GIT Department of Computer Engineering

CSE 222/505 - Spring 2020

Homework 6

Q1:

a) A = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

b) B = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1}

c) C = {5, 2, 13, 9, 1, 7, 6, 8, 1, 15, 4, 11}

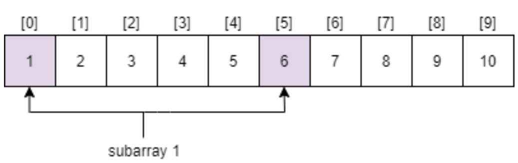
d) D = {‘S’, ‘B’, ‘I’, ‘M’, ‘H’, ‘Q’, ‘C’, ‘L’, ‘R’, ‘E’, ‘P’, ‘K’}

a)

Shell Sort:

gap value = A.length / 2 = 5

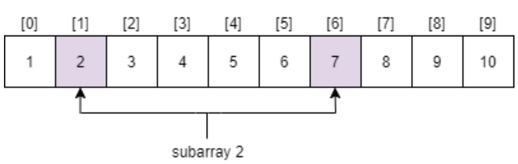
Sort subarray 1:



Compare 6 and 1.

6 > 1 , there is no change.

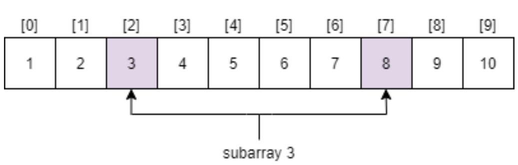
Sort subarray 2:



Compare 7 and 2.

7 > 2, there is no change.

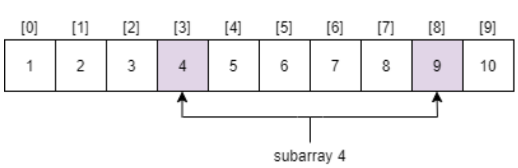
Sort subarray 3:



Compare 8 and 3.

8 > 3, there is no change.

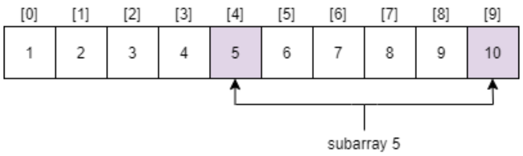
Sort subarray 4:



Compare 9 and 4.

9 > 4, there is no change.

Sort subarray 5:

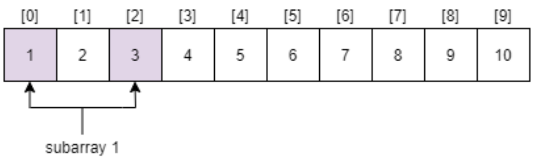


Compare 10 and 5.

10 > 5, there is no change.

gap value = 5 / 2.2 = 2

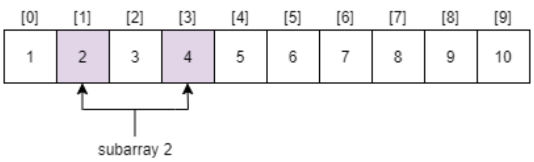
Sort subarray 1:



Compare 3 and 1.

3 > 1, there is no change.

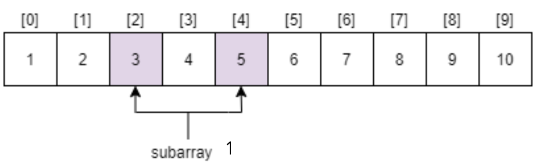
Sort subarray 2:



Compare 4 and 2.

4 > 2, there is no change.

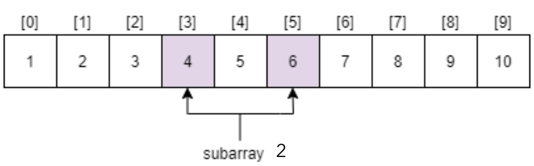
Sort subarray 1:



Compare 5 and 3.

5 > 3, there is no change.

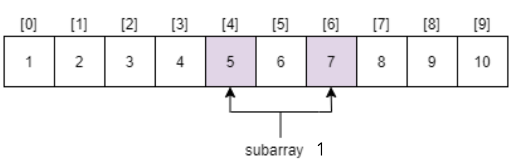
Sort subarray 2:



Compare 6 and 4.

6 > 4, there is no change.

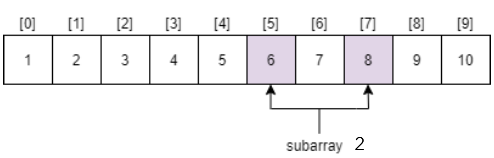
Sort subarray 1:



Compare 7 and 5.

7 > 5, there is no change.

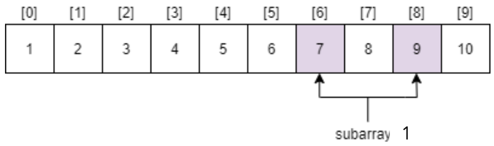
Sort subarray 2:



Compare 8 and 6.

8 > 6, there is no change.

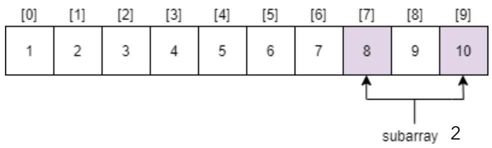
Sort subarray 1:



Compare 9 and 7.

9 > 7, there is no change.

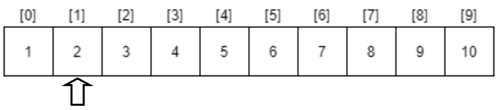
Sort subarray 2:



Compare 10 and 8.

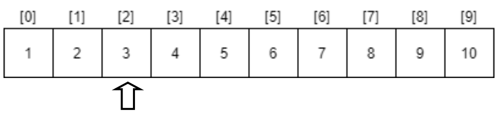
10 > 8, there is no change.

gap value = 1 (a regular insertion sort)



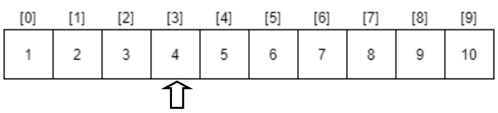
Compare 2 and 1.

2 > 1, there is no change.



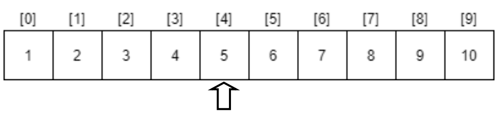
Compare 3 and 2.

3 > 2, there is no change.



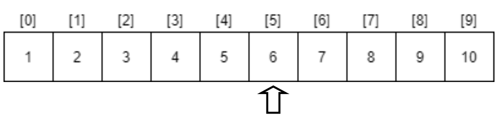
Compare 4 and 3.

4 > 3, there is no change.



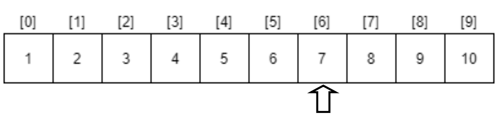
Compare 5 and 4.

5 > 4, there is no change.



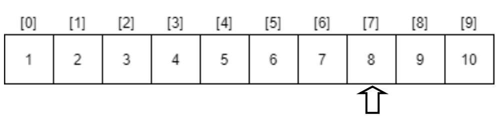
Compare 6 and 5.

6 > 5, there is no change.



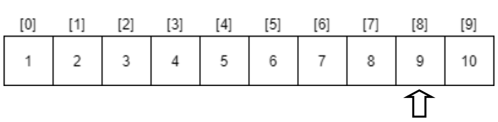
Compare 7 and 6.

7 > 6, there is no change.



Compare 8 and 7.

8 > 7, there is no change.



Compare 9 and 8.

9 > 8, there is no change.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	2	3	4	5	6	7	8	9	10

Compare 10 and 9.

10 > 9, there is no change.

