Quach Duc Thien
23020161

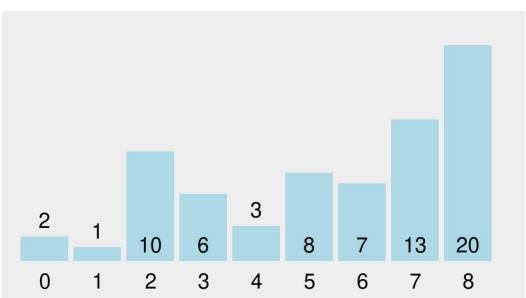
Exercise 1: Given a list of integer numbers: [2, 1, 10, 6, 3, 8, 7, 13, 20]. Draw the steps to sort (ascending sort) the list by following methods :

1. **Selection sort**
   a) Selection sort is a sorting algorithm that works by repeatedly selecting the smallest (or largest) element from unsorted portion of the list and moving it to the sorted portion of the list
   b) Pseudo Code

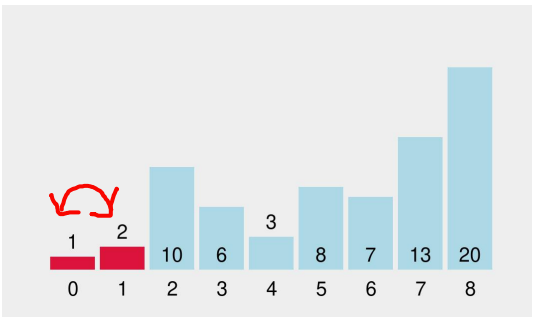
```
repeat (numOfElements - 1) times

  set the first unsorted element as the minimum

  for each of the unsorted elements

    if element < currentMinimum

      set element as new minimum

  swap minimum with first unsorted position
```

   c) Steps to sort list [2, 1, 10, 6, 3, 8, 7, 13, 20]

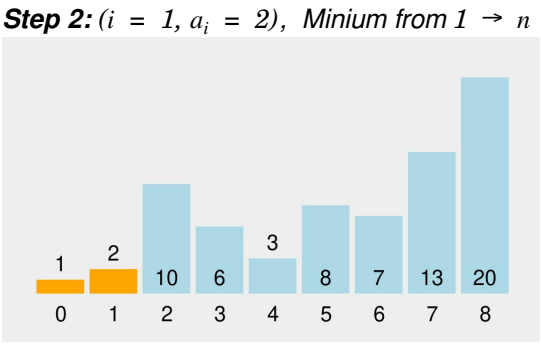
Step 0: *Original list*



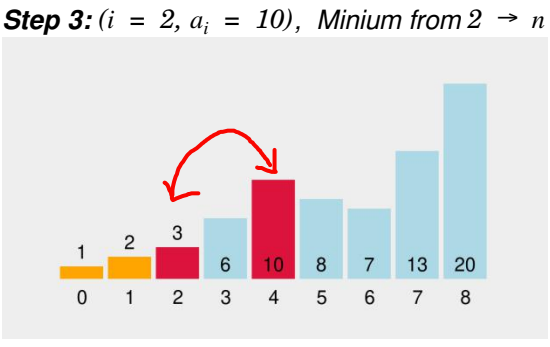
Step 1: *$(i = 0, a_i = 2)$, Minium from $0 \rightarrow n - 1$ is $a_1 = 1$. Swap(a_0, a_1)*



Step 2: *$(i = 1, a_i = 2)$, Minium from $1 \rightarrow n - 1$ is $a_1 = 2$. Swap(a_1, a_1). This step list no change*



Step 3: *$(i = 2, a_i = 10)$, Minium from $2 \rightarrow n - 1$ is $a_4 = 3$. Swap(a_2, a_4).*



Step 4: *Do the same step 1,2, 3 with i from $3, 4, \ldots$*
....
Finally*, we have the sorted list*

   d) Complexity
      i. Worst case time complexity $O(n^2)$
      ii. Best case time complexity $O(n^2)$

