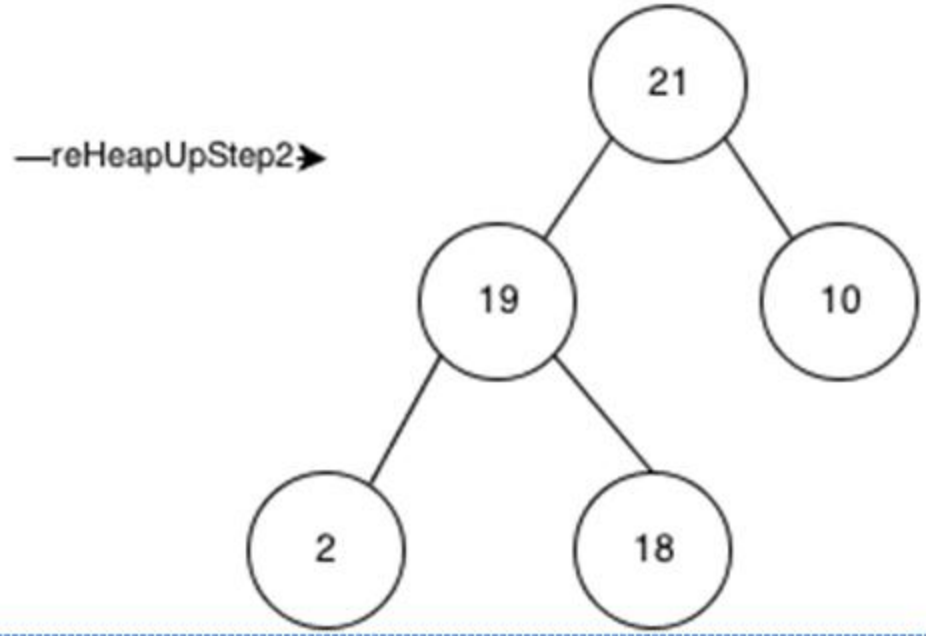
Heap Sort

- A heap is a binary tree structure with following properties
  - The tree is complete or nearly complete
  - The key value of each node is greater than or equal to key value in each of its descendant
- Reheap Up
  - It reorders a broken heap by floating the last element up the tree until it is in its correct location in heap
- Reheap Down
  - It reorders a broken heap by pushing the root down the tree until it is in its correct position


10
insertNext
10
2
reHeapUp
10
2
insertNext
10
2
18


reHeapUp
18
2
10
insertNext
18
2
10
19
reHeapUpStep1
18
19
10
2


reHeapUpStep2
19
18
10
2
insertNext
19
18
10
2
21
reHeapUpStep1
19
21
10
2
18


reHeapUpStep2
21
19
10
2
18

Algorithm reheapup(list , current)

