GIT Department of Computer Engineering

CSE 222/505 - Spring 2020

Homework #6 Question 1

Emirhan Uzun

171044019

Sorting 1.a Array A (Integer Array From Small To Large-10 Elements) – Shell Sort Algorithm

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

I will use the Shell sort algorithm in the book.

```
public <T extends Comparable<T>> void sort(T[] table) {
        int gap = table.length / 2;
        while (gap > 0) {
                for (int nextPos = gap; nextPos < table.length;nextPos++) {
                        insert(table, nextPos, gap);
                }
                if (gap == 2)  gap = 1;
                else gap = (int) (gap / 2.2);
        }
}

private static <T extends Comparable<T>> void insert(T[] table,int nextPos,int gap) {
        T nextVal = table[nextPos];
        while ((nextPos > gap - 1) && (nextVal.compareTo(table[nextPos - gap]) < 0)) {
                table[nextPos] = table[nextPos - gap];
                nextPos -= gap;
        }
        table[nextPos] = nextVal;
}
```

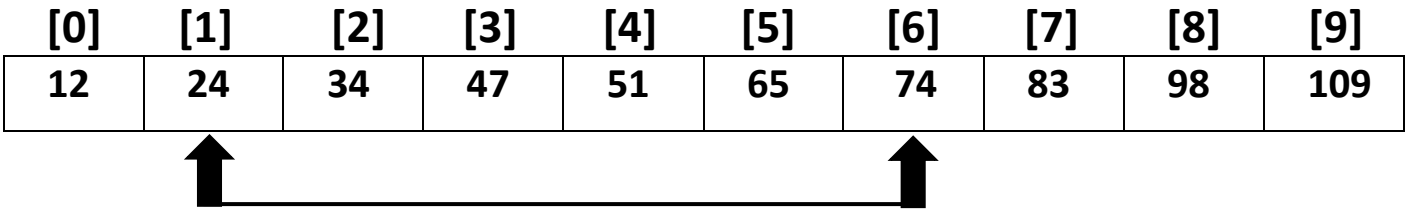
Step 1 : The gap is array length / 2 which is 5. NextPos = gap.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

Gap = 5
nextPos = 5
insert (nextVal = table[5] = 65)
while(5 > 4 && 65 < 12) False X
table[5] = 65

Comparison = 1
Displacement = 0

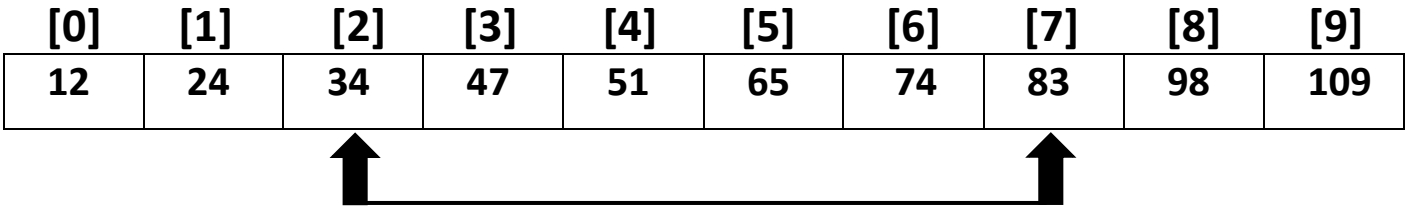
Step 2 : Next Pos is 6



Gap = 5
nextPos = 6
insert (nextVal = table[6] = 74)
while(6 > 4 && 74 < 24) False X
table[6] = 74

Comparison = 2
Displacement = 0

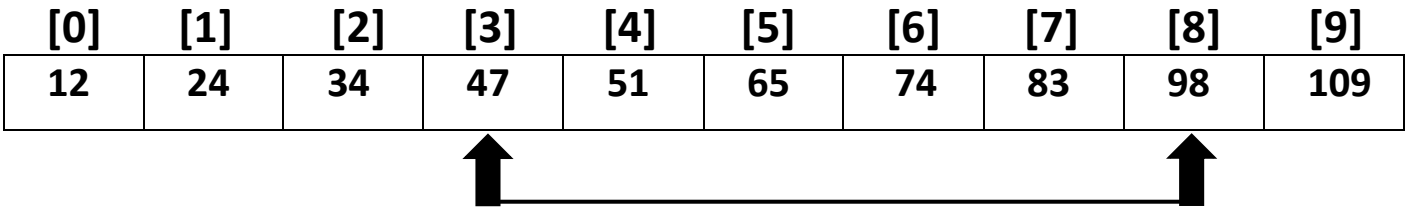
Step 3 : Next Pos is 7



Gap = 5
nextPos = 7
insert (nextVal = table[7] = 83)
while(7 > 4 && 83 < 34) False X
table[7] = 83

Comparison = 3
Displacement = 0

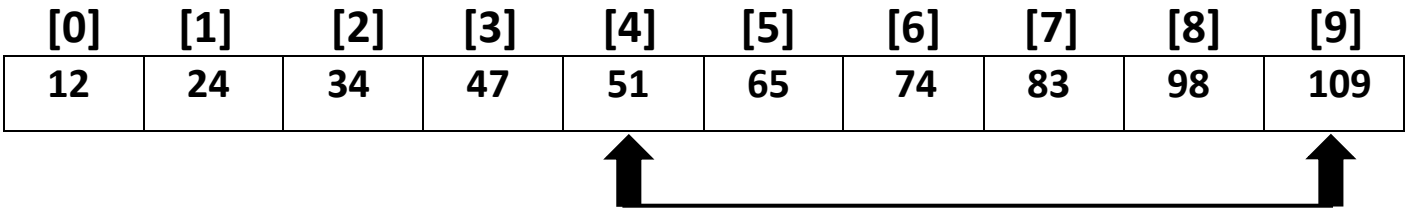
Step 4 : Next Pos is 8



Gap = 5
nextPos = 8
insert (nextVal = table[8] = 98)
while(8 > 4 && 98 < 47) False X
table[8] = 98

Comparison = 4
Displacement = 0

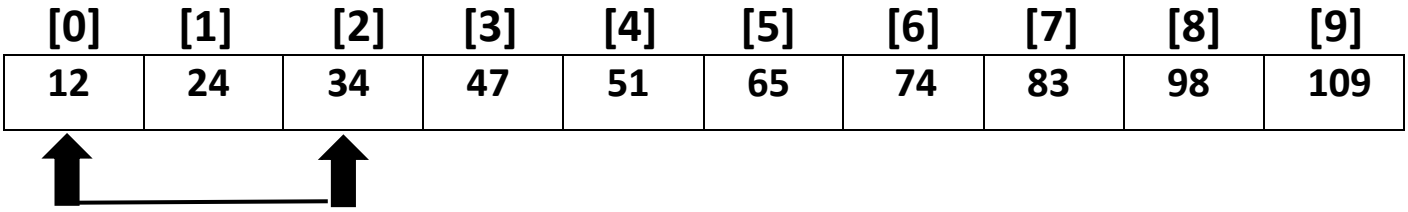
Step 5 : Next Pos is 9



Gap = 5
nextPos = 9
insert (nextVal = table[9] = 109)
while(9 > 4 && 109 < 51) False X
table[9] = 109

Comparison = 5
Displacement = 0

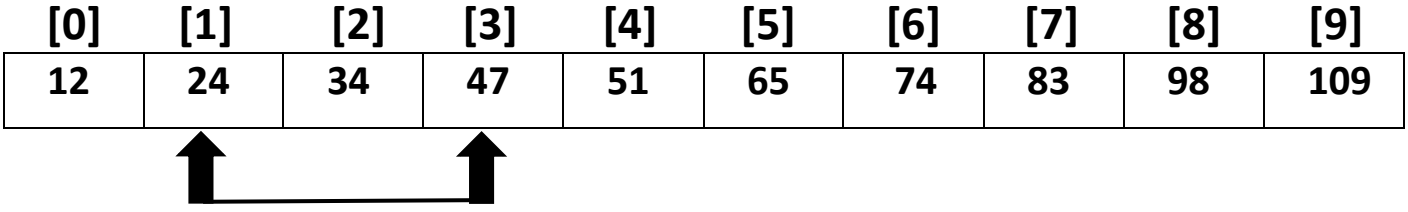
Step 6: The for loop is done and gap is now 5/2.2 = 2



Gap = 2
nextPos = 2
insert (nextVal = table[2] = 34)
while(2 > 1 && 34 < 12) False X
table[2] = 34

Comparison = 6
Displacement = 0

Step 7: Next Pos is 3



Gap = 2
nextPos = 3
insert (nextVal = table[3] = 47)
while(3 > 1 && 47 < 24) False X
table[3] = 47

Comparison = 7
Displacement = 0

Step 8: Next Pos is 4

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

Gap = 2
nextPos = 4
insert (nextVal = table[4] = 51)
while(4 > 1 && 51 < 34) False X
table[4] = 51

Comparison = 8
Displacement = 0

Step 9: Next Pos is 5

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

Gap = 2
nextPos = 5
insert (nextVal = table[5] = 65)
while(5 > 1 && 65 < 47) False X
table[5] = 65

Comparison = 9
Displacement = 0

Step 10: Next Pos is 6

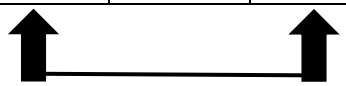
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

Gap = 2
nextPos = 6
insert (nextVal = table[6] = 74)
while(6 > 1 && 74 < 51) False X
table[6] = 74

Comparison = 10
Displacement = 0

Step 11: Next Pos is 7

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109



Gap = 2
nextPos = 7
insert (nextVal = table[7] = 83)
while(7 > 1 && 83 < 65) False X
table[7] = 83

Comparison = 11
Displacement = 0

Step 12: Next Pos is 8

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

Gap = 2
nextPos = 8
insert (nextVal = table[8] = 98)
while(8 > 1 && 98 < 74) False X
table[8] = 98

Comparison = 12
Displacement = 0

Step 13: Next Pos is 9

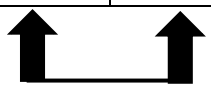
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

Gap = 2
nextPos = 9
insert (nextVal = table[9] = 109)
while(9 > 1 && 109 < 83) False X
table[9] = 109

Comparison = 13
Displacement = 0

Step 14: Now the for loop is done and if gap== 2 then gap = 1.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

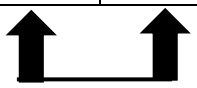


Gap = 1
nextPos = 1
insert (nextVal = table[1] = 24)
while(1 > 0 && 24 < 12) False X
table[1] = 24

Comparison = 14
Displacement = 0

Step 15: NextPos = 2 in for loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

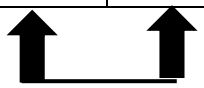


Gap = 1
nextPos = 2
insert (nextVal = table[2] = 34)
while(2 > 0 && 34 < 24) False X
table[2] = 34

Comparison = 15
Displacement = 0

Step 16: NextPos = 3 in for loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

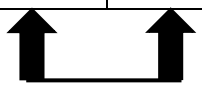


Gap = 1
nextPos = 3
insert (nextVal = table[3] = 47)
while(3 > 0 && 47 < 34) False X
table[3] = 47

Comparison = 16
Displacement = 0

Step 17: NextPos = 4 in for loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

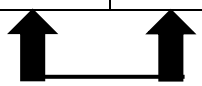


Gap = 1
nextPos = 4
insert (nextVal = table[4] = 51)
while(4 > 0 && 51 < 47) False X
table[4] = 51

Comparison = 17
Displacement = 0

Step 18: NextPos = 5 in for loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

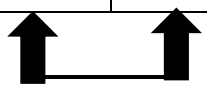


Gap = 1
nextPos = 5
insert (nextVal = table[5] = 65)
while(5 > 0 && 65 < 51) False X
table[5] = 65

Comparison = 18
Displacement = 0

Step 19: NextPos = 6 in for loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

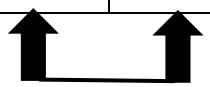


Gap = 1
nextPos = 6
insert (nextVal = table[6] = 74)
while(6 > 0 && 74 < 65) False X
table[6] = 74

Comparison = 19
Displacement = 0

Step 20: NextPos = 7 in for loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

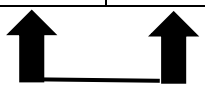


Gap = 1
nextPos = 7
insert (nextVal = table[7] = 83)
while(7 > 0 && 83 < 74) False X
table[7] = 83

Comparison = 20
Displacement = 0

Step 21: NextPos = 8 in for loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

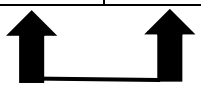


Gap = 1
nextPos = 8
insert (nextVal = table[8] = 98)
while(8 > 0 && 98 < 83) False X
table[8] = 98

Comparison = 21
Displacement = 0

Step 22: NextPos = 9 in for loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109



Gap = 1
nextPos = 9
insert (nextVal = table[9] = 109)
while(9 > 0 && 109 < 98) False X
table[9] = 109

Comparison = 22
Displacement = 0

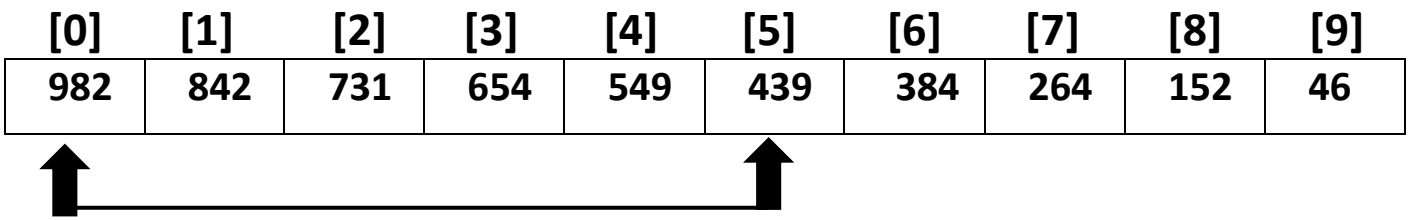
Step 23: Next Pos is 10 and loop is done. So the sorting is done. The last counters and sorted array is below :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

Comparison = 22
Displacement = 0

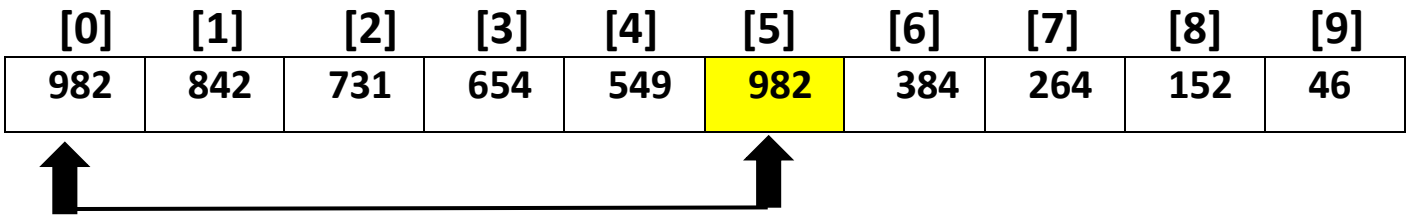
Sorting 1.b Array B (Integer Array From Large To Small-10 Elements) – Shell Sort Algorithm

Step 1 : The gap is array length / 2 which is 5. NextPos = gap.