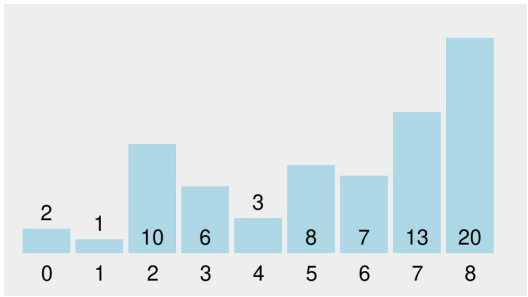
2. Insertion Sort

   a) Insertion sort is a sorting algorithm that works by iteratively inserting each element of an unsorted list into its correct position in a sorted portion of the list.
   b) Pseudo Code

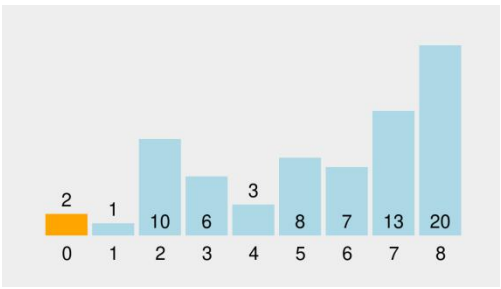
```
mark first element as sorted

for each unsorted element X

  'extract' the element X

  for j = lastSortedIndex down to 0

    if current element j > X

      move sorted element to the right by 1

    break loop and insert X here
```

   c) Steps to sort list [2, 1, 10, 6, 3, 8, 7, 13, 20]

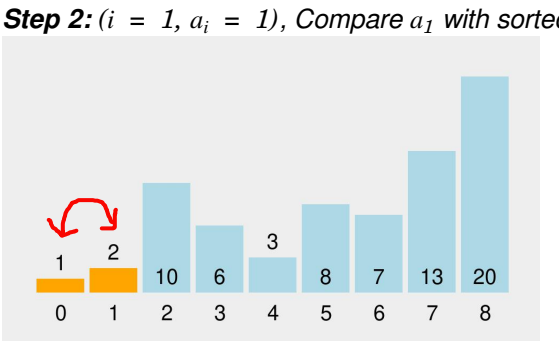
Step 0: *Original list*



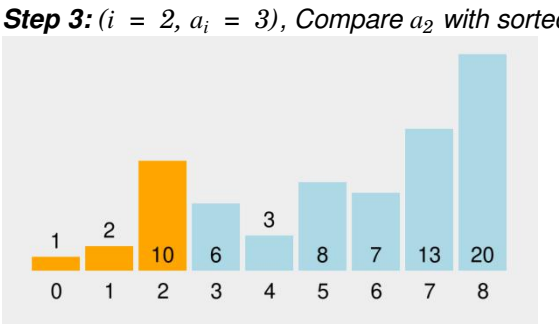
Step 1: *Mark the first element* a_1 *as sorted*



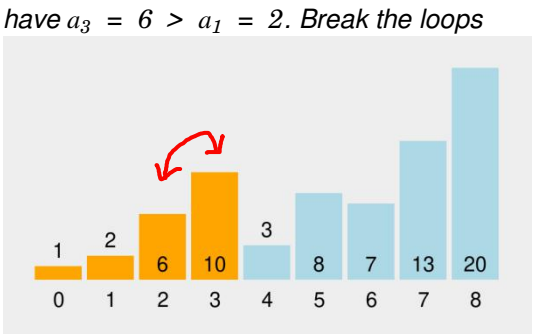
Step 2: *(*$i = 1, a_i = 1$*), Compare* a_1 *with sorted list* $[a_0]$ *have* $a_1 = 1 < a_0 = 2$*. Swap*(a_1, a_0)*.*



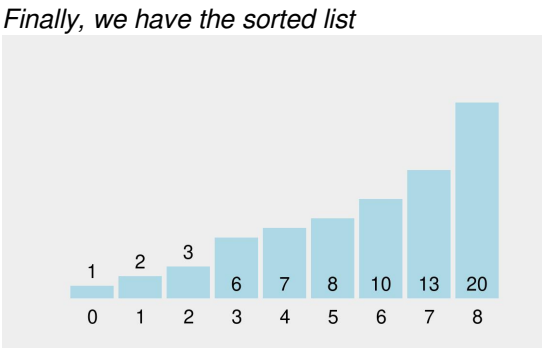
Step 3: *(*$i = 2, a_i = 3$*), Compare* a_2 *with sorted list* $[a_0, a_1]$ *have* $a_2 = 10 > a_1 = 2$*. Break the loops*