Merge Sort:

1. Step:
   Split table into halves.

1	2	3	4	5	6	7	8	9	10

Left

1	2	3	4	5

Right

6	7	8	9	10

2. Step:
   Recursive call for left table.
   Split left table into halves.

1	2	3	4	5	6	7	8	9	10

Left

1	2	3	4	5

Right

6	7	8	9	10

Left

1	2

Right

3	4	5

3. Step.
   Recursive call for left table.
   Split left table into halves.

1	2	3	4	5	6	7	8	9	10

Left

1	2	3	4	5

Right

6	7	8	9	10

Left

1	2

Right

3	4	5

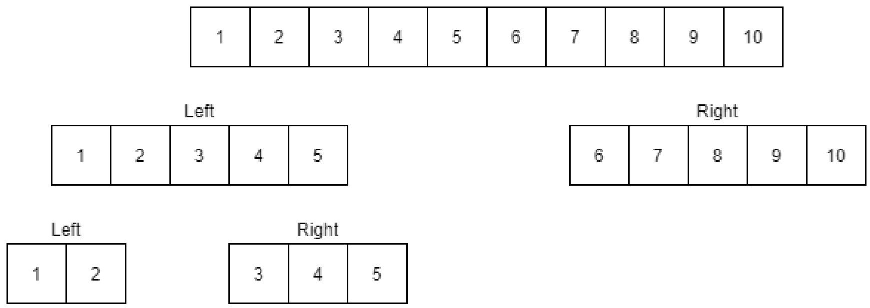
Left

1

Right

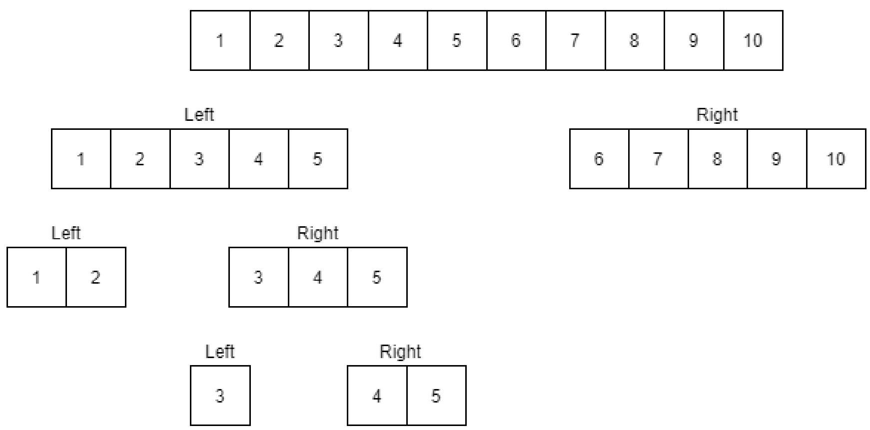
2

4. Step:
   Return left and right table directly because their size is 1. If a table has one element, it is sorted.
   Merge the halves.
   Find the smaller and insert it into the table. Perform this process for each element.



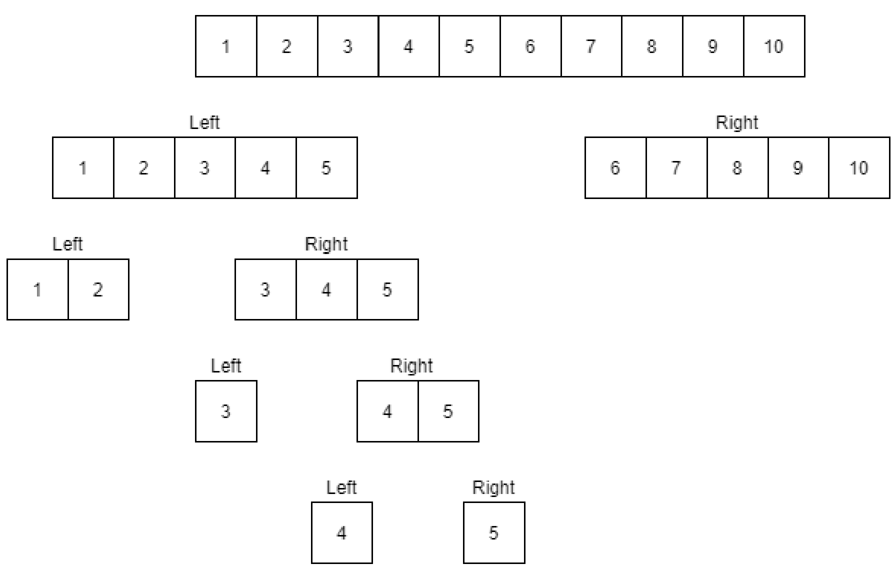
5. Step:
   Recursive call for right table.

   Split right table into halves.

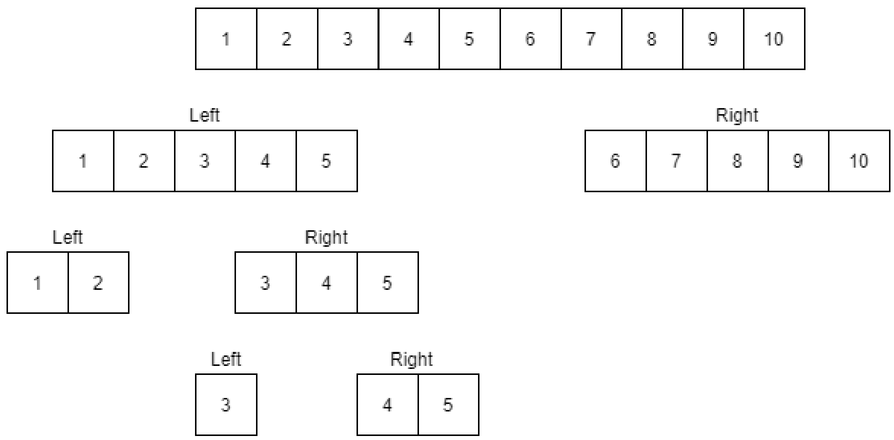


6. Step:

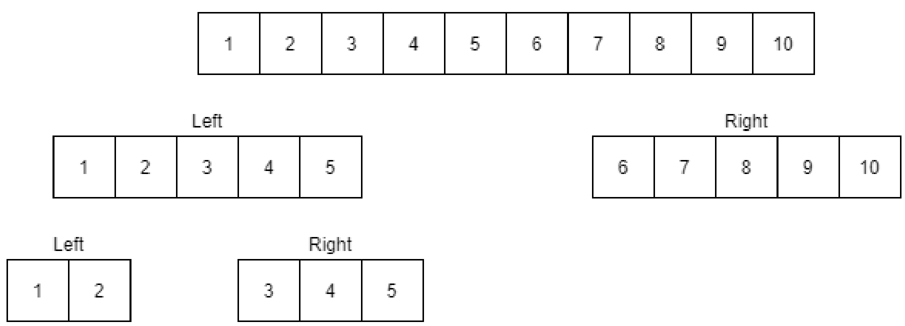
   Return left table, the table has one element, it is sorted.
   Split right table into halves.



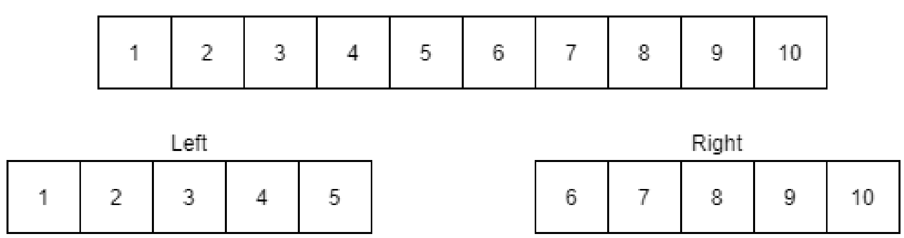
7. Step:
   Merge the halves.
   Find the smaller and insert it into the table. Perform this process for each element.