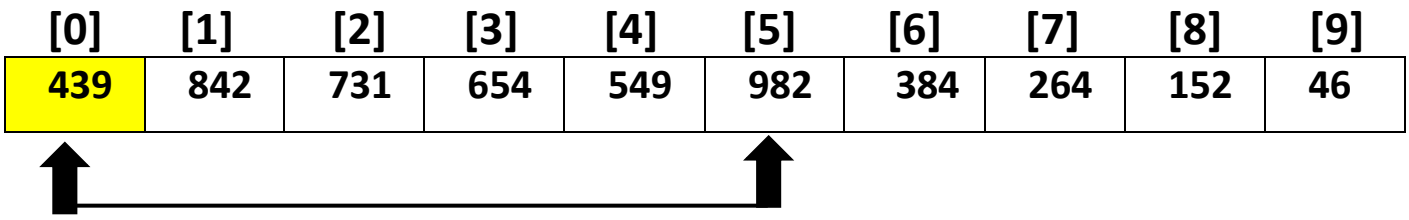


Gap = 5
nextPos = 5
insert (nextVal = table[5] = 439)
while(5> 4 && 439 < 982) True
table[5] = 982
nextPos = 0

Comparison = 1
Displacement = 1



Step 2 : NextPos = 0 (it was changed in while loop)

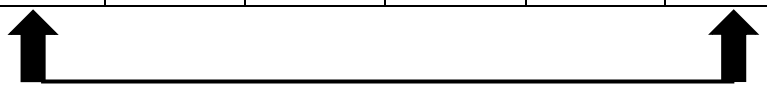


Gap = 5
nextPos = 0
insert (nextVal = 439)
while(0> 4) False X
table[0] = 439

Comparison = 1
Displacement = 2

Step 3 : NextPos = 6

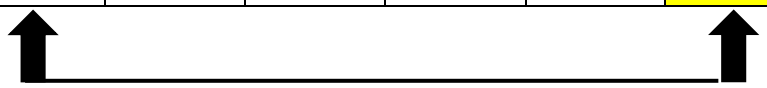
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
439	842	731	654	549	982	384	264	152	46



Gap = 5
nextPos = 6
insert (nextVal = table[6] = 384)
while(6> 4 && 384 < 842) True
table[6] = 842
nextPos = 1

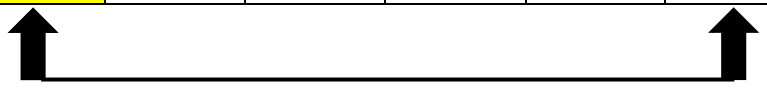
Comparison = 2
Displacement = 3

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
439	842	731	654	549	982	842	264	152	46



Step 4 : NextPos = 1 (it was changed in while loop)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
439	384	731	654	549	982	842	264	152	46

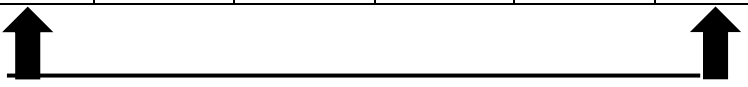


Gap = 5
nextPos = 1
insert (nextVal = 384)
while(1> 4) False X
table[1] = 384

Comparison = 2
Displacement = 4

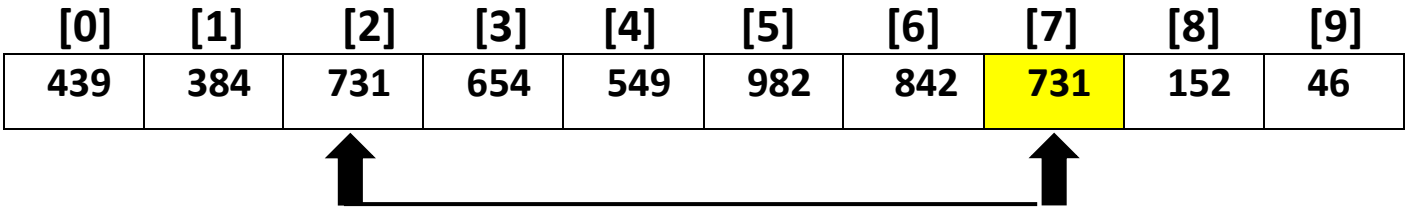
Step 5 : NextPos = 7

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
439	384	731	654	549	982	842	264	152	46

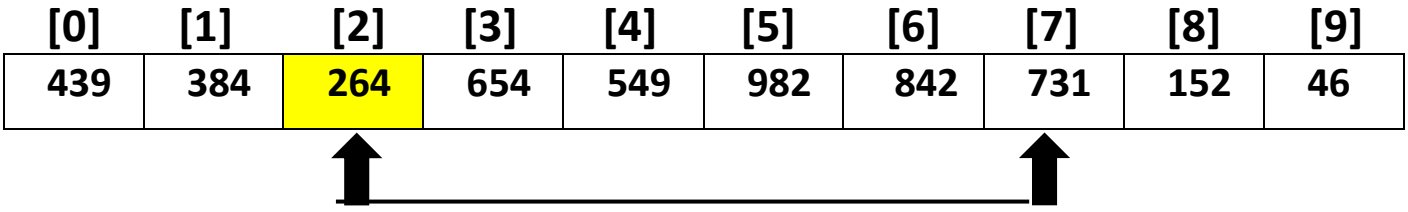


Gap = 5
nextPos = 7
insert (nextVal = table[7] = 264)
while(7> 4 && 264 < 731) True
table[7] = 731
nextPos = 2

Comparison = 3
Displacement = 5



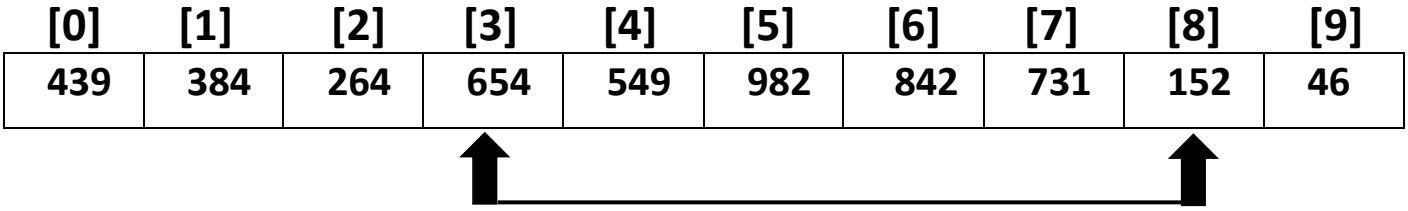
Step 6 : NextPos = 2 (it was changed in while loop)



Gap = 5
nextPos = 2
insert (nextVal = 264)
while(2> 4) False X
table[2] = 264

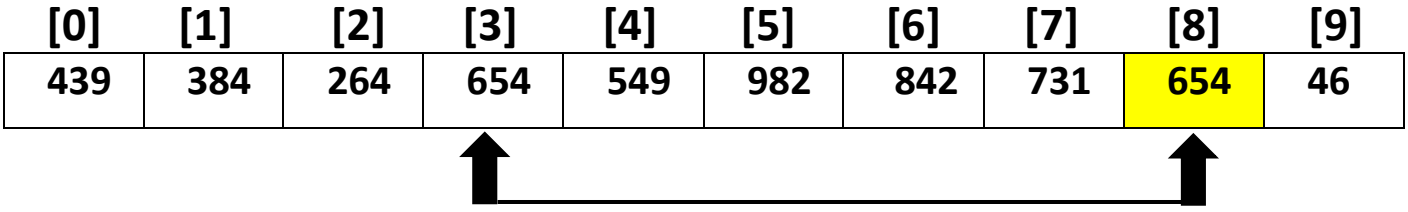
Comparison = 3
Displacement = 6

Step 7 : NextPos = 8



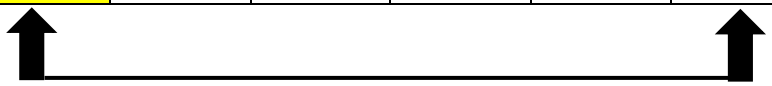
Gap = 5
nextPos = 8
insert (nextVal = table[8] = 152)
while(8> 4 && 152 < 654) True
table[8] = 654
nextPos = 3

Comparison = 4
Displacement = 7



Step 8 : NextPos = 3 (it was changed in while loop)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
439	384	264	152	549	982	842	731	654	46

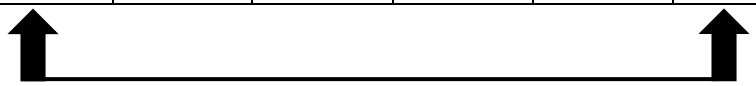


Gap = 5
nextPos = 3
insert (nextVal = table[8] = 152)
while(3> 4) False X
table[3] = 152

Comparison = 4
Displacement = 8

Step 9 : NextPos = 9

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
439	384	264	152	549	982	842	731	654	46



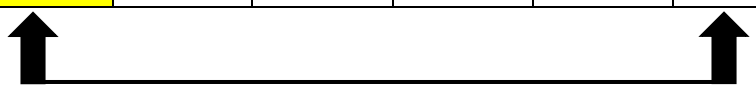
Gap = 5
nextPos = 9
insert (nextVal = table[9] = 46)
while(9> 4 && 46 < 549) True
table[9] = 549
nextPos = 4

Comparison = 5
Displacement = 9

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
439	384	264	152	549	982	842	731	654	549

Step 10 : NextPos = 4 (it was changed in while loop)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
439	384	264	152	46	982	842	731	654	549



Gap = 5
nextPos = 4
insert (nextVal = table[9] = 46)
while(4> 4) False X
table[4] = 46

Comparison = 5
Displacement = 10

Step 11 : Now the for loop is done (nextPos is 10 < 10). So we check the gap again.
Gap = 5/2.2 = 2.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
439	384	264	152	46	982	842	731	654	549

Gap = 2
nextPos = 2
insert (nextVal = table[2] = 264)
while(2> 1 && 264 < 439) True
table[2] = 439
nextPos = 0

Comparison = 6
Displacement = 11

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
439	384	439	152	46	982	842	731	654	549

Step 12 : NextPos is 0 in while loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
264	384	439	152	46	982	842	731	654	549

Gap = 2
nextPos = 0
insert (nextVal = table[2] = 264)
while(0> 1) False X
table[0] = 264

Comparison = 6
Displacement = 12

Step 13 : NextPos = 3

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
264	384	439	152	46	982	842	731	654	549

Gap = 2
nextPos = 3
insert (nextVal = table[3] = 152)
while(3> 1 && 152 < 384) True
table[3] = 384
nextPos = 1

Comparison = 7
Displacement = 13

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
264	384	439	384	46	982	842	731	654	549

Step 14 : NextPos is 1 in while loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
264	152	439	384	46	982	842	731	654	549

Gap = 2
nextPos = 1
insert (nextVal = table[3] = 152)
while(1> 1) False X
table[1] = 152

Comparison = 7
Displacement = 14

Step 15 : NextPos = 4 (sort method for loop)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
264	152	439	384	46	982	842	731	654	549

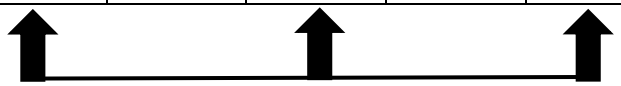
Gap = 2
nextPos = 4
insert (nextVal = table[4] = 46)
while(1> 1 && 46 < 439) True
table[4] = 439
nextPos = 2

Comparison = 8
Displacement = 15

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
264	152	439	384	439	982	842	731	654	549

Step 16 : NextPos is 2 in while loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
264	152	439	384	439	982	842	731	654	549



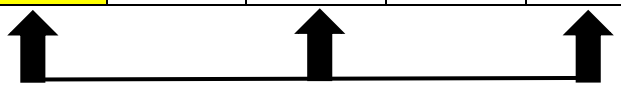
Gap = 2
nextPos = 2
insert (nextVal = table[4] = 46)
while(2> 1 && 46 < 264) True
table[2] = 264
nextPos = 0

Comparison = 10
Displacement = 16

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
264	152	264	384	439	982	842	731	654	549

Step 17 : NextPos is 0 in while loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	982	842	731	654	549



Gap = 2
nextPos = 0
insert (nextVal = table[4] = 46)
while(0> 1) False
table[0] = 46