The heap is re-ordered to ensure the new item is placed in correct position

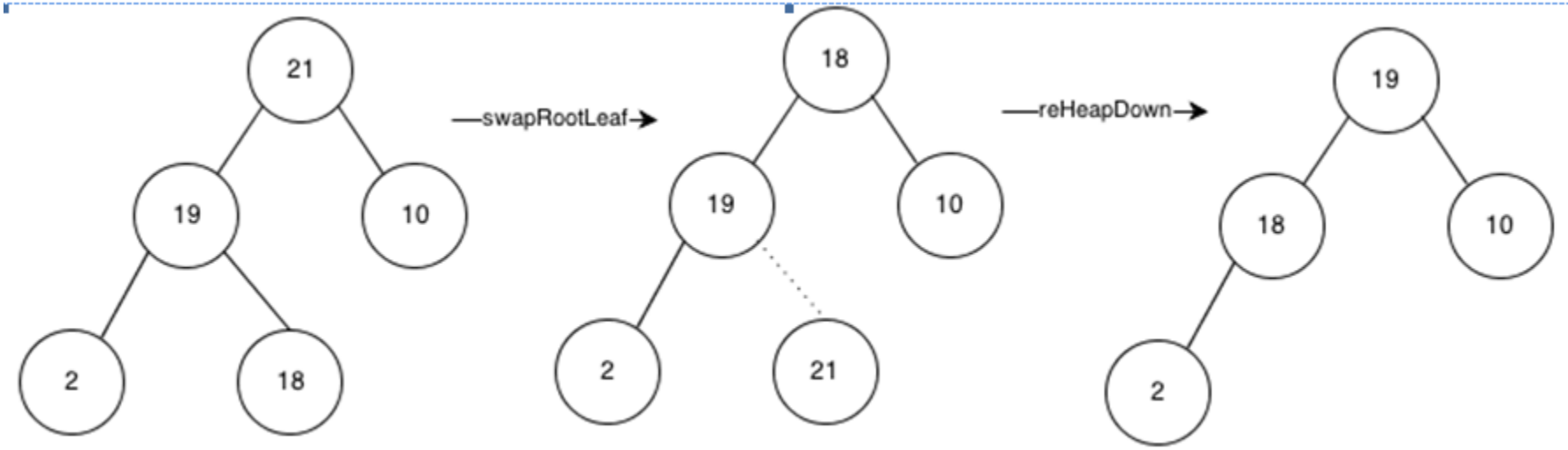
Pre : list stores the items in heap

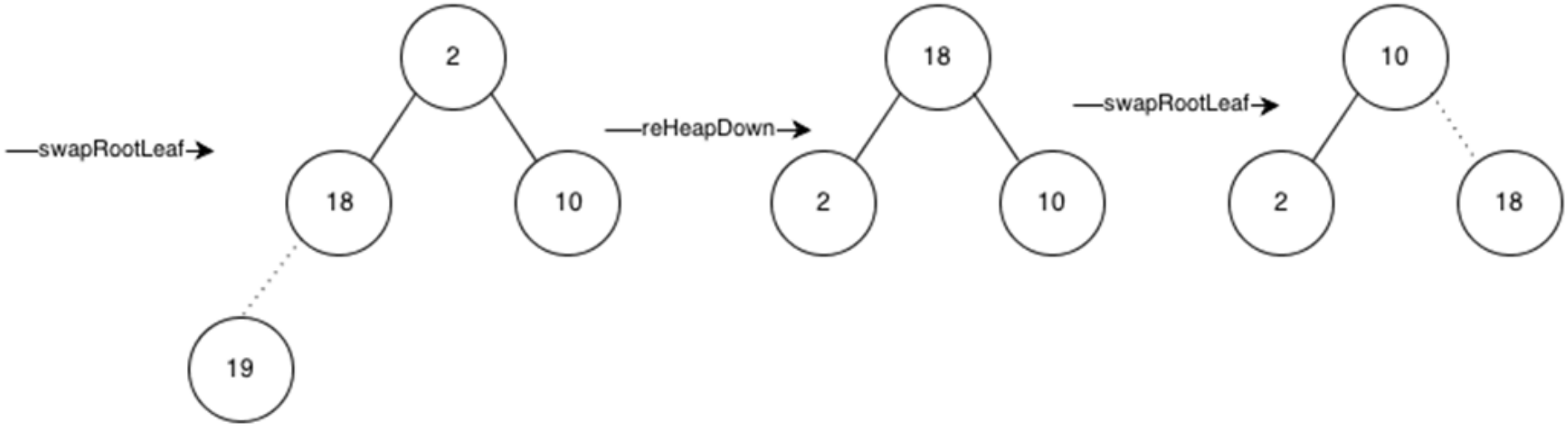
: current is the index of the new element