Step 3: *(*$i = 3, a_i = 6$*), Compare* a_3 *with sorted list* $[a_0, a_1, a_2]$ *have* $a_3 = 6 < a_2 = 2$*. Swap*(a_3, a_2)*.*
have $a_3 = 6 > a_1 = 2$*. Break the loops*



Step 4: *Do the same step 1,2, 3 with* i *from* $4, 5, \ldots$
....
Finally, we have the sorted list



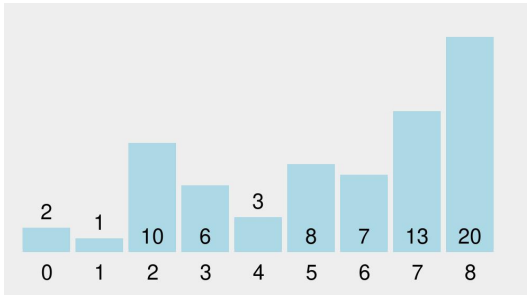
d) Complexity
   i. Worst case time complexity $\mathrm{O}(\mathrm{n}^2)$
   ii. Best case time complexity $\mathrm{O}(\mathrm{n})$

3. **Bubble sort**
   a) Selection sort is a sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order.
   b) Pseudo Code

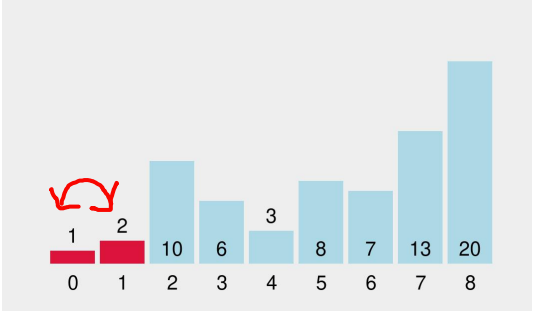
```
do
  swapped = false
  for i = 1 to indexOfLastUnsortedElement-1
    if leftElement > rightElement
      swap(leftElement, rightElement)
      swapped = true; ++swapCounter
while swapped
```

   c) Steps to sort list [2, 1, 10, 6, 3, 8, 7, 13, 20]

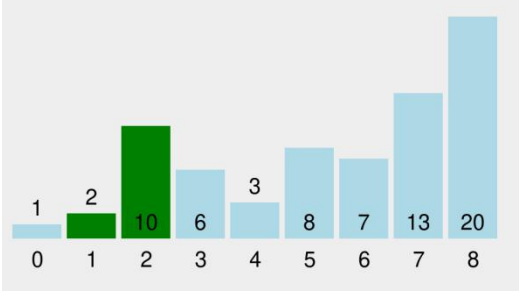
      Step 0: *Original list*



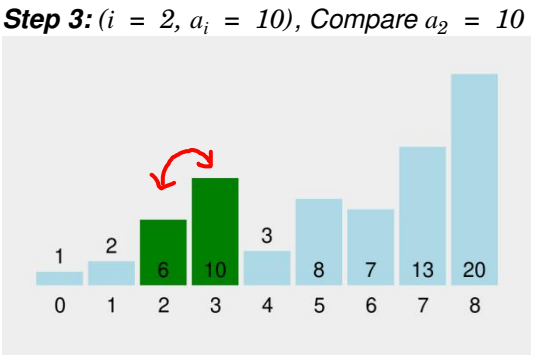
Step 1: *(*$i = 0, a_i = 2$*), Compare* $a_0 = 2 > a_1 = 1$*. Swap*(a_0, a_1)



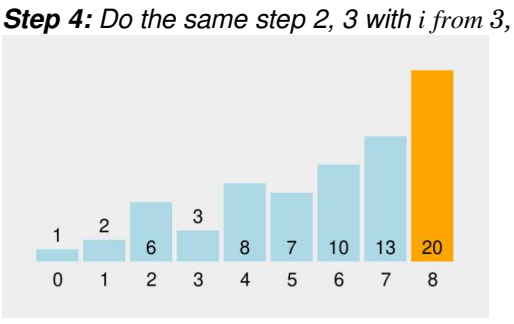
Step 2: *(*$i = 1, a_i = 2$*), Compare* $a_1 = 2 < a_2 = 10$*. No swap here*