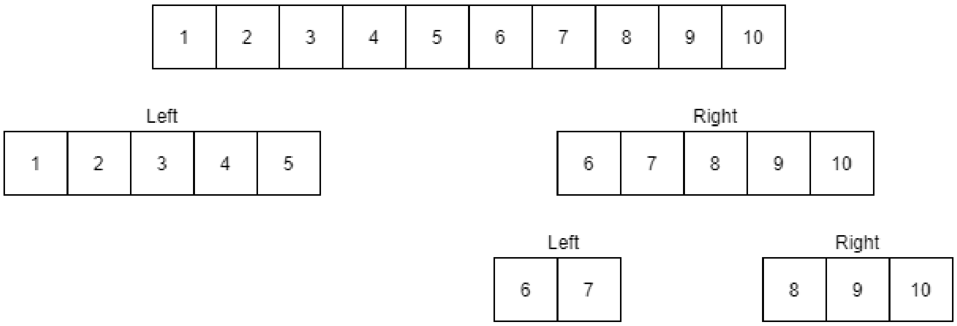
8. Step:
   Merge the halves.
   Find the smaller and insert it into the table. Perform this process for each element (3, 4 , 5).



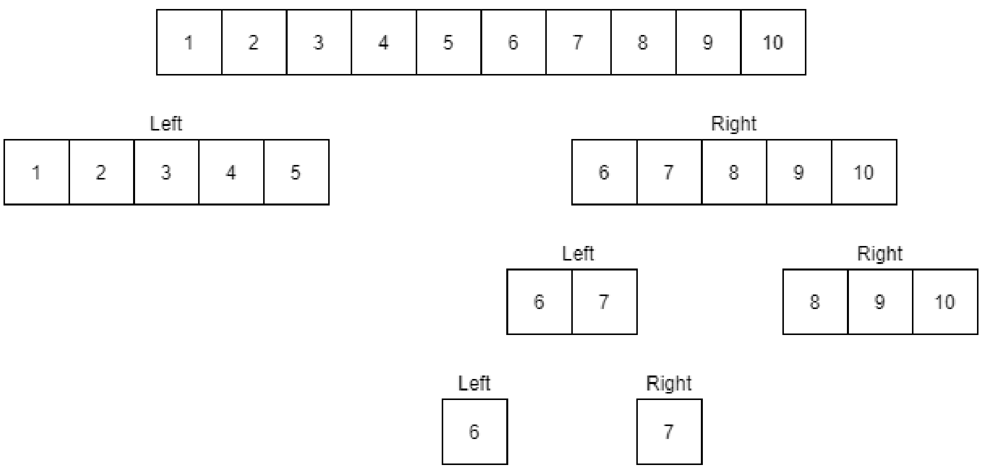
9. Step:
   Merge the halves.
   Find the smaller and insert it into the table. Perform this process for each element (1, 2, 3, 4 , 5).



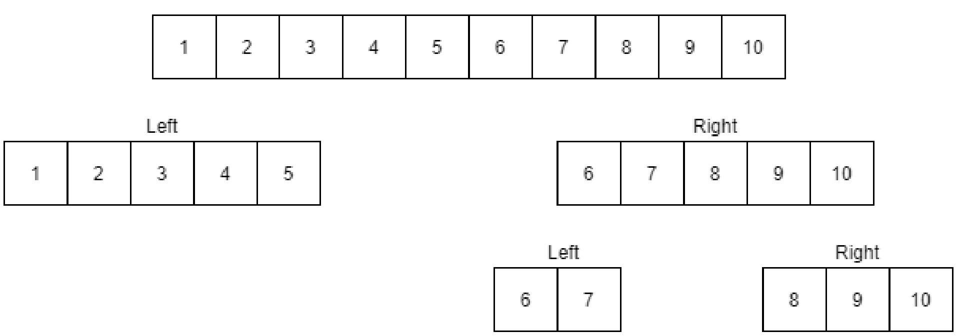
10. Step:
    The left table is returned.
    So, Recursive call for right table.
    Split right table into halves.



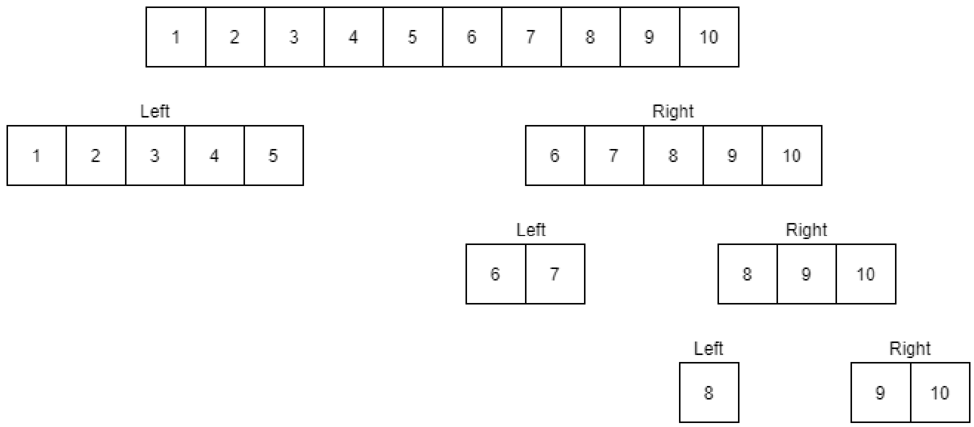
11. Step:
    Recursive call for left table.
    Split left table into halves.



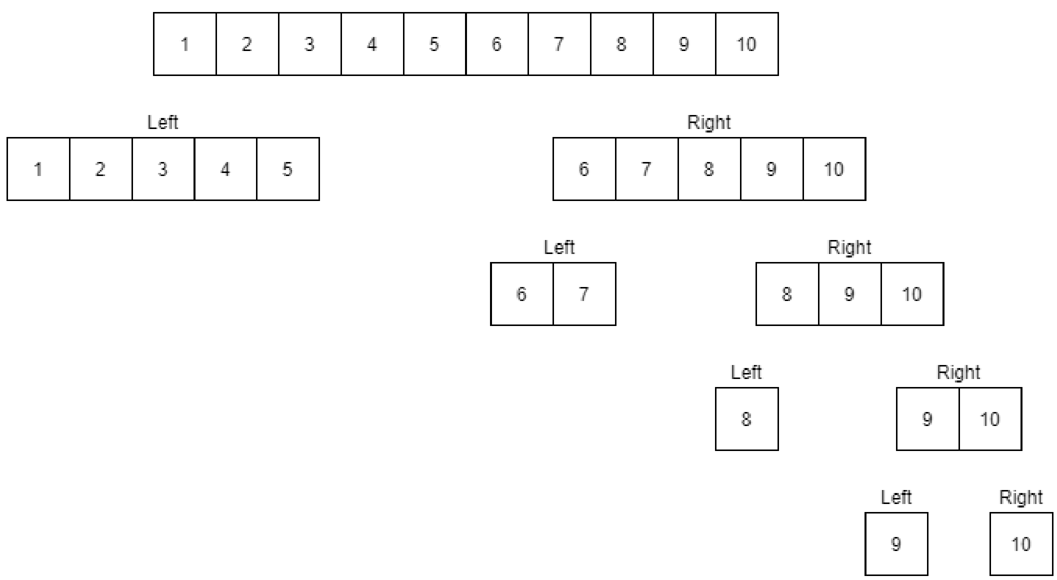
12. Step:
    Return left and right table.
    Merge the halves.
    Find the smaller and insert it into the table. Perform this process for each element. (6 , 7)



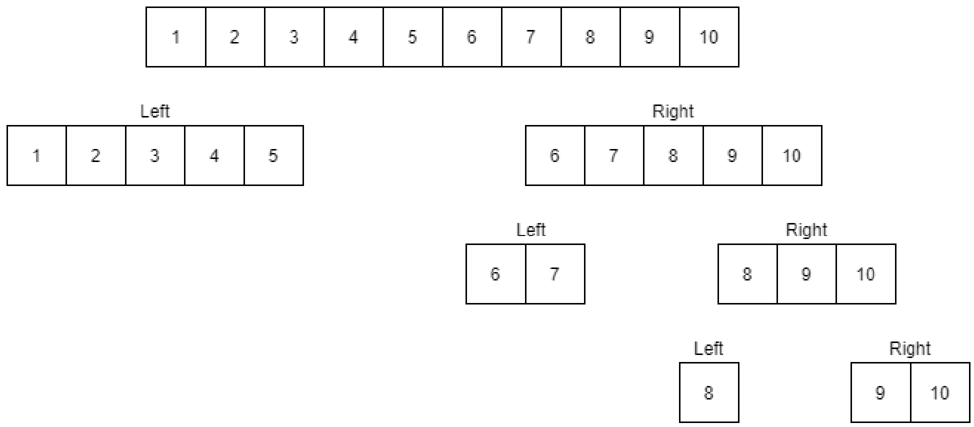
13. Step:
    Return the left table.
    Recursive call for the right table.
    Split right table into halves.