Comparison = 10
Displacement = 17

Step 18 : NextPos is 5 in sort method

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	982	842	731	654	549

Gap = 2
nextPos = 5
insert (nextVal = table[5] = 982)
while(5> 1 && 982 < 384) False X
table[5] = 982

Comparison = 11
Displacement = 17

Step 19 : NextPos is 6 in sort method

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	982	842	731	654	549

Gap = 2
nextPos = 6
insert (nextVal = table[6] = 842)
while(6> 1 && 842 < 439) False
table[6] = 842

Comparison = 12
Displacement = 17

Step 20 : NextPos is 7 in sort method

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	982	842	731	654	549

Gap = 2
nextPos = 7
insert (nextVal = table[7] = 731)
while(6> 1 && 731 < 982) True
table[7] = 982
nextPost = 5

Comparison = 13
Displacement = 18

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	982	842	982	654	549

Step 21 : NextPos is 5 in while loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	731	842	982	654	549

Gap = 2
nextPos = 5
insert (nextVal = 731)
while(6> 1 && 731 < 384) False X
table[5] = 731

Comparison = 14
Displacement = 19

Step 22 : NextPos is 8 in sort method

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	731	842	982	654	549

Gap = 2
nextPos = 8
insert (nextVal = table[8] = 654)
while(8> 1 && 654 < 842) True
table[8] = 982
nextPost = 6

Comparison = 15
Displacement = 20

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	731	842	982	842	549

Step 23 : NextPos is 6 in while loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	731	654	982	842	549

Gap = 2
nextPos = 6
insert (nextVal = 654)
while(6> 1 && 654 < 439) False X
table[6] = 654

Comparison = 16
Displacement = 21

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	731	654	982	842	549

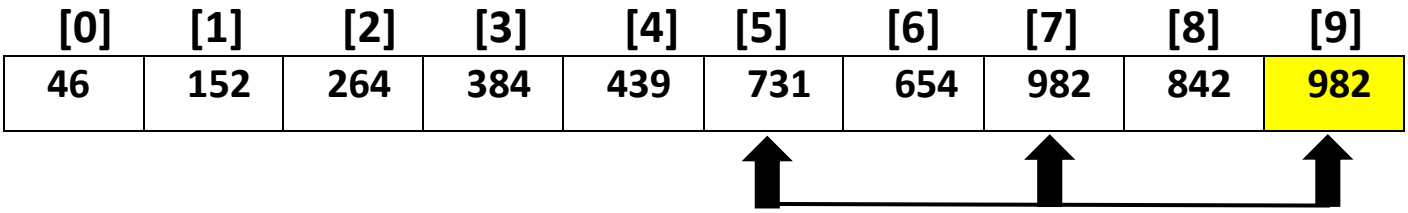
Step 24 : NextPos is 9 in sort method

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	731	654	982	842	549

Gap = 2
nextPos = 9
insert (nextVal = table[9] = 549)
while(9> 1 && 549 < 982) True
table[9] = 982
nextPos = 7

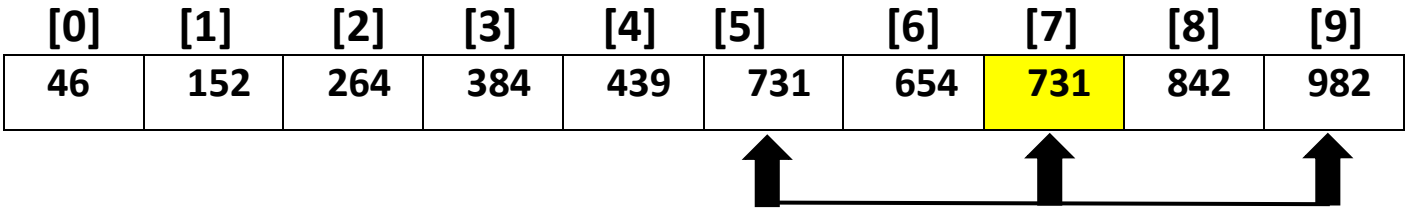
Comparison = 17
Displacement =22

Step 25 : NextPos is 7 in while loop

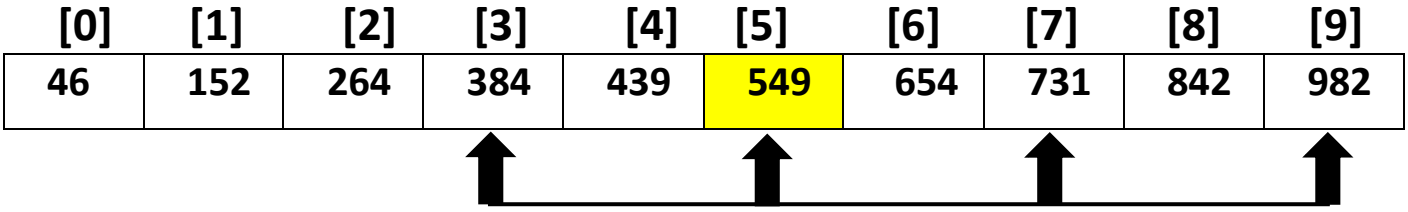


Gap = 2
nextPos = 7
insert (nextVal = 549)
while(7 > 1 && 549 < 731) True
table[7] = 731
nextPos = 5

Comparison = 18
Displacement = 23



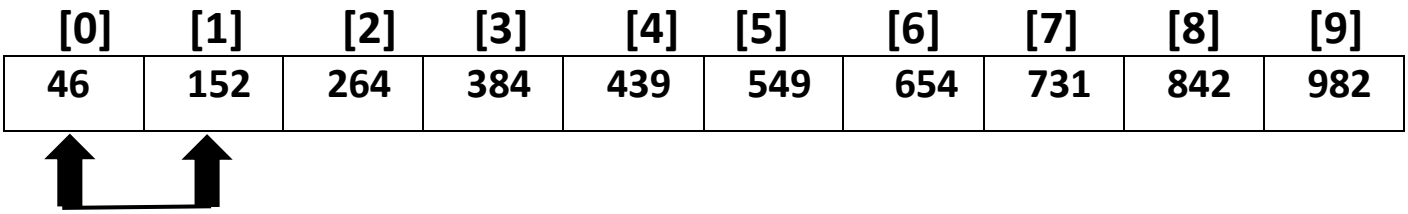
Step 26 : NextPos is 5 in while loop



Gap = 2
nextPos = 5
insert (nextVal = 549)
while(5 > 1 && 549 < 384) False X
table[5] = nextVal = 549

Comparison = 19
Displacement = 24

Step 27 : NextPos is done (sort for loop 10 < 10). So the gap is now 1.

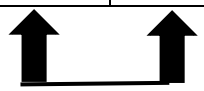


Gap = 1
nextPos = 1
insert (nextVal = table[1] = 152)
while(1 > 0 && 152 < 46) False X
table[1] = nextVal = 152

Comparison = 20
Displacement = 24

Step 28 : NextPos = 2 in for loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	549	654	731	842	982

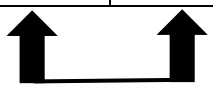


Gap = 1
nextPos = 2
insert (nextVal = table[2] = 264)
while(2 > 0 && 152 < 152) False X
table[2] = nextVal = 264

Comparison = 21
Displacement = 24

Step 29 : NextPos = 3 in for loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	549	654	731	842	982

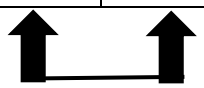


Gap = 1
nextPos = 3
insert (nextVal = table[3] = 384)
while(3 > 0 && 384 < 264) False X
table[3] = nextVal = 384

Comparison = 22
Displacement = 24

Step 30 : NextPos = 4 in for loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	549	654	731	842	982



Gap = 1
nextPos = 4
insert (nextVal = table[4] = 439)
while(4 > 0 && 439 < 384) False X
table[4] = nextVal = 439

Comparison = 23
Displacement = 24

Step 31 : NextPos = 5 in for loop

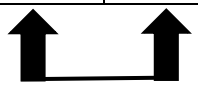
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	549	654	731	842	982

Gap = 1
nextPos = 5
insert (nextVal = table[5] = 549)
while(5 > 0 && 549 < 439) False X
table[5] = nextVal = 549

Comparison = 24
Displacement = 24

Step 32 : NextPos = 6 in for loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	549	654	731	842	982

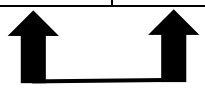


Gap = 1
nextPos = 6
insert (nextVal = table[6] = 654)
while(6 > 0 && 654 < 549) False X
table[6] = nextVal = 654

Comparison = 25
Displacement = 24

Step 33 : NextPos = 7 in for loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	549	654	731	842	982

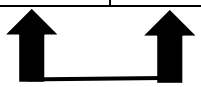


Gap = 1
nextPos = 7
insert (nextVal = table[7] = 982)
while(7 > 0 && 982 < 654) False X
table[7] = nextVal = 731

Comparison = 26
Displacement = 24

Step 34 : NextPos = 8 in for loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	549	654	731	842	982

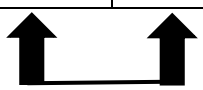


Gap = 1
nextPos = 8
insert (nextVal = table[8] = 842)
while(8 > 0 && 842 < 731) False X
table[8] = nextVal = 842

Comparison = 27
Displacement = 24

Step 35 : NextPos = 9 in for loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	549	654	731	842	982



Gap = 1
nextPos = 9
insert (nextVal = table[9] = 982)
while(9 > 0 && 982 < 842) False X
table[9] = nextVal = 982

Comparison = 28
Displacement = 24

Step 36 : Now nextPos is reached 10 so for loop and sorting was done. The last sorted array and counters are below :

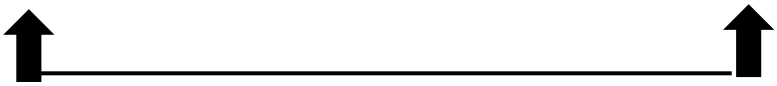
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	549	654	731	842	982

Comparison = 28
Displacement = 24

Sorting 1.c Array C (Integer Given Array -12 Elements) – Shell Sort Algorithm

Step 1 : The gap is array length / 2 which is 6. NextPos = gap.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
5	2	13	9	1	7	6	8	1	15	4	11



Gap = 6
nextPos = 6
insert (nextVal = table[6] = 6)
while(6> 5 && 6 < 5) False X
table[6] = 6

Comparison = 1
Displacement = 0

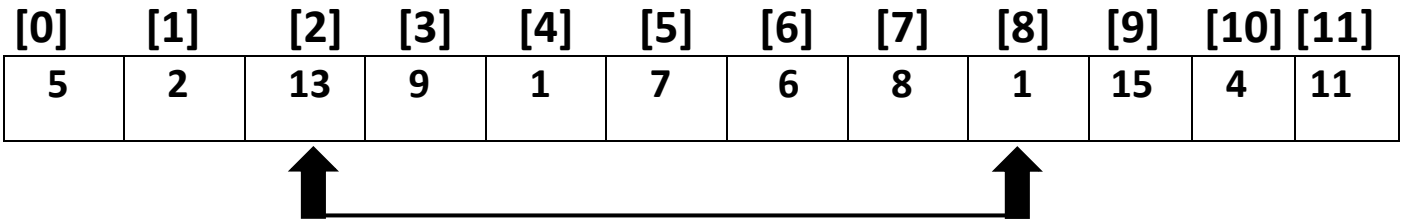
Step 2 : NextPos = 7 in for loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
5	2	13	9	1	7	6	8	1	15	4	11

Gap = 6
nextPos = 7
insert (nextVal = table[7] = 8)
while(7> 5 && 8 < 2) False X
table[7] = 8

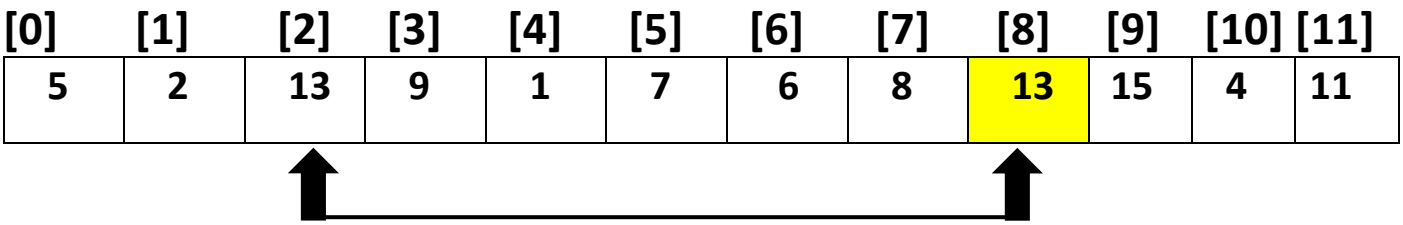
Comparison = 2
Displacement = 0

Step 3 : NextPos = 8 in for loop

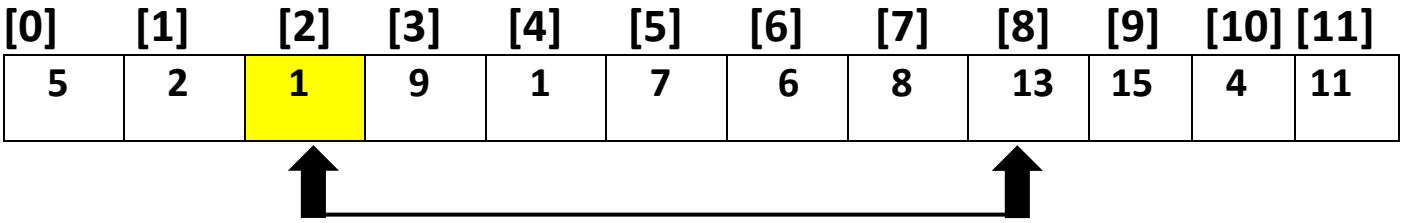


Gap = 6
nextPos = 8
insert (nextVal = table[8] = 1)
while(8> 5 && 1 < 13) True
table[8] = 13
nextPos = 2