Step 3: *(*$i = 2, a_i = 10$*), Compare* $a_2 = 10 > a_3 = 6$*. Swap*(a_2, a_3)

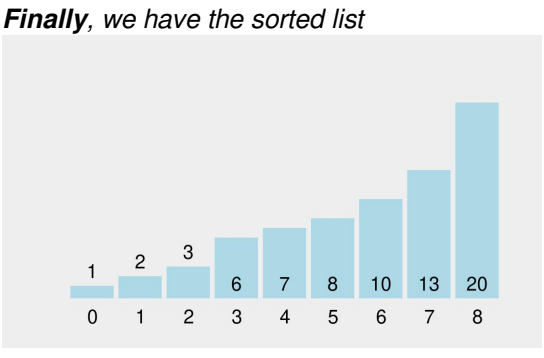


Step 4: *Do the same step 2, 3 with* i *from* $3, 4, \ldots$ *we have the right order of largest number in the end*



Step 5: *Comeback again step 1 until* ***boolean swapped*** *is* ***false****;*

...
Finally*, we have the sorted list*



d) Complexity
   i. Worst case time complexity $O(n^2)$
   ii. Best case time complexity $O(n)$

4. Merge Sort

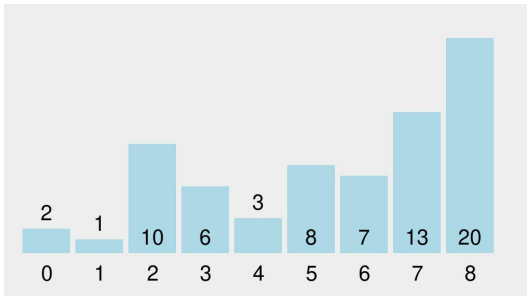
a) Merge sort is a sorting algorithm that follows the divide-and-conquer approach. It works by recursively dividing the input array into smaller subarrays and sorting those subarrays then merging them back together to obtain the sorted array.

b) Pseudo Code

```
split each element into partitions of size 1

recursively merge adjacent partitions

  for i = leftPartIdx to rightPartIdx

    if leftPartHeadValue <= rightPartHeadValue

      copy leftPartHeadValue

    else: copy rightPartHeadValue; Increase InvIdx

copy elements back to original array
```

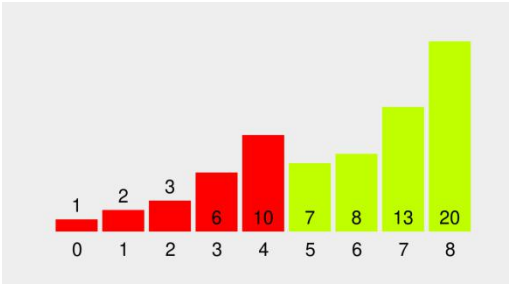
c) Steps to sort list [2, 1, 10, 6, 3, 8, 7, 13, 20]

Step 0: *Original list*

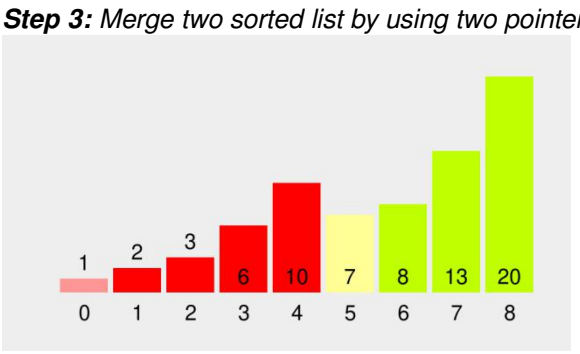


Step 1: *Split the list into two part with* $middle\ is\ n\ /\ 2$

Step 2: *Call the recursively merge sort to have sorted list from* $0 \rightarrow 4$ *and* $5 \rightarrow 8$



Step 3: *Merge two sorted list by using two pointer : Compare each element from begin to the end of sorted list*



Step 4: $a_0 = 1 < a_5 = 7$. *We push* a_0 *to final sorted list and increase the index of compare element in first part new sorted list :* $[a_0]$

Step 5: $a_1 = 2 < a_5 = 7$. *We push* a_1 *to final sorted list and increase the index of compare element in first part new sorted list :* $[a_0, a_1]$

Step 6: *Do the same step 4, 5 until all of element from two sorted list*

...