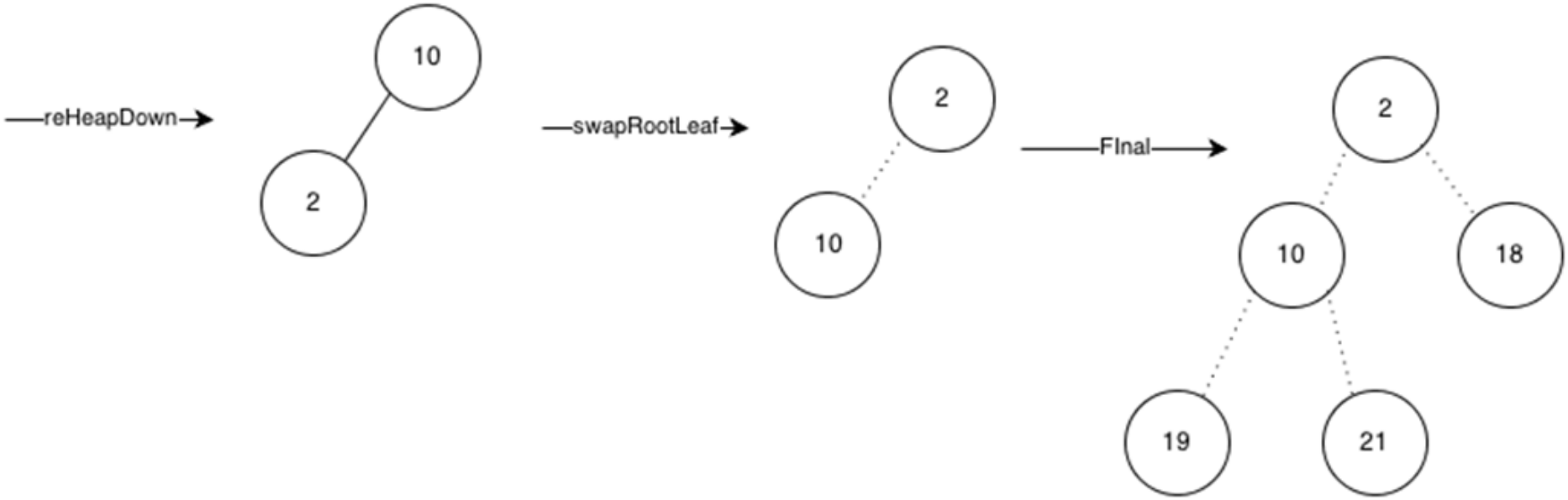
Post : list is re-ordered to maintain heap property

Return: index where the new item is finally placed

```
if( current != root )
   parent = (current-1)/2
   if( list[current] > list[parent] )
        swap(list, current, parent)
        return( reheapup(list , parent ) )
return current
end reheapup
```


21
19
10
2
18
swapRootLeaf
18
19
10
2
21
reHeapDown
19
18
10
2


2
18
10
19
swapRootLeaf
18
2
10
reHeapDown
10
2
18
swapRootLeaf


reHeapDown
10
2
swapRootLeaf
2
10
FInal
2
10
18
19
21

```
Algorithm reheapdown( list,  currindex , lastindex )
The heap is re-ordered to ensure the new item is placed in correct position
      Pre      : list stores the items in heap
               : current is the index of the new element
      Post     : list is re-ordered to maintain heap property
      Return   : index where the new item is finally placed
leftindex= 2 * currindex + 1
       if( leftindex <= lastindex)
              leftdata = list[leftindex]
              rightindex = 2*currindex + 2
              if( rightindex <=lastindex )
                     rightdata = list[rightindex]
              else
                     rightdata = notprsent
              end if
              if( leftdata > rightdata )
                     largetree index= leftindex
              else
                     largetreeindex = rightindex
              end if
              if( list[ largetreeindex] > list [ lastindex ] )
                     swap(list, largetreeindex, lastindex)
                     reheapdown(list , largetreeindex , lastindex )
              end if
       end if
return currindex
end reheapdown
```

Contact Info

- trainers@finaldesk.com
- rishabh@finaldesk.com
- nilesh@finaldesk.com
- jignesh@finaldesk.com
- yash@finaldesk.com
- anand@finaldesk.com