Comparison = 3
Displacement = 1



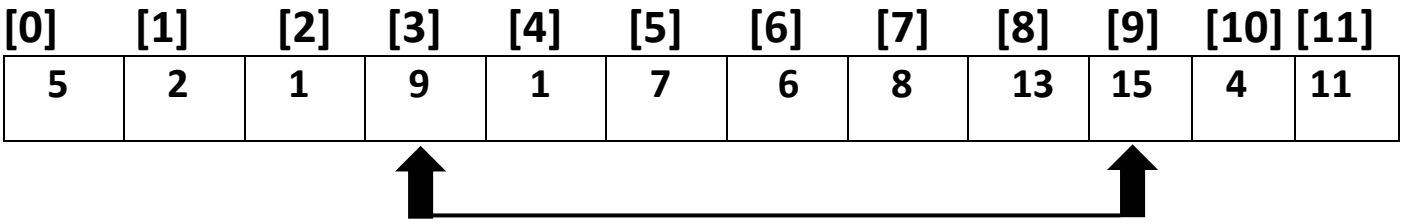
Step 4 : NextPos = 2 in while loop



Gap = 6
nextPos = 2
insert (nextVal = 1)
while(2> 5) False X
table[2] = 1

Comparison = 3
Displacement = 2

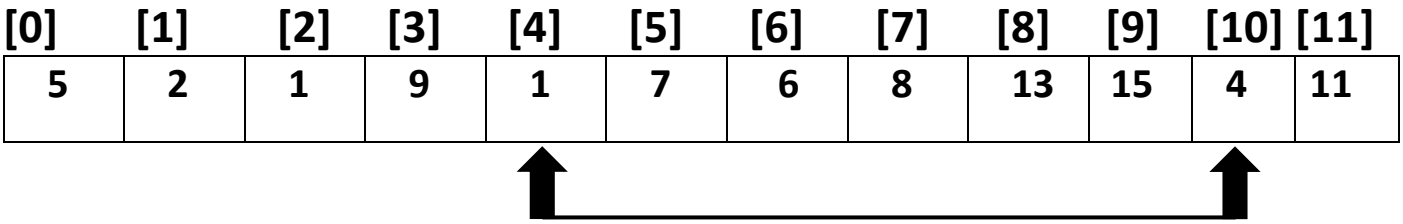
Step 5 : NextPos = 9 in for loop



Gap = 6
nextPos = 9
insert (nextVal = table[9] = 15)
while(9> 5 && 15 < 9) False X
table[9] = 15

Comparison = 4
Displacement = 2

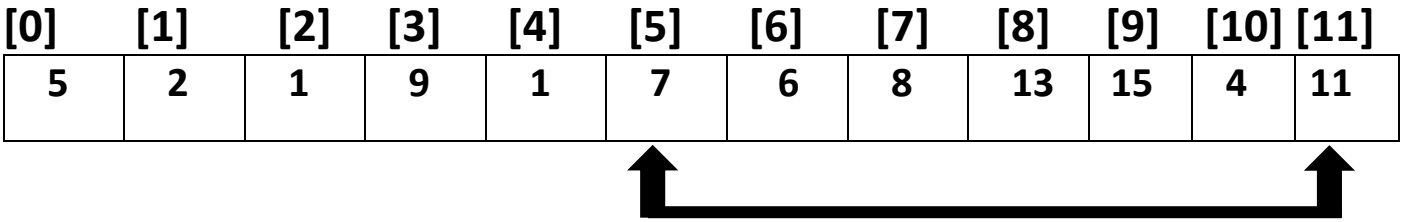
Step 6 : NextPos = 10 in for loop



Gap = 6
nextPos = 10
insert (nextVal = table[9] = 15)
while(10> 5 && 4 < 1) False X
table[10] = 4

Comparison = 5
Displacement = 2

Step 7 : NextPos = 11 in for loop

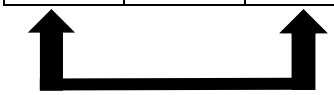


Gap = 6
nextPos = 11
insert (nextVal = table[11] = 11)
while(10> 5 && 11 < 7) False X
table[11] = 11

Comparison = 6
Displacement = 2

Step 8 : NextPos is 12 and reached the array length. So I change the gap. Gap = 6 / 2.2 = 2. NextPos = gap

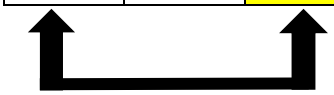
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
5	2	1	9	1	7	6	8	13	15	4	11



Gap = 2
nextPos = 2
insert (nextVal = table[2] = 1)
while(2> 1 && 1 < 5) True
table[2] = 5
nextPos = 0

Comparison = 7
Displacement = 3

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
5	2	5	9	1	7	6	8	13	15	4	11



Step 9 : NextPos is 0 in while loop

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	2	5	9	1	7	6	8	13	15	4	11

Gap = 2
nextPos = 2
insert (nextVal = table[2] = 1)
while(0> 1) False X
table[0] = 1

Comparison = 7
Displacement = 4

Step 10 : NextPos is 3 in for loop (sort method loop)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	2	5	9	1	7	6	8	13	15	4	11

Gap = 2
nextPos = 3
insert (nextVal = table[3] = 9)
while(3 > 1 && 9 < 2) False X
table[3] = 9

Comparison = 8
Displacement = 4

Step 11 : NextPos is 4 in for loop (sort method loop)

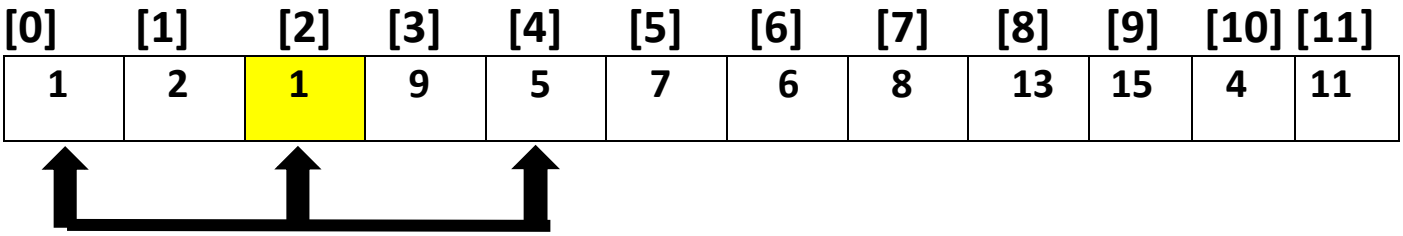
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	2	5	9	1	7	6	8	13	15	4	11

Gap = 2
nextPos = 4
insert (nextVal = table[4] = 1)
while(4 > 1 && 1 < 5) True
table[4] = 5
nextPos = 2

Comparison = 9
Displacement = 5

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	2	5	9	5	7	6	8	13	15	4	11

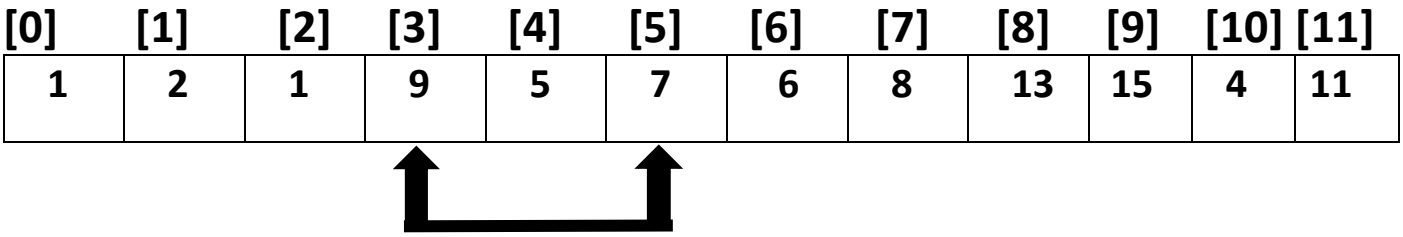
Step 12 : NextPos is 2 in while loop (insert method)



Gap = 2
nextPos = 2
insert (nextVal = 1)
while(2 > 1 && 1 < 1) False X
table[2] = 1

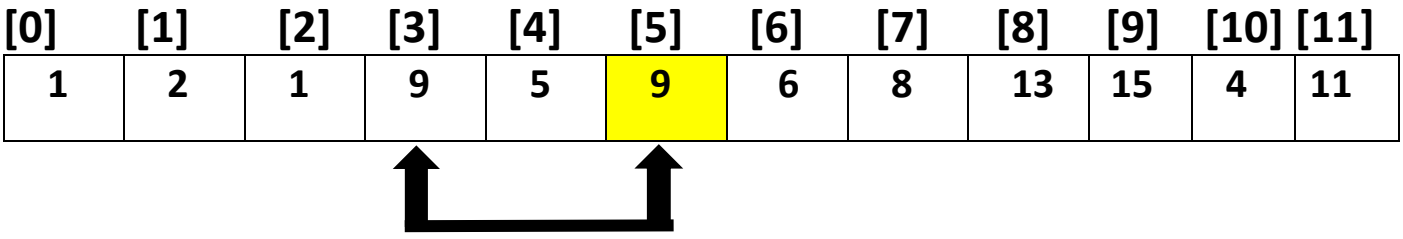
Comparison = 9
Displacement = 6

Step 13 : NextPos is 5 in for loop (insert method)

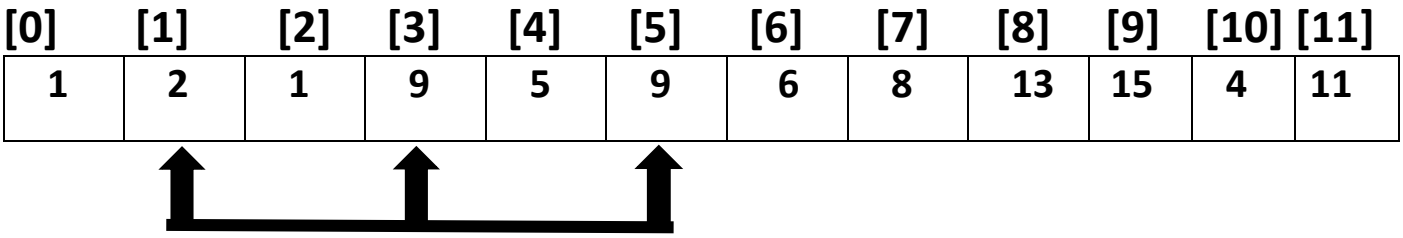


Gap = 2
nextPos = 5
insert (nextVal = table[5] = 7)
while(5 > 1 && 7 < 9) True
table[5] = 9
nextPos = 3

Comparison = 10
Displacement = 7

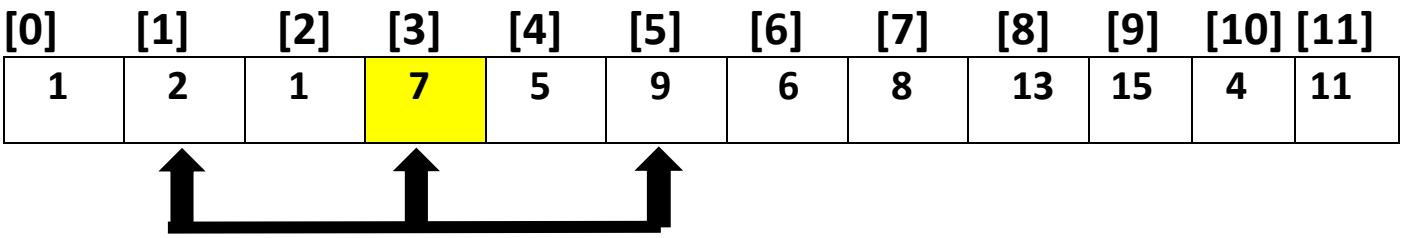


Step 14 : NextPos is 3 in while loop (insert method)

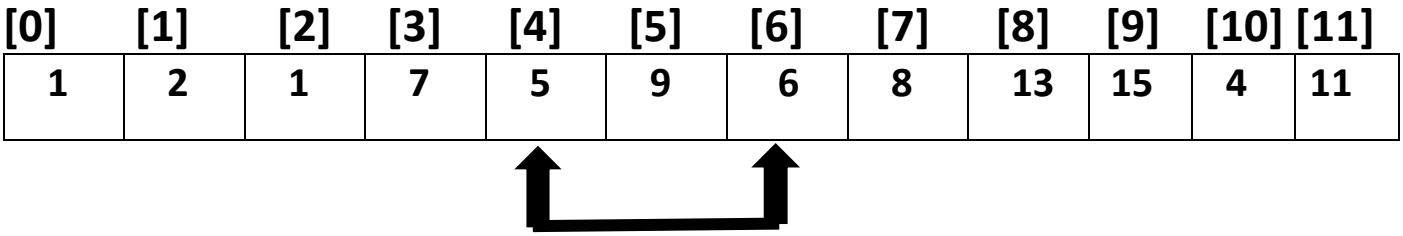


Gap = 2
nextPos = 3
insert (nextVal = 7)
while(3 > 1 && 7 < 2) False X
table[3] = 7

Comparison = 11
Displacement = 8



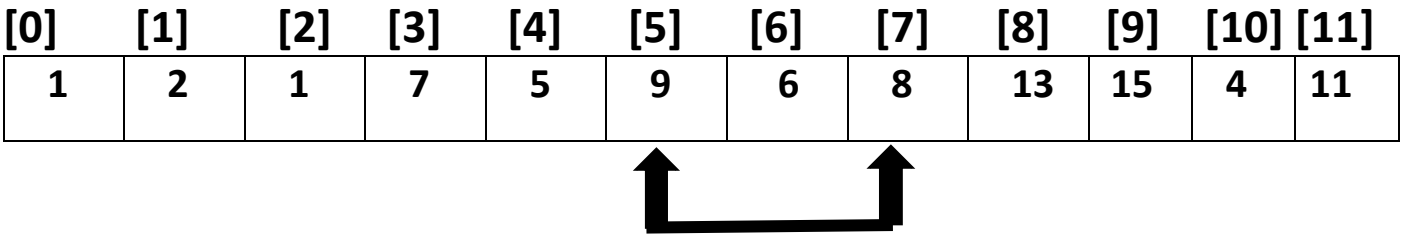
Step 15 : NextPos is 6 in for loop (sort method)