Finally*, we have the sorted list*

d) Complexity
   i. Worst case time complexity $O(nlogn)$
   ii. Best case time complexity $O(nlogn)$

5. Radix Sort

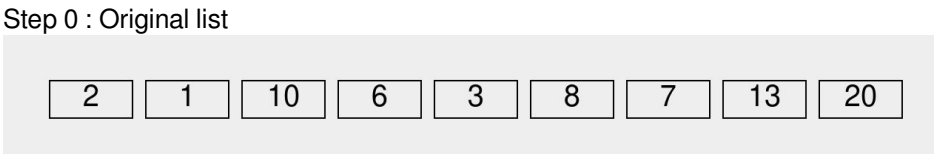
a) Radix Sort is a linear sorting algorithm that sorts elements by processing them digit by digit. It is an efficient sorting algorithm for integers or strings with fixed-size keys.

b) Pseudo Code

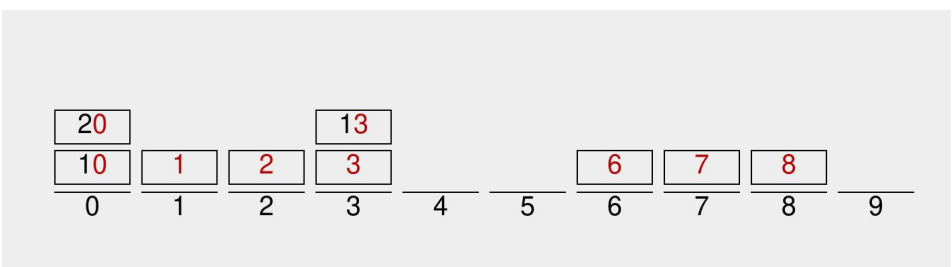
```
create 10 buckets (queues) for each digit (0 to 9)

for each digit placing

 for each e in list, move e into its bucket

   for each bucket b, starting from smallest digit

     while b is non-empty, restore e to list
```

c) Steps to sort list [2, 1, 10, 6, 3, 8, 7, 13, 20]

Step 0 : Original list

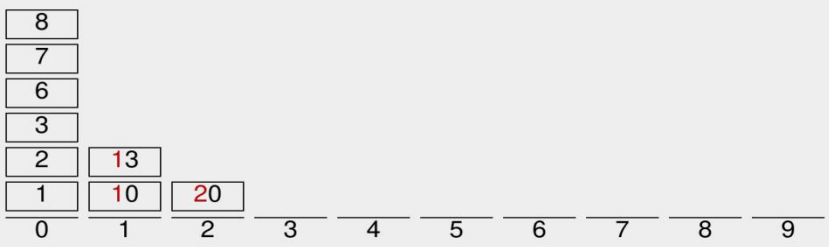


Step 1 : Consider the from last digit to first digit of all number, move it to the its digit bucket

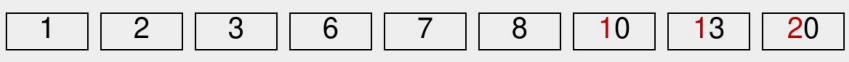


Step 2 : Starting from smallest digit, restore list

Step 3 : Continue with second digit from last



Step 4 : Starting from smallest digit, restore list



Step 5 : Repeat until out of digit we have the sorted list

d) Complexity
   i. Worst case time complexity $O(n * log10(max(a))$
   ii. Best case time complexity $O(n * log10max(a))$

6. Heap Sort

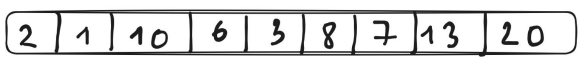
a) Heap sort is a comparison-based sorting technique based on Binary Heap Data Structure. It can be seen as an optimization over selection sort where we first find the max (or min) element and swap it with the last (or first). We repeat the same process for the remaining elements