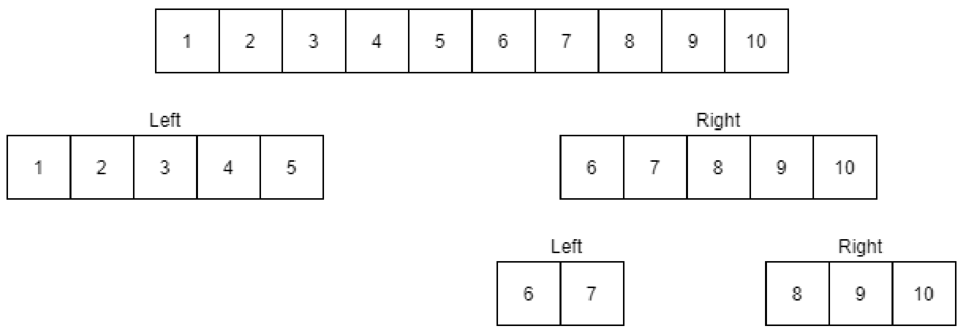
14. Step:
    The left table is returned.
    Recursive call for the right table.
    Split right table into halves.



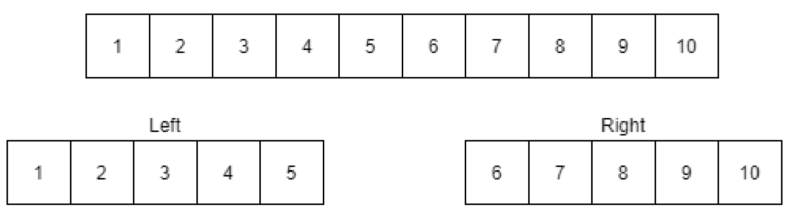
15. Step:
    Return left and right table, and merge these halves.
    Find the smaller and insert it into the table. Perform this process for each element. (9, 10)



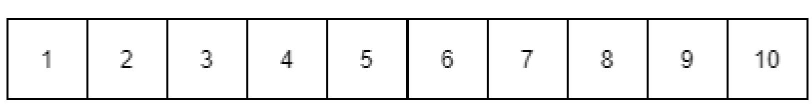
16. Step:
    Merge halves.
    Find the smaller and insert it into the table. Perform this process for each element. (8, 9, 10)



17. Step:
    The left and right tables are returned.
    Merge halves.
    Find the smaller and insert it into the table. Perform this process for each element. (6, 7, 8, 9, 10)



18. Step:
    The left and right tables are returned.
    Merge halves.
    Find the smaller and insert it into the table. Perform this process for each element. (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)



Heap Sort:

1. Step : Build Heap

Symbols:

: is index of parent

: is index of child

Array	Action	Result
1 2 3 4 5 6 7 8 9 10	Compare parent and child (0 and 1) parent < child swap (parent, child)	2 1 3 4 5 6 7 8 9 10
2 1 3 4 5 6 7 8 9 10	Compare parent and child (0 and 2) parent < child swap(parent, child)	3 1 2 4 5 6 7 8 9 10
3 1 2 4 5 6 7 8 9 10	Compare parent and child (1 and 3) parent < child swap(parent, child), child = 1, parent = 0 Compare parent and child parent < child swap(parent, child)	3 4 2 1 5 6 7 8 9 10 4 3 2 1 5 6 7 8 9 10
4 3 2 1 5 6 7 8 9 10	Compare parent and child (1 and 4) parent < child swap(parent, child), child = 1, parent = 0 Compare parent and child parent < child swap(parent, child)	4 5 2 1 3 6 7 8 9 10 5 4 2 1 3 6 7 8 9 10
5 4 2 1 3 6 7 8 9 10	Compare parent and child (2 and 5) parent < child swap(parent, child), child = 2, parent = 0 Compare parent and child parent < child swap(parent, child)	5 4 6 1 3 2 7 8 9 10 6 4 5 1 3 2 7 8 9 10
6 4 5 1 3 2 7 8 9 10	Compare parent and child (2 and 6) parent < child swap(parent, child), child = 2, parent = 0 Compare parent and child parent < child swap(parent, child)	6 4 7 1 3 2 5 8 9 10 7 4 6 1 3 2 5 8 9 10

7 4 6 1 3 2 5 8 9 10	Compare parent and child (3 and 7) parent < child swap(parent, child), child = 3, parent = 1 Compare parent and child parent < child swap(parent, child), child = 1, parent = 0 Compare parent and child parent < child swap(parent, child)	7 4 6 8 3 2 5 1 9 10 7 8 6 4 3 2 5 1 9 10 8 7 6 4 3 2 5 1 9 10
8 7 6 4 3 2 5 1 9 10	Compare parent and child (3 and 8) parent < child swap(parent, child), child = 3, parent = 1 Compare parent and child parent < child swap(parent, child), child = 1, parent = 0 Compare parent and child parent < child swap(parent, child)	8 7 6 9 3 2 5 1 4 10 8 9 6 7 3 2 5 1 4 10 9 8 6 7 3 2 5 1 4 10
9 8 6 7 3 2 5 1 4 10	Compare parent and child (4 and 9) parent < child swap(parent, child), child = 4, parent = 1 Compare parent and child parent < child swap(parent, child), child = 1, parent = 0 Compare parent and child parent < child swap(parent, child)	9 8 6 7 10 2 5 1 4 3 9 10 6 7 8 2 5 1 4 3 10 9 6 7 8 2 5 1 4 3

Heap : 10 9 6 7 8 2 5 1 4 3

2. Step: Shrink Heap

Symbols:

: is index of parent

: is index of max child

n : end of the heap

Array	Action	Result
10 9 6 7 8 2 5 1 4 3	n = 9, swap(0, n)	3 9 6 7 8 2 5 1 4 10
3 9 6 7 8 2 5 1 4 10	Compare parent and max child (0 and 1) parent < max child swap(parent, max child), parent = 1	9 3 6 7 8 2 5 1 4 10

9 3 6 7 8 2 5 1 4 10	Compare parent and max child (1 and 4) parent < max child swap(parent, max child), parent = 4	9 8 6 7 3 2 5 1 4 10
9 8 6 7 3 2 5 1 4 10	n = 8, swap(0, n)	4 8 6 7 3 2 5 1 9 10
4 8 6 7 3 2 5 1 9 10	Compare parent and max child (0 and 1) parent < max child swap(parent, max child), parent = 1	8 4 6 7 3 2 5 1 9 10
8 4 6 7 3 2 5 1 9 10	Compare parent and max child (1 and 3) parent < max child swap(parent, max child), parent = 3	8 7 6 4 3 2 5 1 9 10
8 7 6 4 3 2 5 1 9 10	Compare parent and max child (3 and 7) parent > max child	8 7 6 4 3 2 5 1 9 10
8 7 6 4 3 2 5 1 9 10	n = 7, swap(0, n)	1 7 6 4 3 2 5 8 9 10
1 7 6 4 3 2 5 8 9 10	Compare parent and max child (0 and 1) parent < max child swap(parent, max child), parent = 1	7 1 6 4 3 2 5 8 9 10
7 1 6 4 3 2 5 8 9 10	Compare parent and max child (1 and 3) parent < max child swap(parent, max child), parent = 3	7 4 6 1 3 2 5 8 9 10
7 4 6 1 3 2 5 8 9 10	n = 6, swap(0, n)	5 4 6 1 3 2 7 8 9 10
5 4 6 1 3 2 7 8 9 10	Compare parent and max child (0 and 2) parent < max child swap(parent, max child), parent = 2	6 4 5 1 3 2 7 8 9 10
6 4 5 1 3 2 7 8 9 10	Compare parent and max child (2 and 5) parent > max child	6 4 5 1 3 2 7 8 9 10
6 4 5 1 3 2 7 8 9 10	n = 5, swap(0, n)	2 4 5 1 3 6 7 8 9 10
2 4 5 1 3 6 7 8 9 10	Compare parent and max child (0 and 2) parent < max child swap(parent, max child), parent = 2	5 4 2 1 3 6 7 8 9 10
5 4 2 1 3 6 7 8 9 10	n = 4, swap(0, n)	3 4 2 1 5 6 7 8 9 10
3 4 2 1 5 6 7 8 9 10	Compare parent and max child (0 and 1) parent < max child swap(parent, max child), parent = 1	4 3 2 1 5 6 7 8 9 10
4 3 2 1 5 6 7 8 9 10	Compare parent and max child (1 and 3) parent > max child	4 3 2 1 5 6 7 8 9 10
4 3 2 1 5 6 7 8 9 10	n = 3, swap(0, n)	1 3 2 4 5 6 7 8 9 10