Gap = 2
nextPos = 6
insert (nextVal = table[6] = 6)
while(6 > 1 && 6 < 5) False X
table[6] = 6

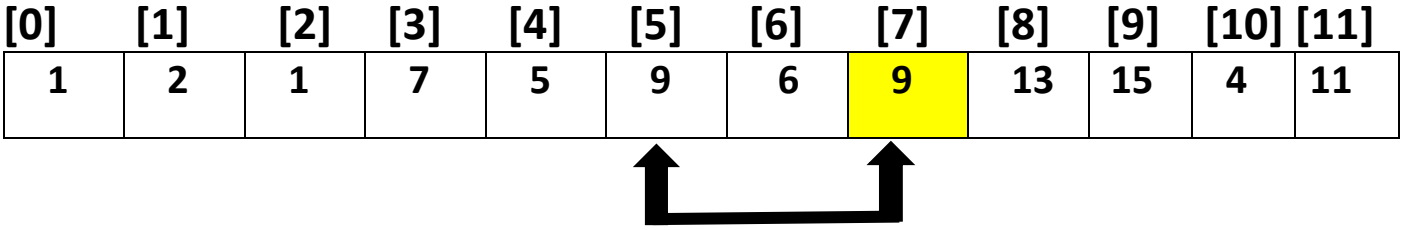
Comparison = 12
Displacement = 8

Step 16 : NextPos is 7 in for loop (sort method)

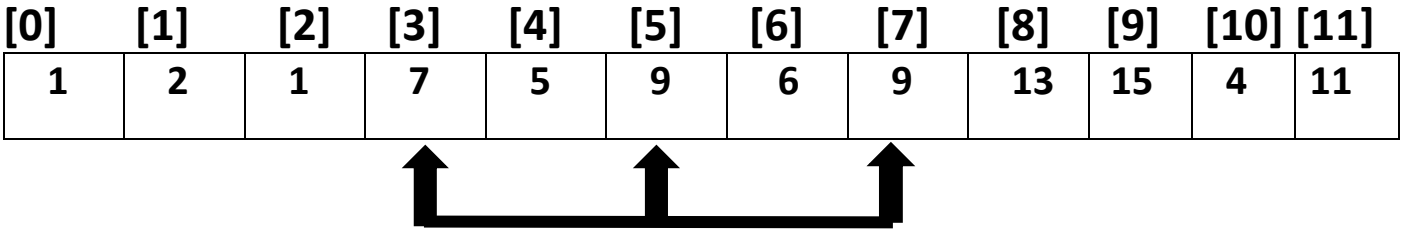


Gap = 2
nextPos = 7
insert (nextVal = table[7] = 8)
while(7 > 1 && 8 < 9) True
table[7] = 9
nextPos = 5

Comparison = 13
Displacement = 9

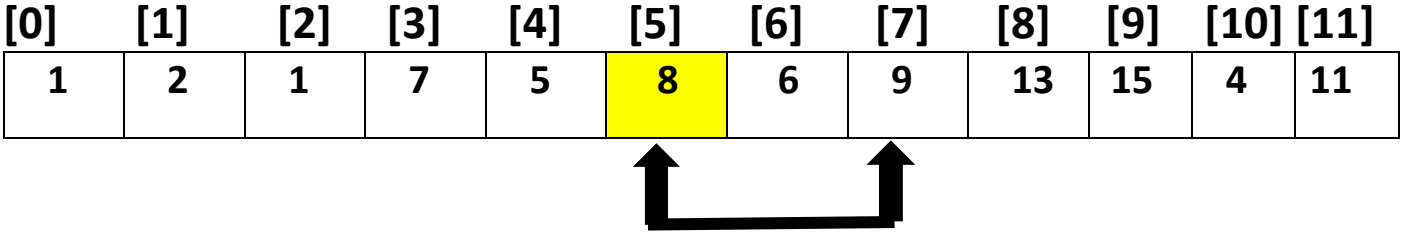


Step 17 : NextPos is 5 in while loop (insert method)



Gap = 2
nextPos = 5
insert (nextVal = table[7] = 8)
while(5 > 1 && 8 < 7) False X
table[5] = 8

Comparison = 14
Displacement = 10



Step 18 : NextPos is 8 in for loop (sort method)

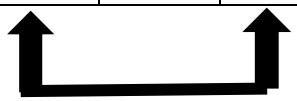
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	2	1	7	5	8	6	9	13	15	4	11

Gap = 2
nextPos = 8
insert (nextVal = table[8] = 13)
while(8 > 1 && 13 < 6) False X
table[8] = 13

Comparison = 15
Displacement = 10

Step 19 : NextPos is 9 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	2	1	7	5	8	6	9	13	15	4	11



Gap = 2
nextPos = 9
insert (nextVal = table[9] = 15)
while(8 > 1 && 15 < 9) False X
table[9] = 15

Comparison = 16
Displacement = 10

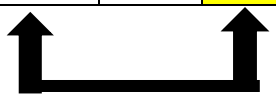
Step 20 : NextPos is 10 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	2	1	7	5	8	6	9	13	15	4	11

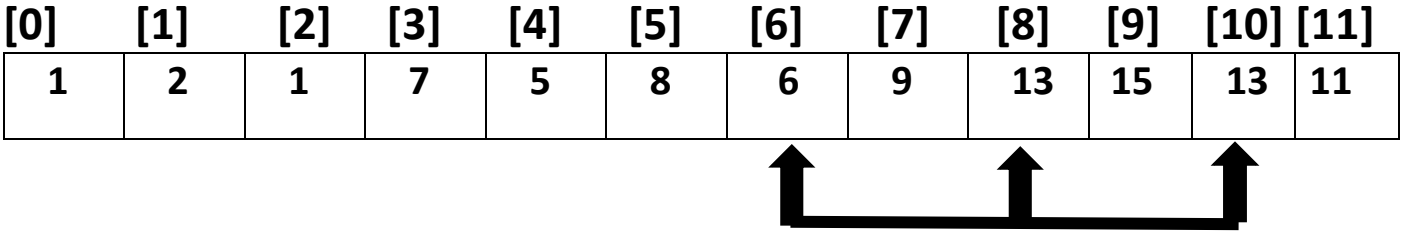
Gap = 2
nextPos = 10
insert (nextVal = table[10] = 4)
while(10 > 1 && 4 < 13) True
table[10] = 13
nextPos = 8

Comparison = 17
Dislacement = 11

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	2	1	7	5	8	6	9	13	15	13	11

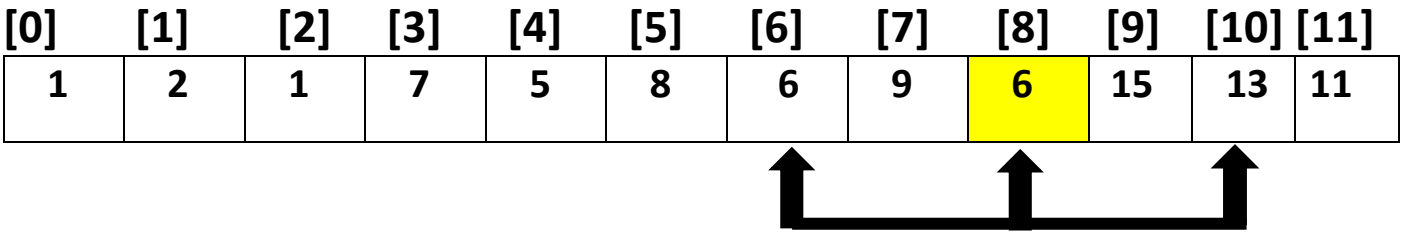


Step 21 : NextPos is 8 in while loop (insert method)

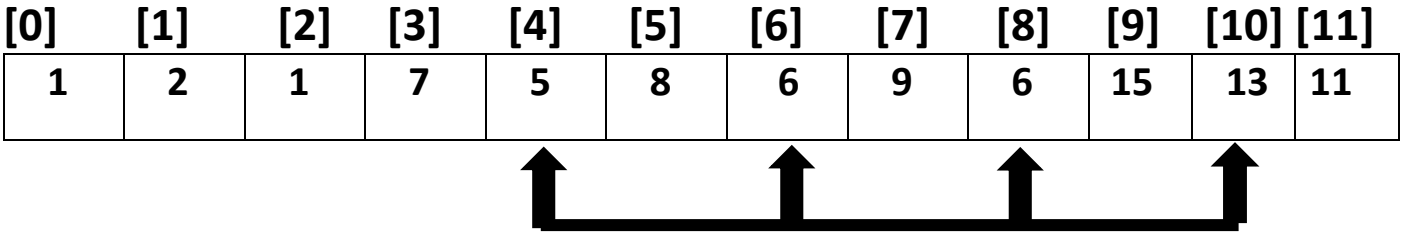


Gap = 2
nextPos = 8
insert (nextVal = 4)
while(8 > 1 && 4 < 6) True
table[8] = 6
nextPos = 6

Comparison = 18
Displacement = 12

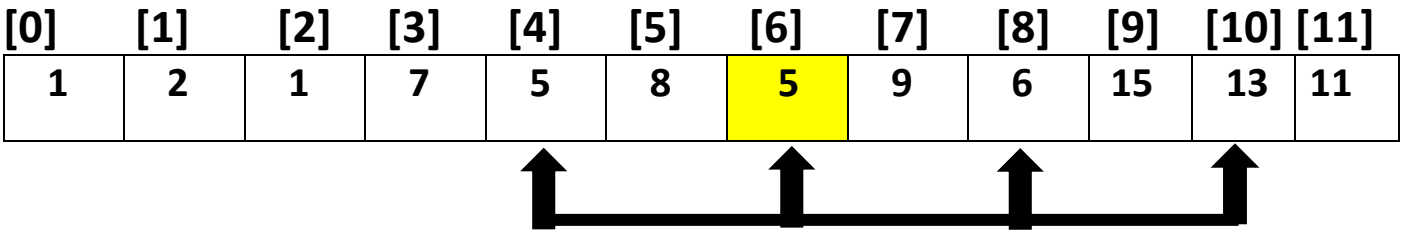


Step 22 : NextPos is 6 in while loop (insert method)



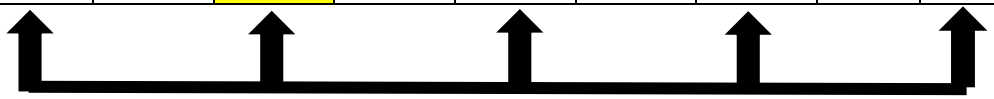
Gap = 2
nextPos = 6
insert (nextVal = 4)
while(6 > 1 && 4 < 5) True
table[6] = 5
nextPos = 4

Comparison = 19
Displacement = 13



Step 23 : NextPos is 4 in while loop (insert method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	2	1	7	4	8	5	9	6	15	13	11



Gap = 2
nextPos = 4
insert (nextVal = 4)
while(4 > 1 && 4 < 1) False X
table[4] = 4

Comparison = 20
Displacement = 14

Step 24 : NextPos is 11 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	2	1	7	4	8	5	9	6	15	13	11

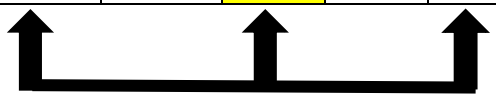
Gap = 2
nextPos = 11
insert (nextVal = table[11] = 11)
while(11 > 1 && 11 < 15) True
table[11] = 15
nextPos = 9

Comparison = 21
Displacement = 15

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	2	1	7	4	8	5	9	6	15	13	15

Step 25 : NextPos is 9 in while loop (insert method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	2	1	7	4	8	5	9	6	11	13	15



Gap = 2
nextPos = 9
insert (nextVal = 11)
while(9 > 1 && 11 < 9) False X
table[9] = 11

Comparison = 22
Displacement = 16

Step 26 : NextPos is 12 and reached the length. So now I will change the gap. İf gap == 2, gap = 1. So gap = NextPos = 1

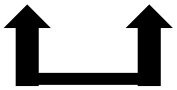
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	2	1	7	4	8	5	9	6	11	13	15

Gap = 1
nextPos = 1
insert (nextVal = table[1] = 2)
while(1 > 0 && 2 < 1) False X
table[1] = 2

Comparison = 23
Displacement = 16

Step 27 : NextPos = 2 in for loop (sort method)

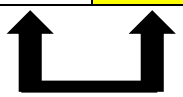
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	2	1	7	4	8	5	9	6	11	13	15



Gap = 1
nextPos = 2
insert (nextVal = table[2] = 1)
while(2 > 0 && 1 < 2) True
table[2] = 2
nextPost = 1

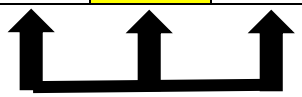
Comparison = 24
Displacement = 17

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	2	2	7	4	8	5	9	6	11	13	15



Step 28 : NextPos = 1 in while loop (insert method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	7	4	8	5	9	6	11	13	15



Gap = 1
nextPos = 1
insert (nextVal = table[2] = 1)
while(1 > 1) False X
table[1] = 1

Comparison = 24
Displacement = 18

Step 29 : NextPos = 3 in for loop (sort method)

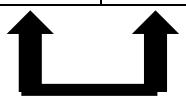
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	7	4	8	5	9	6	11	13	15

Gap = 1
nextPos = 3
insert (nextVal = table[3] = 7)
while(3 > 1 && 7 < 2) False X
table[3] = 7