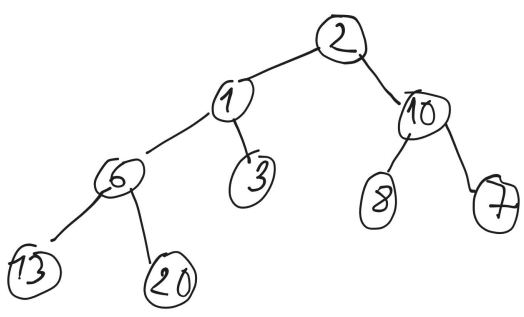
b) Pseudo Code

```
Heapify(A, i) {
   le <- left(i)
   ri <- right(i)
   if (le<=heapsize) and (A[le]>A[i])
      largest <- le
   else
      largest <- i
   if (ri<=heapsize) and (A[ri]>A[largest])
      largest <- ri
   if (largest != i) {
      exchange A[i] <-> A[largest]
      Heapify(A, largest)
   }
}

create the heap
for i <- length(A) downto 2 {
      exchange A[1] <-> A[i]
      heapsize <- heapsize -1
      Heapify(A, 1)
```

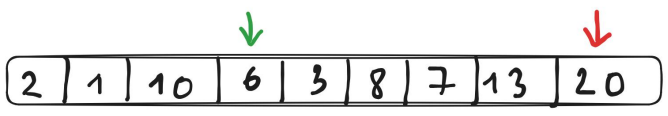
c) Steps to sort list [2, 1, 10, 6, 3, 8, 7, 13, 20]

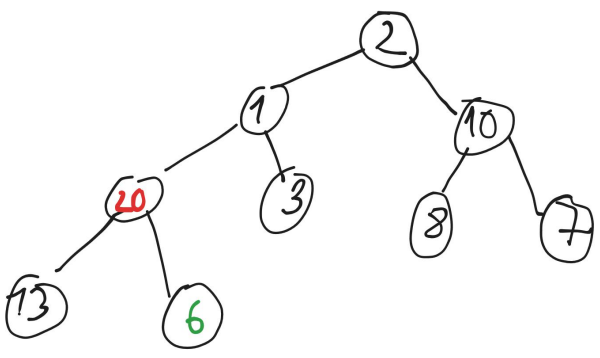
Step 0: Original list and build the heap tree



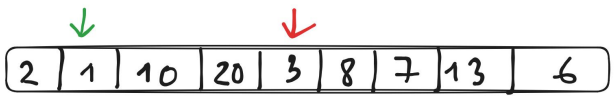


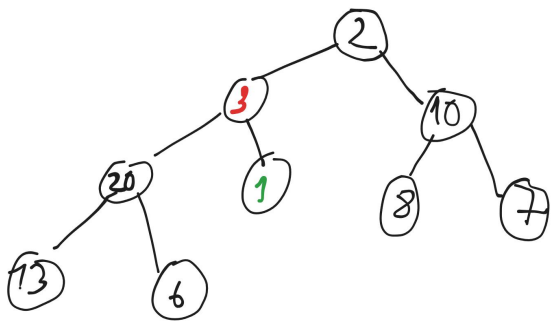
Step 1: From the leaves node, swap this to the parent if parent value is larger than. Swap(20, 6)



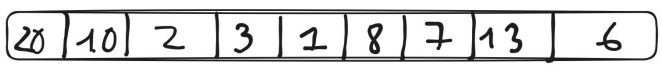


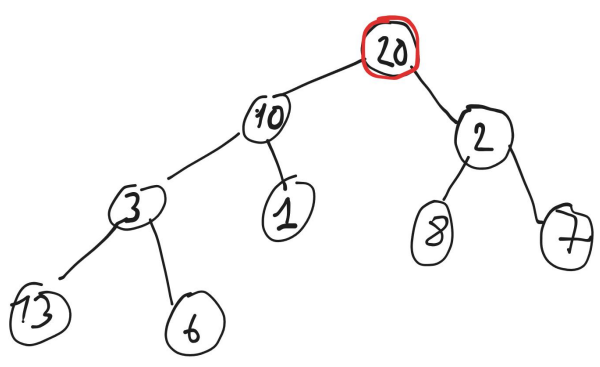
Step 2: Continue with next node, 20 > 13 so no swap here
Step 3: Continue with next node 10 > 7, so no swap here
Step 4: Continue with next node 10 > 7, so no swap here
Step 5: Continue with next node 3 > 1, Swap(3, 1)