1 3 2 4 5 6 7 8 9 10	Compare parent and max child (0 and 1) parent < max child swap(parent, max child), parent = 1	3 1 2 4 5 6 7 8 9 10
3 1 2 4 5 6 7 8 9 10	n = 2, swap(0, n)	2 1 3 4 5 6 7 8 9 10
2 1 3 4 5 6 7 8 9 10	Compare parent and max child (0 and 1) parent < max child swap(parent, max child), parent = 1	1 2 3 4 5 6 7 8 9 10

Quicksort:

Symbols:
: is up

: is down

Array	Action	Result
1 2 3 4 5 6 7 8 9 10	Put the median of table[first], table[middle], table[last] into table[first], and use this value as the pivot.	6 2 3 4 5 1 7 8 9 10
6 2 3 4 5 1 7 8 9 10	Compare pivot and up, pivot >= up, up++	6 2 3 4 5 1 7 8 9 10
6 2 3 4 5 1 7 8 9 10	Compare pivot and up, pivot >= up, up++	6 2 3 4 5 1 7 8 9 10
6 2 3 4 5 1 7 8 9 10	Compare pivot and up, pivot >= up, up++	6 2 3 4 5 1 7 8 9 10
6 2 3 4 5 1 7 8 9 10	Compare pivot and up, pivot >= up, up++	6 2 3 4 5 1 7 8 9 10
6 2 3 4 5 1 7 8 9 10	Compare pivot and up, pivot >= up, up++	6 2 3 4 5 1 7 8 9 10
6 2 3 4 5 1 7 8 9 10	Compare pivot and up, pivot >= up, up++	6 2 3 4 5 1 7 8 9 10
6 2 3 4 5 1 7 8 9 10	Compare pivot and up, pivot < up. Compare pivot and down, pivot < down, down--	6 2 3 4 5 1 7 8 9 10
6 2 3 4 5 1 7 8 9 10	Compare pivot and down, pivot < down, down--	6 2 3 4 5 1 7 8 9 10
6 2 3 4 5 1 7 8 9 10	Compare pivot and down, pivot < down, down--	6 2 3 4 5 1 7 8 9 10
6 2 3 4 5 1 7 8 9 10	Compare pivot and down, pivot > down, down--	6 2 3 4 5 1 7 8 9 10

6 2 3 4 5 1 7 8 9 10	Compare pivot and down, pivot > down.	6 2 3 4 5 1 7 8 9 10
6 2 3 4 5 1 7 8 9 10	Swap(first, down) Return down	1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10	Recursive call for left table	1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10	Put the median of table[first], table[middle], table[last] into table[first], and use this value as the pivot.	3 2 1 4 5 6 7 8 9 10
3 2 1 4 5 6 7 8 9 10	Compare pivot and up, pivot >= up, up++	3 2 1 4 5 6 7 8 9 10
3 2 1 4 5 6 7 8 9 10	Compare pivot and up, pivot >= up, up++	3 2 1 4 5 6 7 8 9 10
3 2 1 4 5 6 7 8 9 10	Compare pivot and up, pivot >= up, up++	3 2 1 4 5 6 7 8 9 10
3 2 1 4 5 6 7 8 9 10	Compare pivot and up, pivot < up. Compare pivot and down, pivot < down, down--	3 2 1 4 5 6 7 8 9 10
3 2 1 4 5 6 7 8 9 10	Compare pivot and down, pivot < down, down--	3 2 1 4 5 6 7 8 9 10
3 2 1 4 5 6 7 8 9 10	Compare pivot and down, pivot > down	3 2 1 4 5 6 7 8 9 10
3 2 1 4 5 6 7 8 9 10	Swap(first, down) Return down	1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10	Recursive call for left table (1 2)	1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10	Put the median of table[first], table[middle], table[last] into table[first], and use this value as the pivot.	1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10	Compare pivot and up, pivot >= up, up++	1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10	Compare pivot and up, pivot < up. Compare pivot and down, pivot < down, down--	1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10	Compare pivot and down, pivot = down. Swap(first, down) Return down	1 2 3 4 5 6 7 8 9 10

1 2 3 4 5 6 7 8 9 10	Return this part. (1 2) Recursive call for the right table (3 4)	1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10	Put the median of table[first], table[middle], table[last] into table[first], and use this value as the pivot.	1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10	Compare pivot and up, pivot >= up, up++	1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10	Compare pivot and up, pivot < up. Compare pivot and down, pivot < down, down--	1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10	Compare pivot and down, pivot = down. Swap(first, down) Return down	1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10	Return this part. (4 5)	1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10	Return left table (1 2 3 4 5) Recursive call right table (7 8 9 10)	1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10	Put the median of table[first], table[middle], table[last] into table[first], and use this value as the pivot.	1 2 3 4 5 6 8 7 9 10
1 2 3 4 5 6 8 7 9 10	Compare pivot and up, pivot >= up, up++	1 2 3 4 5 6 8 7 9 10
1 2 3 4 5 6 8 7 9 10	Compare pivot and up, pivot >= up, up++	1 2 3 4 5 6 8 7 9 10
1 2 3 4 5 6 8 7 9 10	Compare pivot and up, pivot <up. Compare pivot and down, pivot < down	1 2 3 4 5 6 8 7 9 10
1 2 3 4 5 6 8 7 9 10	Compare pivot and down, pivot < down	1 2 3 4 5 6 8 7 9 10
1 2 3 4 5 6 8 7 9 10	Compare pivot and down, pivot > down	1 2 3 4 5 6 8 7 9 10
1 2 3 4 5 6 8 7 9 10	Swap(first, down) Return down	1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10	Recursive call for the left part (7) Return left part. Recursive call for the right part.(9 10)	1 2 3 4 5 6 7 8 9 10

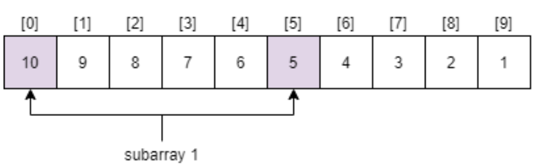
1 2 3 4 5 6 7 8 9 10	Put the median of table[first], table[middle], table[last] into table[first], and use this value as the pivot.	1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10	Compare pivot and up, pivot >= up, up++	1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10	Compare pivot and up, pivot < up. Compare pivot and down, pivot < down, down --	1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10	Compare pivot and down, pivot = down. Swap(first, down) Return down	1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10	Return this part. (4 5)	1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10	Return right table (7 8 9 10)	1 2 3 4 5 6 7 8 9 10

b)

Shell Sort:

gap value = A.length / 2 = 5

Sort subarray 1:

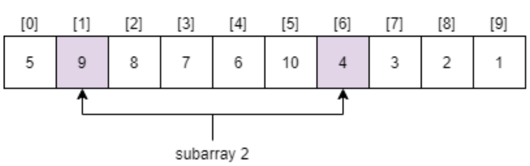


Compare 5 and 10.