Comparison = 25
Displacement = 18

Step 30 : NextPos = 4 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	7	4	8	5	9	6	11	13	15



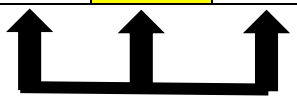
Gap = 1
nextPos = 4
insert (nextVal = table[4] = 4)
while(4 > 1 && 4 < 7) True
table[4] = 7
nextPos = 3

Comparison = 26
Displacement = 19

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	7	7	8	5	9	6	11	13	15

Step 31 : NextPos = 3 in while loop (insert method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	7	8	5	9	6	11	13	15

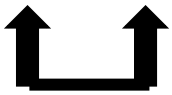


Gap = 1
nextPos = 3
insert (nextVal = 4)
while(3 > 1 && 4 < 2) False X
table[3] = 4

Comparison = 27
Displacement = 20

Step 32 : NextPos = 5 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	7	8	5	9	6	11	13	15

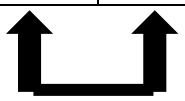


Gap = 1
nextPos = 5
insert (nextVal = table[5] = 8)
while(5 > 1 && 8< 7) False X
table[5] = 8

Comparison = 28
Displacement = 20

Step 33 : NextPos = 6 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	7	8	5	9	6	11	13	15



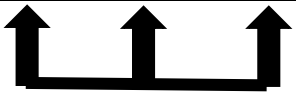
Gap = 1
nextPos = 6
insert (nextVal = table[6] = 5)
while(6 > 1 && 5< 8) True
table[6] = 8
nextPos = 5

Comparison = 29
Displacement = 21

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	7	8	8	9	6	11	13	15

Step 34 : NextPos = 5 in while loop (insert method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	7	8	8	9	6	11	13	15



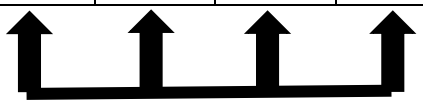
Gap = 1
nextPos = 5
insert (nextVal = 5)
while(5 > 1 && 5< 7) True
table[5] = 7
nextPos = 4

Comparison = 30
Displacement = 22

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	7	7	8	9	6	11	13	15

Step 35 : NextPos = 4 in while loop (insert method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	7	7	8	9	6	11	13	15



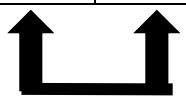
Gap = 1
nextPos = 4
insert (nextVal = 5)
while(4 > 1 && 5< 4) False X
table[4] = 5

Comparison = 31
Displacement = 23

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	7	8	9	6	11	13	15

Step 36 : NextPos = 7 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	7	8	9	6	11	13	15

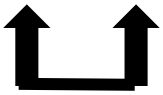


Gap = 1
nextPos = 7
insert (nextVal = table[7] = 9)
while(7 > 1 && 9< 8) False X
table[7] = 9

Comparison = 32
Displacement = 23

Step 37 : NextPos = 8 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	7	8	9	6	11	13	15



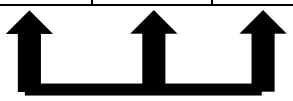
Gap = 1
nextPos = 8
insert (nextVal = table[8] = 6)
while(8 > 1 && 6< 9) True
table[8] = 9
nextPos = 7

Comparison = 33
Displacement = 24

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	7	8	9	9	11	13	15

Step 38 : NextPos = 7 in while loop (insert method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	7	8	9	9	11	13	15

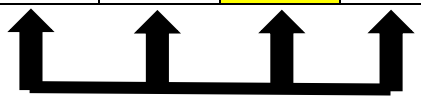


Gap = 1
nextPos = 7
insert (nextVal = table[8] = 6)
while(7 > 1 && 6< 8) True
table[7] = 8
nextPos = 6

Comparison = 34
Displacement = 25

Step 39 : NextPos = 6 in while loop (insert method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	7	8	8	9	11	13	15



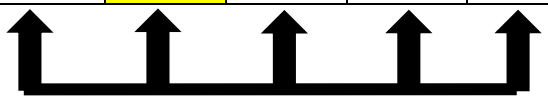
Gap = 1
nextPos = 6
insert (nextVal = table[8] = 6)
while(6 > 1 && 6< 7) True
table[6] = 7
nextPos = 5

Comparison = 35
Displacement = 26

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	7	7	8	9	11	13	15

Step 40 : NextPos = 5 in while loop (insert method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	6	7	8	9	11	13	15

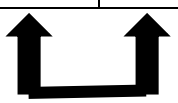


Gap = 1
nextPos = 5
insert (nextVal = table[8] = 6)
while(5 > 1 && 6< 5) False X
table[5] = 6

Comparison = 36
Displacement = 27

Step 41 : NextPos = 9 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	6	7	8	9	11	13	15

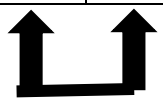


Gap = 1
nextPos = 9
insert (nextVal = table[9] = 11)
while(9 > 1 && 11< 9) False X
table[9] = 11

Comparison = 37
Displacement = 27

Step 42 : NextPos = 10 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	6	7	8	9	11	13	15

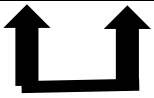


Gap = 1
nextPos = 10
insert (nextVal = table[10] = 13)
while(9 > 1 && 13< 11) False X
table[10] = 13

Comparison = 38
Displacement = 27

Step 43 : NextPos = 11 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	6	7	8	9	11	13	15



Gap = 1
nextPos = 11
insert (nextVal = table[11] = 15)
while(9 > 1 && 15< 13) False X
table[11] = 15

Comparison = 39
Displacement = 27

Step 44 : NextPos reached the length of array. So loops and sorting are done. The last counters and sorted array is below :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	6	7	8	9	11	13	15

Comparison = 39
Displacement = 27

Sorting 1.d Array D (Char Given Array -12 Elements) – Shell Sort Algorithm

Step 1 : The gap is array length / 2 which is 6. NextPos = gap.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
S	B	I	M	H	Q	C	L	R	E	P	K

Gap = 6
nextPos = 6
insert (nextVal = table[6] = C)
while(6 > 5 && C < S) True
table[6] = S
nextPos = 0

Comparison = 1
Displacement = 1

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
S	B	I	M	H	Q	S	L	R	E	P	K

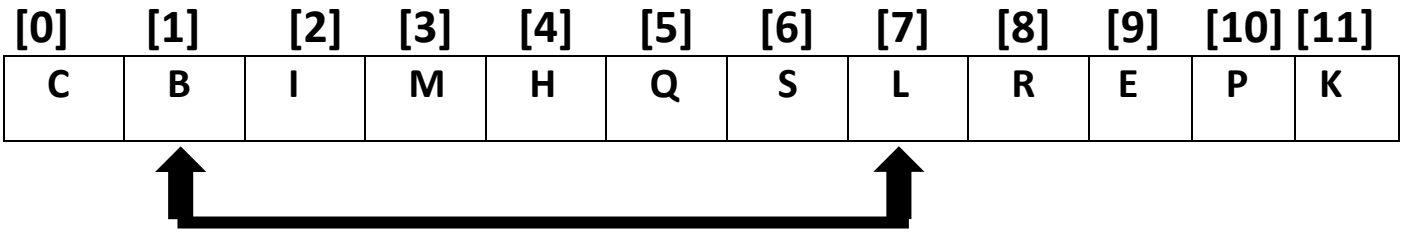
Step 2 : NextPos = 0 in while loop (insert method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	I	M	H	Q	S	L	R	E	P	K

Gap = 6
nextPos = 0
insert (nextVal = table[6] = C)
while(0 > 5) False X
table[0] = C

Comparison = 1
Displacement = 2

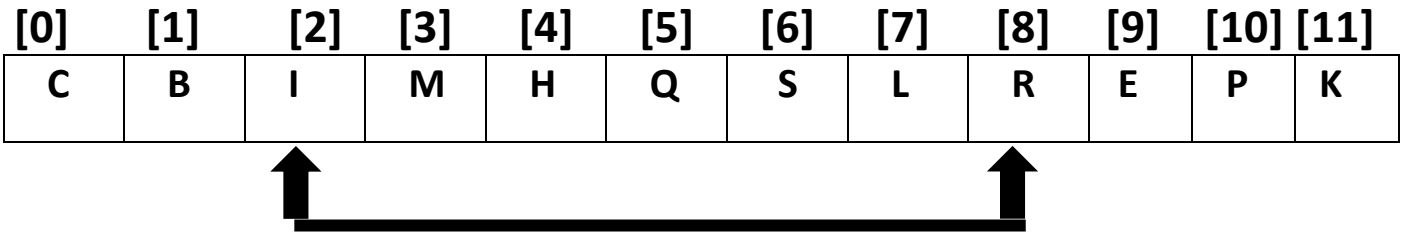
Step 3 : NextPos = 7 in for loop (sort method)



Gap = 6
nextPos = 7
insert (nextVal = table[6] = L)
while(7 > 5 && L < B) False X
table[7] = L

Comparison = 2
Displacement = 2

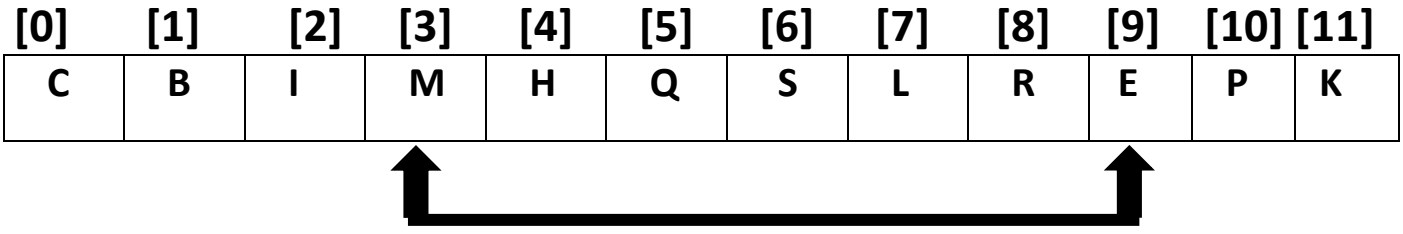
Step 4 : NextPos = 8 in for loop (sort method)



Gap = 6
nextPos = 8
insert (nextVal = table[8] = R)
while(8 > 5 && R < I) False X
table[8] = R

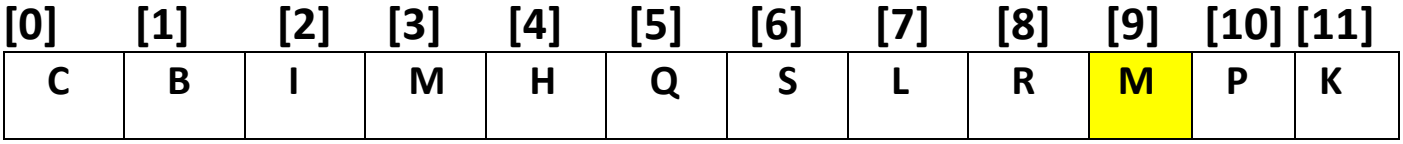
Comparison = 3
Displacement = 2

Step 5 : NextPos = 9 in for loop (sort method)



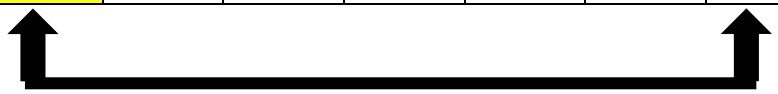
Gap = 6
nextPos = 9
insert (nextVal = table[9] = E)
while(9 > 5 && E < M) True
table[9] = M
nextPos = 3

Comparison = 4
Displacement = 3



Step 6 : NextPos = 3 in while loop (insert method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	I	E	H	Q	S	L	R	M	P	K

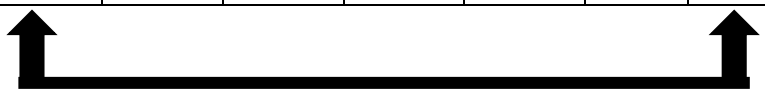


Gap = 6
nextPos = 3
insert (nextVal = table[9] = E)
while(3 > 5) False X
table[3] = E

Comparison = 4
Displacement = 4

Step 7 : NextPos = 10 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	I	E	H	Q	S	L	R	M	P	K

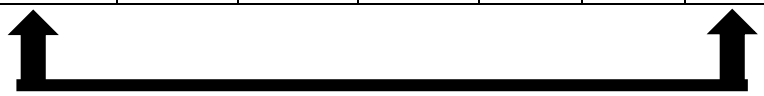


Gap = 6
nextPos = 10
insert (nextVal = table[10] = P)
while(10 > 5 && P < H) False X
table[10] = P

Comparison = 5
Displacement = 4

Step 8 : NextPos = 11 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	I	E	H	Q	S	L	R	M	P	K



Gap = 6
nextPos = 11
insert (nextVal = table[11] = K)
while(11 > 5 && P < H) True
table[11] = Q
nextPos = 5

Comparison = 6
Displacement = 5

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	I	E	H	Q	S	L	R	M	P	Q

Step 9 : NextPos = 5 in while loop (insert method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	I	E	H	K	S	L	R	M	P	Q

Gap = 6
nextPos = 5
insert (nextVal = table[11] = K)
while(5 > 5) False X
table[5] = K

Comparison = 6
Displacement = 6

Step 10: NextPos reached the array length. So I will change the gap. Gap = 6 / 2.2 = 2. NextPos is also 2.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	I	E	H	K	S	L	R	M	P	Q

Gap = 2
nextPos = 2
insert (nextVal = table[2] = I)
while(2 > 1 && I < C) False X
table[2] = I

Comparison = 7
Displacement = 6

Step 11: NextPos = 3 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	I	E	H	K	S	L	R	M	P	Q

Gap = 2
nextPos = 3
insert (nextVal = table[3] = E)
while(3 > 1 && E < B) False X
table[3] = E

Comparison = 8
Displacement = 6

Step 12: NextPos = 4 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	I	E	H	K	S	L	R	M	P	Q