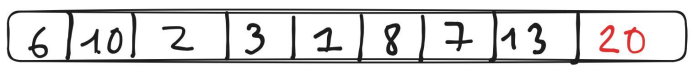


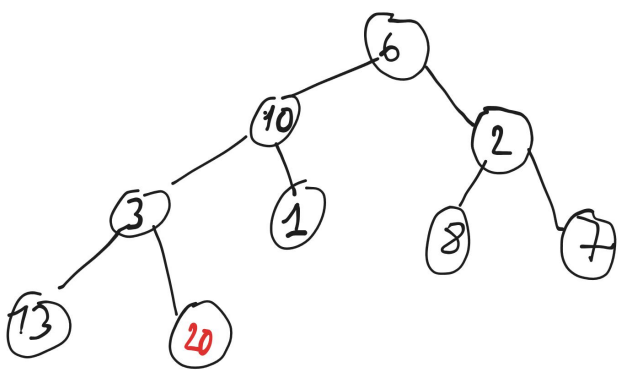
Step 6: Repeated until to the top node





Step 7: Swap the top node to the last unsorted node and repeated from step 1 until all list sorted





Finnaly, we have sorted list ....

d) Complextity
   i. Worst case time complexity $O(nlogn)$
   ii. Best case time complexity $O(nlogn)$

7. Quick Sort

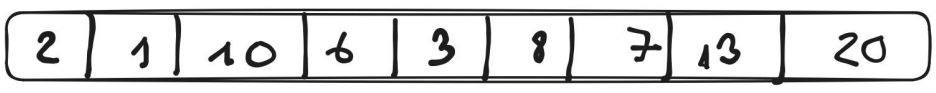
a) QuickSort is a sorting algorithm based on the Divide and Conquer that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array.

b) Pseudo Code

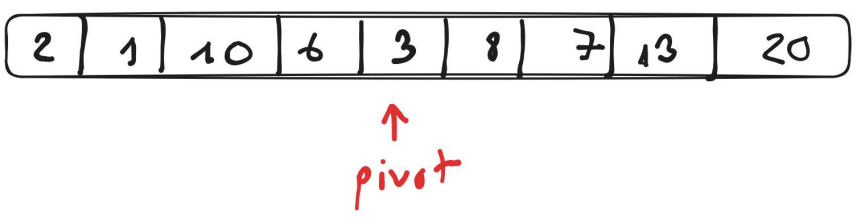
```
select the random pivot
i = left_most_index, j = right_most_index
while(i <= j) :
       while(a[i] < pivot) increase i
       while(a[j] > pivot) decrease j
       swap(a[i], a[j])
       increase i
       decrease j
   call recursively(left_most_index, j)
   call recursively(right_most_index, i)
```

Steps to sort list [2, 1, 10, 6, 3, 8, 7, 13, 20]

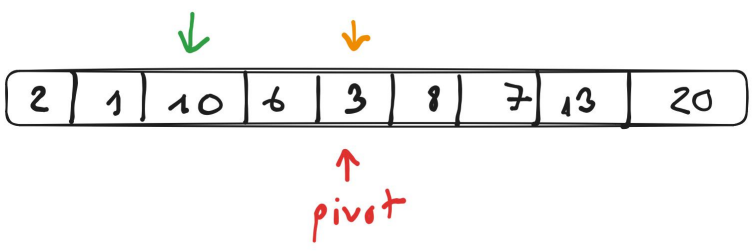
Step 0: *Original list*



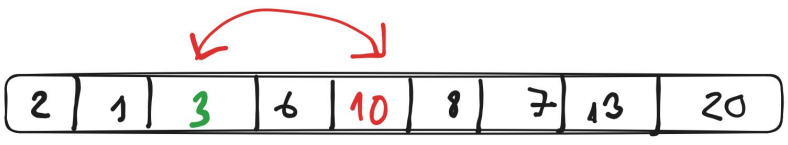
Step 1: Choose the pivot is 3



Step 2 : Loop to find the a[i] have a[i] >= pivot, i now is 10 (green), and a[j] <= pivot j now is 3 (orange)



Step 3: Swap (i, j) we have the right order or pivot, all left element smaller than pivot and right element larger than pivot



Step 4: Call the `recursively` to sort left subarray and right subarray

Finally, we have the sorted array

c) Complexity

i. Worst case time complexity $O(n^2)$

ii. Best case time complexity $O(nlogn)$