5 < 10, table[5] = 10,

table[0] = 5

Sort subarray 2:

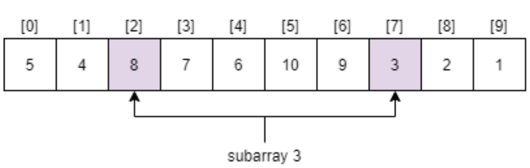


Compare 4 and 9.

4 < 9, table[6] = 9,

table[1] = 4

Sort subarray 3:

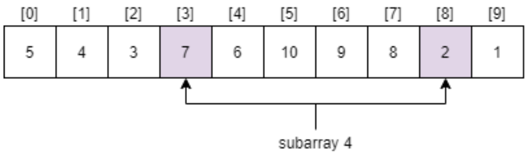


Compare 3 and 8.

3 < 8, table[7] = 8,

table[2] = 3

Sort subarray 4:

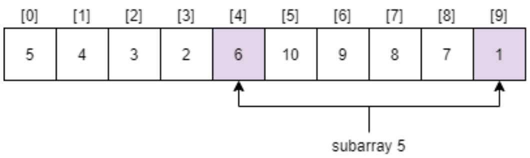


Compare 2 and 7.

2 < 7, table[8] = 7,

table[3] = 2

Sort subarray 5:



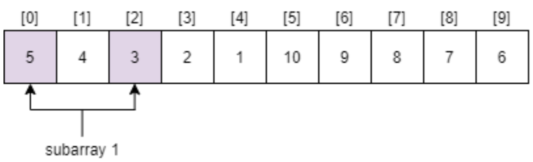
Compare 1 and 6.

1 < 6, table[9] = 6,

table[4] = 1

gap value = 5 / 2.2 = 2

Sort subarray 1:

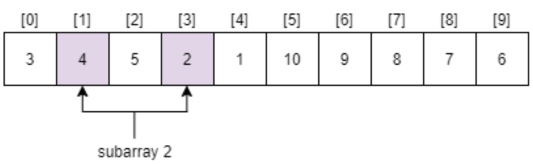


Compare 3 and 5.

3 < 5, table[2] = 5,

table[0] = 3

Sort subarray 2:

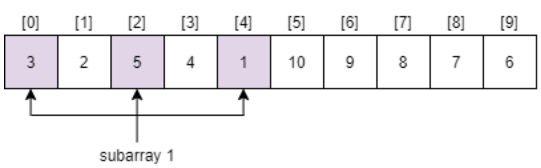


Compare 2 and 4.

2 < 4, table[3] = 4,

table[1] = 2

Sort subarray 1:

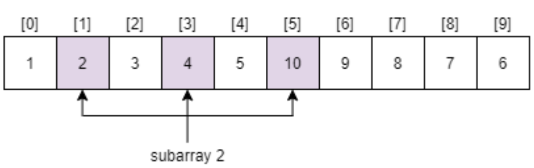


Compare 1 and 5, 1 < 5 , table[4] = 5,

Compare 1 and 3, 1 < 3, table[2] = 3,

table[0] = 1

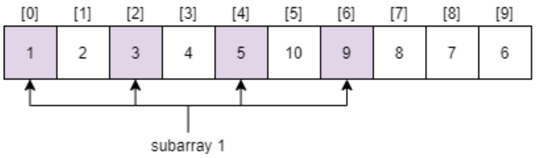
Sort subarray 2:



Compare 10 and 4, 10 > 4,

There is no change.

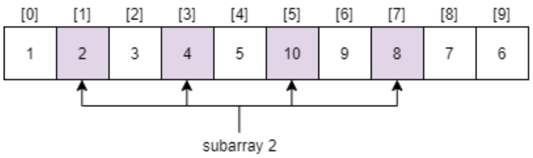
Sort subarray 1:



Compare 9 and 5, 9 >5,

There is no change.

Sort subarray 2:

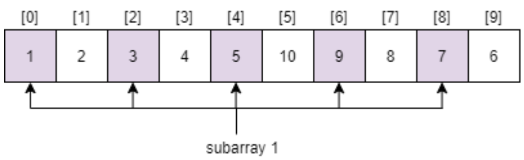


Compare 8 and 10, 8 < 10, table[7] = 10,

Compare 8 and 4, 8 > 4,

table[5] = 8

Sort subarray 1:

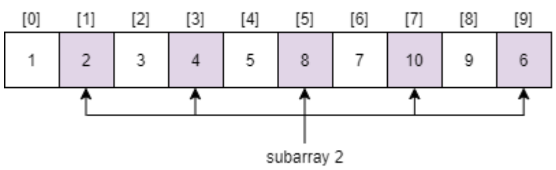


Compare 7 and 9, 7 < 9, table[8] = 9,

Compare 7 and 5, 7 > 5,

table[6] = 7.

Sort subarray 2:



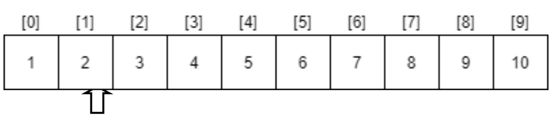
Compare 6 and 10, 6 < 10, table[9] = 10,

Compare 6 and 8, 6 < 8, table[7] = 8,

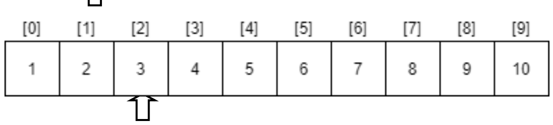
Compare 6 and 4, 6 > 4,

table[5] = 6

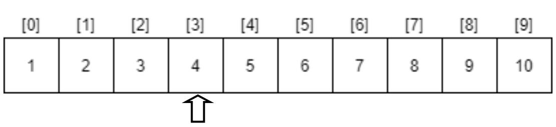
gap value = 1 (a regular insertion sort)



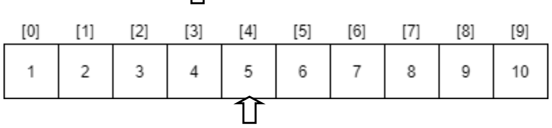
Compare 2 and 1.
2 > 1, there is no change.



Compare 3 and 2.
3 > 2, there is no change.

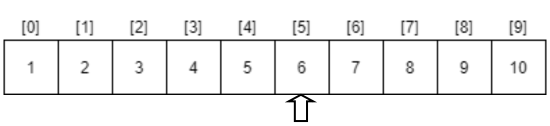


Compare 4 and 3.
4 > 3, there is no change.



Compare 5 and 4.

5 > 4, there is no change.



Compare 6 and 5.

6 > 5, there is no change.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	2	3	4	5	6	7	8	9	10

Compare 7 and 6.

7 > 6, there is no change.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	2	3	4	5	6	7	8	9	10

Compare 8 and 7.

8 > 7, there is no change.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	2	3	4	5	6	7	8	9	10

Compare 9 and 8.

9 > 8, there is no change.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	2	3	4	5	6	7	8	9	10

Compare 10 and 9.

10 > 9, there is no change.