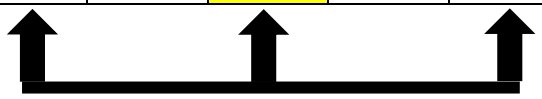
Gap = 2
nextPos = 4
insert (nextVal = table[4] = H)
while(4 > 1 && H < I) True
table[4] = I
nextPos = 2

Comparison = 9
Displacement = 7

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	I	E	I	K	S	L	R	M	P	Q

Step 13: NextPos = 2 in while loop (insert method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	H	E	I	K	S	L	R	M	P	Q



Gap = 2
nextPos = 2
insert (nextVal = table[4] = H)
while(2 > 1 && H < C) False X
table[2] = H

Comparison = 10
Displacement = 8

Step 14: NextPos = 5 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	H	E	I	K	S	L	R	M	P	Q

Gap = 2
nextPos = 5
insert (nextVal = table[5] = K)
while(5 > 1 && K < E) False X
table[5] = K

Comparison = 11
Displacement = 8

Step 15: NextPos = 6 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	H	E	I	K	S	L	R	M	P	Q

Gap = 2
nextPos = 6
insert (nextVal = table[6] = S)
while(2 > 1 && S < I) False X
table[6] = S

Comparison = 12
Displacement = 8

Step 16: NextPos = 7 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	H	E	I	K	S	L	R	M	P	Q

Gap = 2
nextPos = 7
insert (nextVal = table[7] = L)
while(7 > 1 && L < K) False X
table[7] = L

Comparison = 13
Displacement = 8

Step 17: NextPos = 8 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	H	E	I	K	S	L	R	M	P	Q

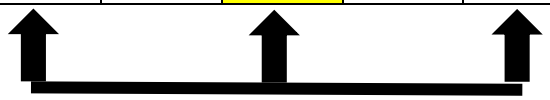
Gap = 2
nextPos = 8
insert (nextVal = table[8] = R)
while(8 > 1 && R < S) True
table[8] = S
nextPos = 6

Comparison = 14
Displacement = 9

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	H	E	I	K	S	L	S	M	P	Q

Step 18: NextPos = 6 in while loop (insert method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	H	E	I	K	R	L	S	M	P	Q



Gap = 2
nextPos = 6
insert (nextVal = table[8] = R)
while(6 > 1 && R < I) False X
table[6] = R

Comparison = 15
Displacement = 10

Step 19: NextPos = 9 in for loop (sort method)

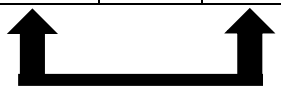
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	H	E	I	K	R	L	S	M	P	Q

Gap = 2
nextPos = 9
insert (nextVal = table[9] = M)
while(9 > 1 && M < L) False X
table[9] = M

Comparison = 16
Displacement = 10

Step 20: NextPos = 10 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	H	E	I	K	R	L	S	M	P	Q

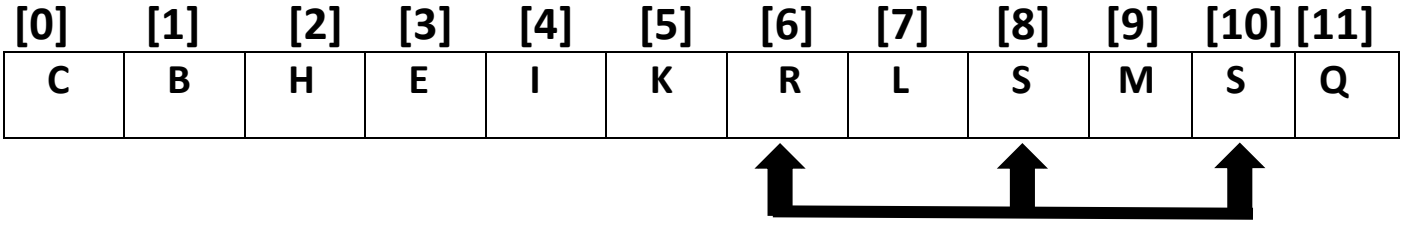


Gap = 2
nextPos = 10
insert (nextVal = table[10] = P)
while(9 > 1 && P < S) True
table[10] = S
nextPos = 8

Comparison = 17
Displacement = 11

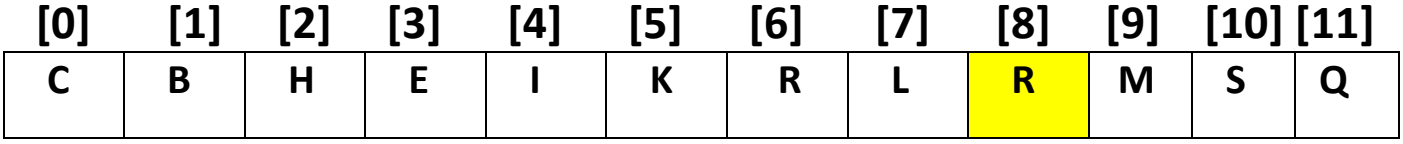
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	H	E	I	K	R	L	S	M	S	Q

Step 21: NextPos = 8 in while loop (insert method)

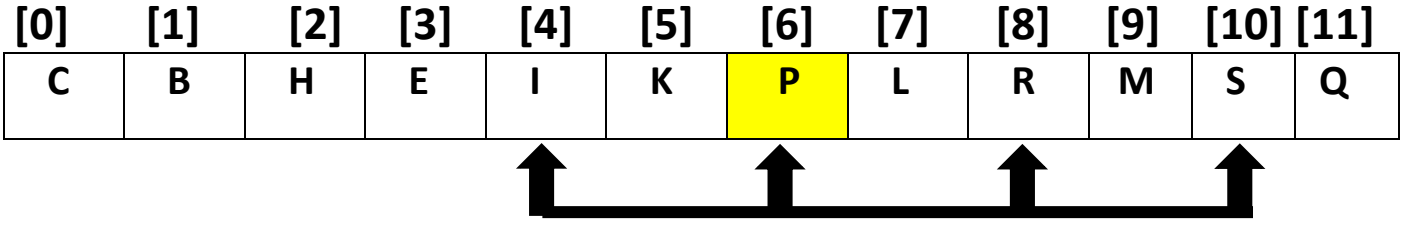


Gap = 2
nextPos = 8
insert (nextVal = table[10] = P)
while(8 > 1 && P < R) True
table[8] = R
nextPos = 6

Comparison = 18
Displacement = 12



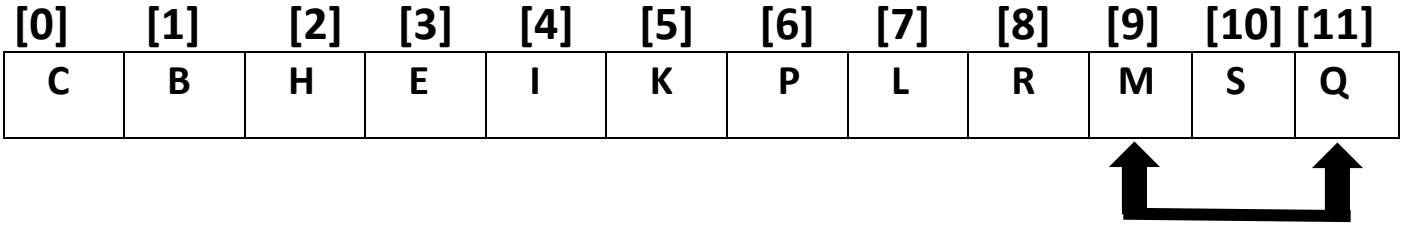
Step 22: NextPos = 6 in while loop (insert method)



Gap = 2
nextPos = 6
insert (nextVal = table[10] = P)
while(6 > 1 && P < I) False X
table[6] = P

Comparison = 19
Displacement = 13

Step 23: NextPos = 11 in for loop (sort method)

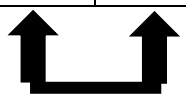


Gap = 2
nextPos = 11
insert (nextVal = table[11] = Q)
while(11 > 1 && Q < M) False X
table[11] = Q

Comparison = 20
Displacement = 13

Step 24: NextPos reached the array length. So I will change the gap. If gap == 2, then gap = 1. Also nextPos = 1.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	H	E	I	K	P	L	R	M	S	Q



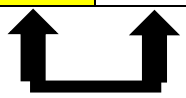
Gap = 1
nextPos = 1
insert (nextVal = table[1] = B)
while(10 > 0 && B < C) True
table[1] = C
nextPos = 0

Comparison = 21
Displacement = 14

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	C	H	E	I	K	P	L	R	M	S	Q

Step 25: NextPos = 0 in while loop (insert method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	H	E	I	K	P	L	R	M	S	Q



Gap = 1
nextPos = 0
insert (nextVal = table[1] = B)
while(0 > 0) False X
table[0] = B

Comparison = 21
Displacement = 15

Step 26: NextPos = 2 in for loop (sort method)

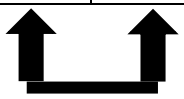
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	H	E	I	K	P	L	R	M	S	Q

Gap = 1
nextPos = 2
insert (nextVal = table[2] = H)
while(2 > 0 && H < C) False X
table[2] = H

Comparison = 22
Displacement = 15

Step 27: NextPos = 3 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	H	E	I	K	P	L	R	M	S	Q



Gap = 1
nextPos = 3
insert (nextVal = table[3] = E)
while(3 > 0 && E < H) True
table[3] = H
nextPos = 2

Comparison = 23
Displacement = 16

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	H	H	I	K	P	L	R	M	S	Q

Step 28: NextPos = 2 in while loop (insert method)

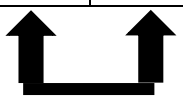
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	P	L	R	M	S	Q

Gap = 1
nextPos = 2
insert (nextVal = table[3] = E)
while(2 > 0 && E < C) False X
table[2] = E

Comparison = 24
Displacement = 17

Step 29: NextPos = 4 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	P	L	R	M	S	Q

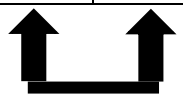


Gap = 1
nextPos = 4
insert (nextVal = table[4] = I)
while(4 > 0 && I < H) False X
table[4] = I

Comparison = 25
Displacement = 17

Step 30: NextPos = 5 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	P	L	R	M	S	Q



Gap = 1
nextPos = 5
insert (nextVal = table[5] = K)
while(5 > 0 && K < I) False X
table[5] = K

Comparison = 26
Displacement = 17

Step 31: NextPos = 6 in for loop (sort method)

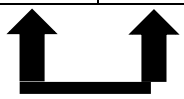
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	P	L	R	M	S	Q

Gap = 1
nextPos = 6
insert (nextVal = table[6] = P)
while(6 > 0 && P < K) False X
table[6] = P

Comparison = 27
Displacement = 17

Step 32: NextPos = 7 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	P	L	R	M	S	Q



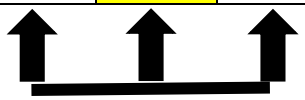
Gap = 1
nextPos = 7
insert (nextVal = table[7] = L)
while(7 > 0 && L < P) True
table[7] = P
nextPos = 6

Comparison = 28
Displacement = 19

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	P	P	R	M	S	Q

Step 33: NextPos = 6 in while loop (insert method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	L	P	R	M	S	Q

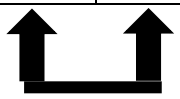


Gap = 1
nextPos = 6
insert (nextVal = table[7] = L)
while(6 > 0 && L < K) False X
table[6] = L

Comparison = 29
Displacement = 20

Step 34: NextPos = 8 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	L	P	R	M	S	Q

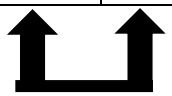


Gap = 1
nextPos = 8
insert (nextVal = table[8] = R)
while(8 > 0 && R < P) False X
table[8] = R

Comparison = 30
Displacement = 20

Step 35: NextPos = 9 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	L	P	R	M	S	Q



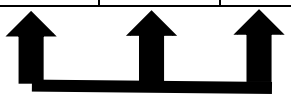
Gap = 1
nextPos = 9
insert (nextVal = table[9] = M)
while(9 > 0 && M < R) True
table[9] = M
nextPos = 8

Comparison = 31
Displacement = 21

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	L	P	R	R	S	Q

Step 36: NextPos = 8 in while loop (insert method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	L	P	R	R	S	Q



Gap = 1
nextPos = 8
insert (nextVal = table[9] = M)
while(8 > 0 && M < P) True
table[8] = P
nextPos = 7

Comparison = 32
Displacement = 22

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	L	P	P	R	S	Q

Step 37: NextPos = 7 in while loop (insert method)

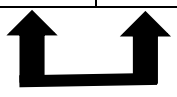
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	L	M	P	R	S	Q

Gap = 1
nextPos = 7
insert (nextVal = table[9] = M)
while(7 > 0 && M < L) False X
table[7] = M

Comparison = 33
Displacement = 23

Step 38: NextPos = 10 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	L	M	P	R	S	Q

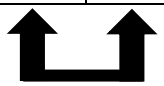


Gap = 1
nextPos = 10
insert (nextVal = table[10] = S)
while(10 > 0 && S < R) False X
table[10] = S

Comparison = 34
Displacement = 23

Step 39: NextPos = 11 in for loop (sort method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	L	M	P	R	S	Q



Gap = 1
nextPos = 11
insert (nextVal = table[11] = Q)
while(11 > 0 && Q < S) True
table[11] = S
nextPos = 10

Comparison = 35
Displacement = 24

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	L	M	P	R	S	S

Step 40: NextPos = 10 in while loop (insert method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	L	M	P	R	S	S

Gap = 1
nextPos = 10
insert (nextVal = table[11] = Q)
while(10 > 0 && Q < R) True
table[10] = R
nextPos = 9

Comparison = 36
Displacement = 25

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	L	M	P	R	R	S

Step 41: NextPos = 9 in while loop (insert method)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	L	M	P	Q	R	S