Merge Sort:

1. Step:
   Split table into halves.

10	9	8	7	6	5	4	3	2	1

Left

10	9	8	7	6

Right

5	4	3	2	1

2. Step:
   Recursive call for left table.
   Split left table into halves.

10	9	8	7	6	5	4	3	2	1

Left

10	9	8	7	6

Right

5	4	3	2	1

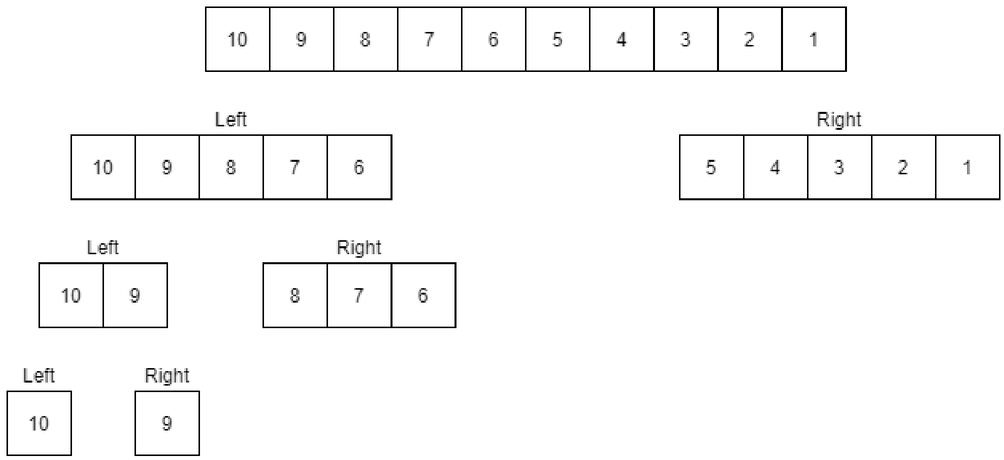
Left

10	9

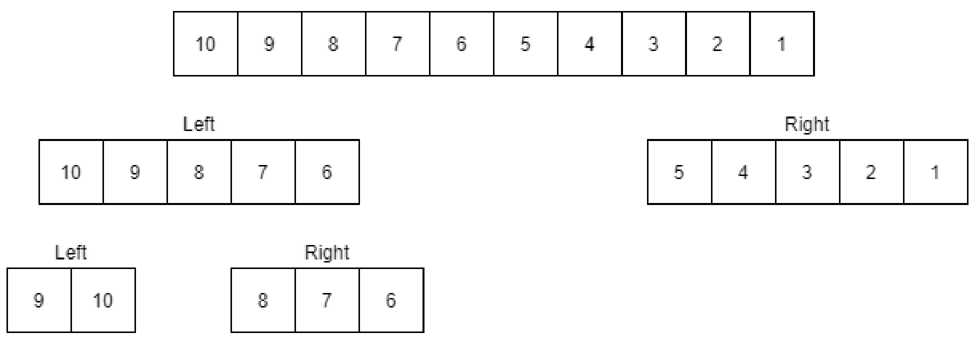
Right

8	7	6

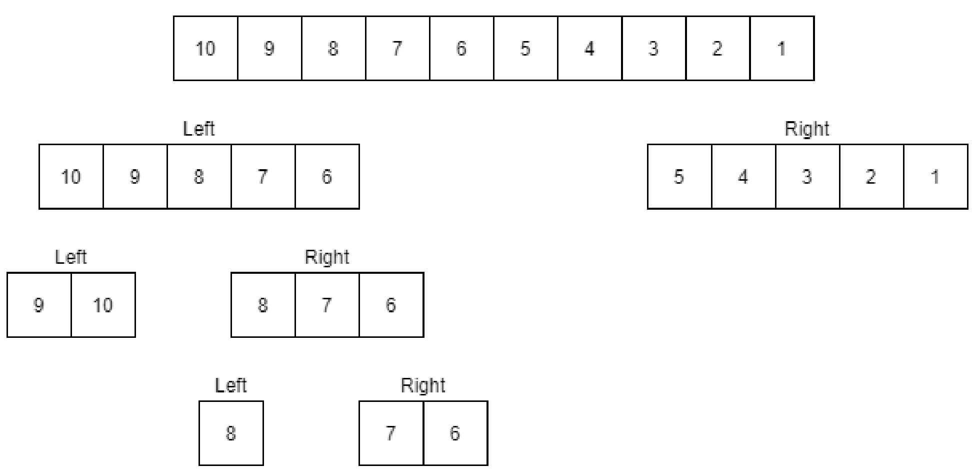
3. Step:
   Recursive call for left table.
   Split left table into halves.



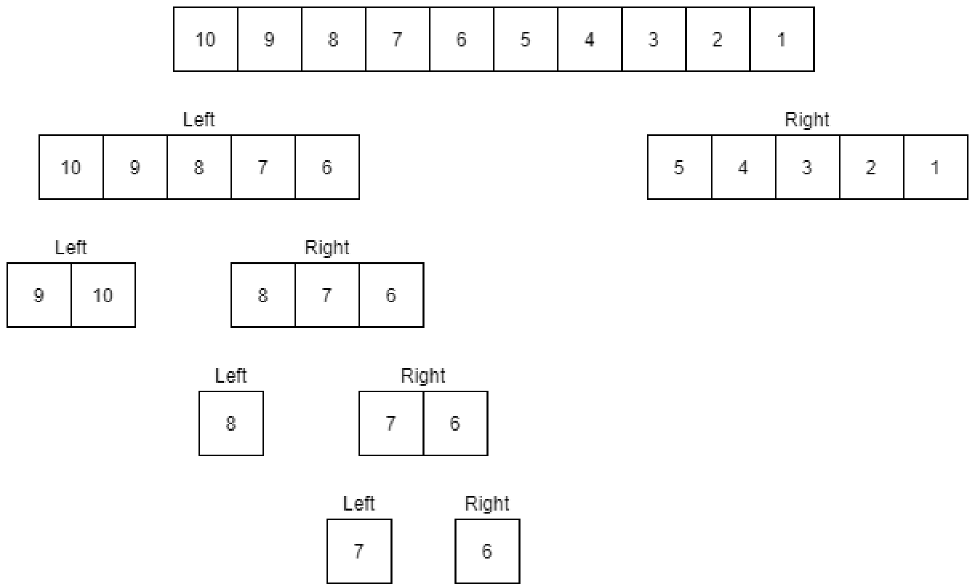
4. Step:
   Return left and right table directly because their size is 1. If a table has one element, it is sorted.
   Merge the halves.
   Find the smaller and insert it into the table. Perform this process for each element. (10 , 9)



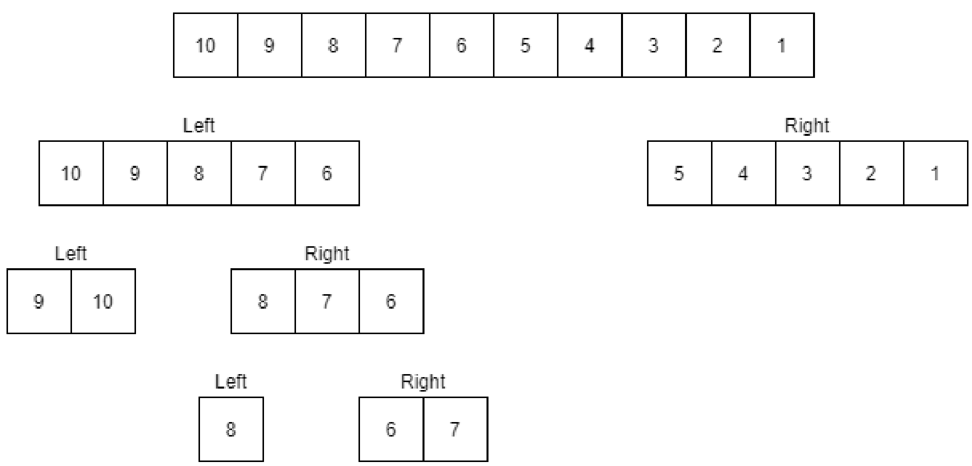
5. Step:
   Return Left table.
   Recursive call for the right table.
   Split right table into halves.



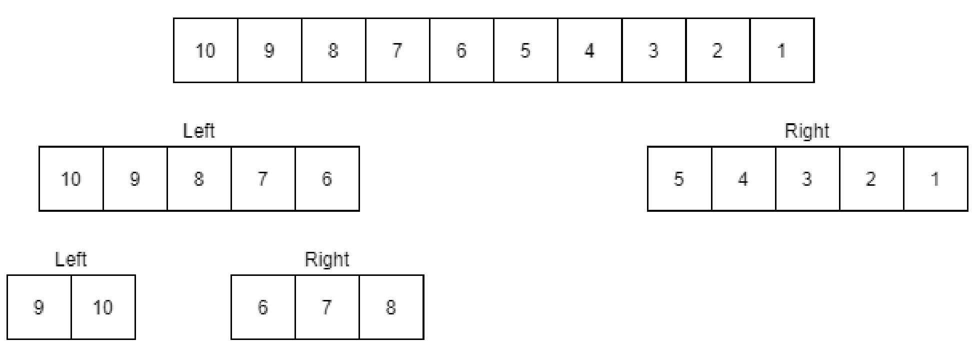
6. Step:
   Return left table, the table has one element, it is sorted.
   Recursive call for right table.
   Split right table into halves.



7. Step:
   Return left and right table.
   Merge the halves.
   Find the smaller and insert it into the table. Perform this process for each element. (7 , 6)



8. Step:
   Return right table.
   Merge the halves.
   Find the smaller and insert it into the table. Perform this process for each element. (8, 6, 7)



9. Step:
   Merge the halves.
   Find the smaller and insert it into the table. Perform this process for each element. (9, 10, 6, 7, 8)

10	9	8	7	6	5	4	3	2	1

Left

6	7	8	9	10

Right

5	4	3	2	1

10. Step:
    Return left tables.
    Recursive call for right table.
    Split right table into halves.

10	9	8	7	6	5	4	3	2	1

Left

6	7	8	9	10

Right

5	4	3	2	1

Left

5	4

Right

3	2	1

11. Step:
    Recursive call for left table.
    Split left table into halves.

10	9	8	7	6	5	4	3	2	1

Left

6	7	8	9	10

Right

5	4	3	2	1

Left

5	4

Right

3	2	1

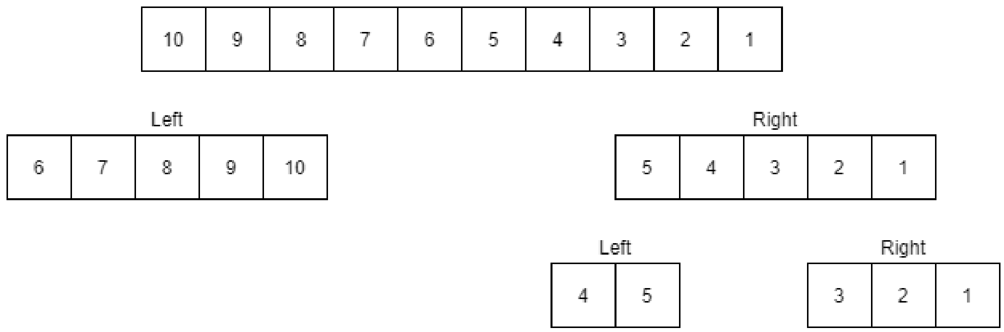
Left

5

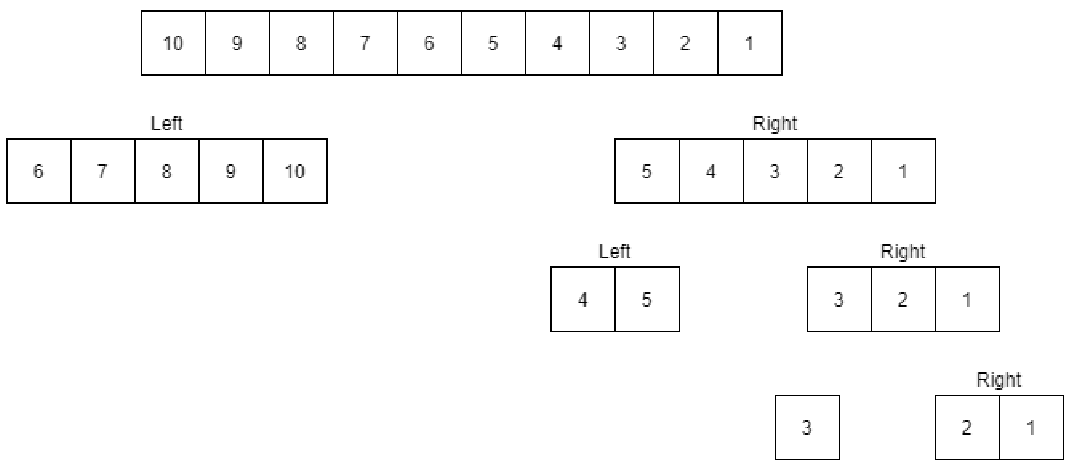
Right

4

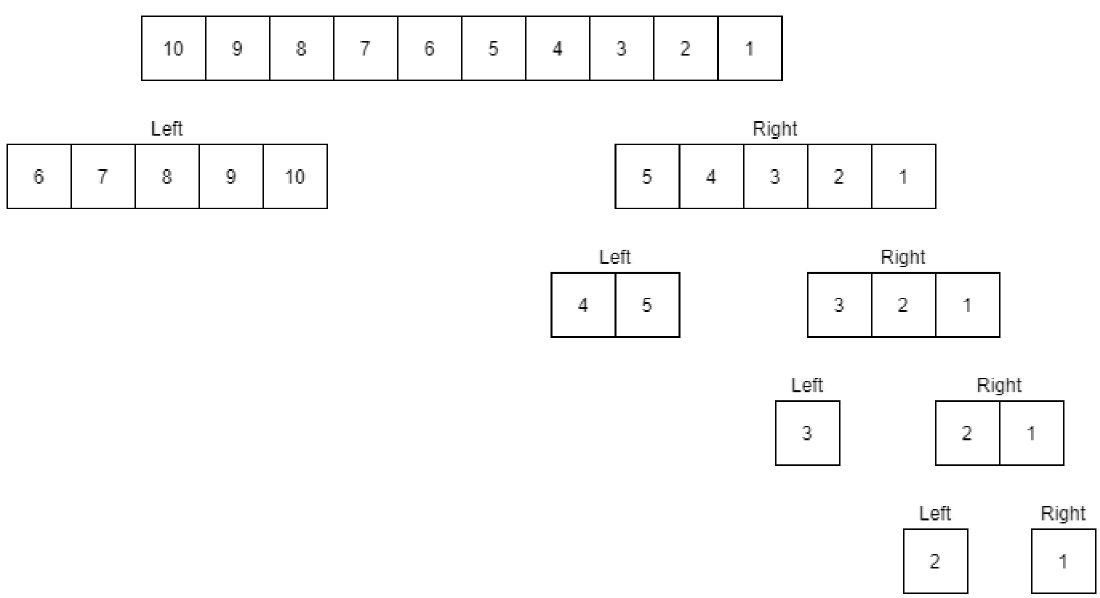
12. Step:
    Return left and right table.
    Merge the halves.
    Find the smaller and insert it into the table. Perform this process for each element. (5 , 4)



13. Step:
    Return left table.
    Recursive call for right table.
    Split rigt table into halves.



14. Step:
    Return left table.
    Recursive call for right table.
    Split rigt table into halves.



15. Step:
    Recursive call left and right table.
    Merge the halves.
    Find the smaller and insert it into the table. Perform this process for each element. (2, 1)

10	9	8	7	6	5	4	3	2	1

Left

6	7	8	9	10

Right

5	4	3	2	1

Left

4	5

Right

3	2	1

Left

3

Right

1	2

16. Step:
    Return right table.
    Merge the halves.
    Find the smaller and insert it into the table. Perform this process for each element. (3, 1, 2)

10	9	8	7	6	5	4	3	2	1

Left

6	7	8	9	10

Right

5	4	3	2	1

Left

4	5

Right

1	2	3

17. Step:
    Return right table.
    Merge the halves.
    Find the smaller and insert it into the table. Perform this process for each element. (4, 5, 1, 2 ,3)

10	9	8	7	6	5	4	3	2	1

Left

6	7	8	9	10

Right

1	2	3	4	5

18. Step:
    Return right table.
    Merge the halves.
    Find the smaller and insert it into the table. Perform this process for each element. (6, 7, 8, 9, 10, 1, 2, 3, 4, 5)

1	2	3	4	6	5	7	8	9	10