Gap = 1
nextPos = 9
insert (nextVal = table[11] = Q)
while(9 > 0 && Q < P) False X
table[9] = Q

Comparison = 37
Displacement = 26

Step 42: NextPos reached the length array. For loop and sorting is done. So the last counters and sorted array is below :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	L	M	P	Q	R	S

Comparison = 37
Displacement = 26

Sorting 2.a Array A (Integer Array From Small To Large-10 Elements) – Merge Sort Algorithm

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

I will use the Merge Sort Algorithm in book :

```
public <T extends Comparable<T>> void sort(T[] table) {
        if (table.length > 1) {
                int halfSize = table.length / 2;
                T[] leftTable = (T[]) new Comparable[halfSize];
                T[] rightTable =
                (T[]) new Comparable[table.length - halfSize];
                System.arraycopy(table, 0, leftTable, 0, halfSize);
                System.arraycopy(table, halfSize, rightTable, 0, table.length - halfSize);
                sort(leftTable);
                sort(rightTable);
                merge(table, leftTable, rightTable);
        }
}
private static <T extends Comparable<T>> void merge(T[] outputSequence,
        T[] leftSequence, T[] rightSequence) {
                int i = 0; // Index into the left input sequence.
                int j = 0; // Index into the right input sequence.
                int k = 0; // Index into the output sequence.
                while (i < leftSequence.length && j < rightSequence.length) {
                        if (leftSequence[i].compareTo(rightSequence[j]) < 0) {
                                outputSequence[k++] = leftSequence[i++];
                        } else {
                                outputSequence[k++] = rightSequence[j++];
                        }
                }
                while (i < leftSequence.length) {
                        outputSequence[k++] = leftSequence[i++];
                }
                while (j < rightSequence.length) {
                        outputSequence[k++] = rightSequence[j++];
                }
}
```

Step 1 :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

This is the left array of Array A

[0]	[1]	[2]	[3]	[4]
12	24	34	47	51

This is the right array of Array A

[0]	[1]	[2]	[3]	[4]
65	74	83	98	109

Step 2 :

This is the left array of above left array

[0]	[1]
12	24

This is the left array of above left array

[0]	[1]	[2]
34	47	51

Step 3 :

Left Array

[0]
12

Right Array

[0]
24

Step 4 : **Now in merge method, i = j = k = 0.**
While (i < left.length && j < right.length)
(Compare left[0] < right[0]) True
output[0] = 12 (i++)(k++)

Comparison = 1
Displacement = 0

Output :

12

Step 5 : **In merge method, i = k = 1 , j = 0**
While (i < left.length && j < right.length) Exit
While(i < left.length) Exit
While(j < right.length)
Output[1] = 24

Comparison = 1
Displacement = 0

Output :

[0]	[1]
12	24

Step 6 :

Left Array

[0]
34

Right Array

[0]	[1]
47	51

Step 7 :

Left Array

[0]
47

Right Array

[0]
51

Step 8 : Now in merge method, i = j = k = 0. (Left Length = Right Length = 1)

While (i < left.length && j < right.length) Comparison = 2

(Compare left[0] < right[0]) True Displacement = 0

output[0] = 47 (i++)(k++)

Output :

47

Step 9 : In merge method, i = k = 1 , j = 0

While (i < left.length && j < right.length) Exit Comparison = 2

While(i < left.length) Exit Displacement = 0

While(j < right.length)

Output[1] = 51

Output :

[0]	[1]
47	51

Step 10 :

Left Array

[0]
34

Right Array

[0]	[1]
47	51

Now in merge method, i = j = k = 0. (Left Length = 1, Right Length = 2)

While (i < left.length && j < right.length) Comparison = 3

(Compare left[0] < right[0]) True Displacement = 0

output[0] = 34 (i++)(k++)

Output :

[0]

34

Step 11 : In merge method, i = k = 1 , j = 0 (Left Length = 1, Right Length = 2)
While (i < left.length && j < right.length) Exit Comparison = 3
While(i < left.length) Exit Displacement = 0
While(j < right.length)
Output[1] = 47 (j++)(k++)

Output :

[0]	[1]
34	47

Step 12 : In merge method, i = 1 ,k = 2 , j = 1 (Left Length = 1, Right Length = 2)
While (i < left.length && j < right.length) Exit Comparison = 3
While(i < left.length) Exit Displacement = 0
While(j < right.length)
Output[2] = 51

Output :

[0]	[1]	[2]
34	47	51

Step 13 :

[0]	[1]
12	24

[0]	[1]	[2]
34	47	51

Now in merge method, i = j = k = 0. (Left Length = 2, Right Length = 3)
While (i < left.length && j < right.length) Comparison = 4
(Compare left[0] < right[0]) True Displacement = 0
output[0] = 12 (i++)(k++)

Output :

[0]
12

Step 14 :

[0]	[1]
12	24

[0]	[1]	[2]
34	47	51

In merge method, i = k = 1 , j = 0
While (i < left.length && j < right.length) Comparison = 5
(Compare left[1] < right[0]) True Displacement = 0
output[1] = 24 (i++)(k++)

Output :

[0]	[1]
12	24

Step 15 :

[0]	[1]
12	24

[0]	[1]	[2]
34	47	51

In merge method, i = k = 2 , j = 0
While (i < left.length && j < right.length) Exit **Comparison = 5**
While(i < left.length) Exit **Displacement = 0**
While(j < right.length)
Output[2] = 34 (k++)(j++)

Output :

[0]	[1]	[2]
12	24	34

Step 16 :

[0]	[1]
12	24

[0]	[1]	[2]
34	47	51

In merge method, i = 2 ,k = 3 , j = 1
While (i < left.length && j < right.length) Exit **Comparison = 5**
While(i < left.length) Exit **Displacement = 0**
While(j < right.length)
Output[3] = 47 (k++)(j++)

Output :

[0]	[1]	[2]	[3]
12	24	34	47

Step 17 :

[0]	[1]
12	24

[0]	[1]	[2]
34	47	51

In merge method, i = 2 ,k = 4 , j = 2
While (i < left.length && j < right.length) Exit **Comparison = 5**
While(i < left.length) Exit **Displacement = 0**
While(j < right.length)
Output[4] = 51 (k++)(j++)

Output :

[0]	[1]	[2]	[3]	[4]
12	24	34	47	51

Step 18 :

This is the left array of above left array

[0]	[1]
65	74

This is the left array of above left array

[0]	[1]	[2]
83	98	109

Step 19 :

Left Array

[0]
65

Right Array

[0]
74

Step 20 : Now in merge method, i = j = k = 0. (Left Length = Right Length = 1)
While (i < left.length && j < right.length) **Comparison = 6**
(Compare left[0] < right[0]) True **Displacement = 0**
output[0] = 65 (i++)(k++)

Output :

[0]
65

Step 21 : In merge method, i = k = 1 , j = 0
While (i < left.length && j < right.length) Exit **Comparison = 6**
While(i < left.length) Exit **Displacement = 0**
While(j < right.length)
Output[1] = 74

Output :

[0]	[1]
65	74

Step 22 :

Left Array	Right Array
[0]	[0]
83	98 | 109

Step 23 :

Left Array	Right Array
[0]	[0]
98	109

Step 24 : Now in merge method, i = j = k = 0. (Left Length = Right Length = 1)

While (i < left.length && j < right.length) — Comparison = 7

(Compare left[0] < right[0]) True — Displacement = 0

output[0] = 98 (i++)(k++)

	[0]
Output :	98

Step 25 : In merge method, i = k = 1 , j = 0

While (i < left.length && j < right.length) Exit — Comparison = 7

While(i < left.length) Exit — Displacement = 0

While(j < right.length)

Output[1] = 109

	[0]	[1]
Output :	98	109

Step 26 :

Left Array	Right Array
[0]	[0] [1]
83	98 | 109

Now in merge method, i = j = k = 0. (Left Length = 1, Right Length = 2)

While (i < left.length && j < right.length) — Comparison = 8

(Compare left[0] < right[0]) True — Displacement = 0

output[0] = 83 (i++)(k++)

	[0]
Output :	83

Step 27 :

Left Array

[0]
83

Right Array

[0]	[1]
98	109

In merge method, i = k = 1 , j = 0 (Left Length = 1, Right Length = 2)
While (i < left.length && j < right.length) Exit **Comparison = 8**
While(i < left.length) Exit **Displacement = 0**
While(j < right.length)
Output[1] = 98 (k++)(j++)

Output :

[0]	[1]
83	98

Step 28 : In merge method, i = 1, k = 2 , j = 1 (Left Length = 1, Right Length = 2)
While (i < left.length && j < right.length) Exit **Comparison = 8**
While(i < left.length) Exit **Displacement = 0**
While(j < right.length)
Output[2] = 109 (k++)(j++)

Output :

83	98	109

Step 29:

[0]	[1]
65	74

[0]	[1]	[2]
83	98	109

Now in merge method, i = j = k = 0. (Left Length = 2, Right Length = 3)
While (i < left.length && j < right.length) **Comparison = 9**
(Compare left[0] < right[0]) True **Displacement = 0**
output[0] = 65 (i++)(k++)

Output :

[0]
65

Step 30:

[0]	[1]
65	74

[0]	[1]	[2]
83	98	109

Now in merge method, i = k = 1, j = 0. (Left Length = 2, Right Length = 3)
While (i < left.length && j < right.length) Comparison = 10
(Compare left[1] < right[0]) True Displacement = 0
output[1] = 74 (i++)(k++)

Output :

[0]	[1]
65	74

Step 31 :

[0]	[1]
65	74

[0]	[1]	[2]
83	98	109

In merge method, i = 2, k = 2 , j = 0
While (i < left.length && j < right.length) Exit Comparison = 10
While(i < left.length) Exit Displacement = 0
While(j < right.length)
Output[2] = 83 (k++)(j++)

Output :

[0]	[1]	[2]
65	74	83

Step 32 :

[0]	[1]
65	74

[0]	[1]	[2]
83	98	109

In merge method, i = 2, k = 3 , j = 1
While (i < left.length && j < right.length) Exit Comparison = 10
While(i < left.length) Exit Displacement = 0
While(j < right.length)
Output[3] = 98 (k++)(j++)

Output :

[0]	[1]	[2]	[3]
65	74	83	98

Step 33 :

[0]	[1]
65	74

[0]	[1]	[2]
83	98	109

In merge method, i = 2, k = 4 , j = 1
While (i < left.length && j < right.length) Exit
While(i < left.length) Exit
While(j < right.length)
Output[4] = 109 (k++)(j++)

Comparison = 10
Displacement = 0

Output :

[0]	[1]	[2]	[3]	[4]
65	74	83	98	109

Step 34 :

[0]	[1]	[2]	[3]	[4]
12	24	34	47	51

[0]	[1]	[2]	[3]	[4]
65	74	83	98	109

Now in merge method, i = j = k = 0. (Left Length = 5, Right Length = 5)
While (i < left.length && j < right.length)
(Compare left[0] < right[0]) True
output[0] = 12 (i++)(k++)

Comparison = 11
Displacement = 0

Output :

[0]
12

Step 35 :

[0]	[1]	[2]	[3]	[4]
12	24	34	47	51

[0]	[1]	[2]	[3]	[4]
65	74	83	98	109

Now in merge method, i = k = 1 , j = 0. (Left Length = 5, Right Length = 5)
While (i < left.length && j < right.length)
(Compare left[1] < right[0]) True
output[1] = 24 (i++)(k++)

Comparison = 12
Displacement = 0

Output :

[0]	[1]
12	24

Step 36:

[0]	[1]	[2]	[3]	[4]
12	24	34	47	51

[0]	[1]	[2]	[3]	[4]
65	74	83	98	109

Now in merge method, i = k = 2 , j = 0. (Left Length = 5, Right Length = 5)
While (i < left.length && j < right.length) Comparison = 13
(Compare left[1] < right[0]) True Displacement = 0
output[2] = 34 (i++)(k++)

Output :

[0]	[1]	[2]
12	24	34

Step 37 :

[0]	[1]	[2]	[3]	[4]
12	24	34	47	51

[0]	[1]	[2]	[3]	[4]
65	74	83	98	109

Now in merge method, i = k = 2 , j = 0. (Left Length = 5, Right Length = 5)
While (i < left.length && j < right.length) Comparison = 14
(Compare left[3] < right[0]) True Displacement = 0
output[3] = 47 (i++)(k++)

Output :

[0]	[1]	[2]	[3]
12	24	34	47

Step 38 :

[0]	[1]	[2]	[3]	[4]
12	24	34	47	51

[0]	[1]	[2]	[3]	[4]
65	74	83	98	109

Now in merge method, i = k = 4 , j = 0. (Left Length = 5, Right Length = 5)
While (i < left.length && j < right.length) Comparison = 15
(Compare left[4] < right[0]) True Displacement = 0
output[4] = 51 (i++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]
12	24	34	47	51

Step 39 :

[0]	[1]	[2]	[3]	[4]
12	24	34	47	51

[0]	[1]	[2]	[3]	[4]
65	74	83	98	109

Now in merge method, i = k = 5 , j = 0. (Left Length = 5, Right Length = 5)
While (i < left.length && j < right.length) Exit Comparison = 15
While(i < left.length) Exit Displacement = 0
While(j < right.length)
output[5] = 65 (j++)(k++)

Step 40 :

[0]	[1]	[2]	[3]	[4]
12	24	34	47	51

[0]	[1]	[2]	[3]	[4]
65	74	83	98	109

Now in merge method, i =5 , k = 6 , j = 1. (Left Length = 5, Right Length = 5)
While (i < left.length && j < right.length) Exit Comparison = 15
While(i < left.length) Exit Displacement = 0
While(j < right.length)
output[6] = 74 (j++)(k++)

Step 41 :

[0]	[1]	[2]	[3]	[4]
12	24	34	47	51

[0]	[1]	[2]	[3]	[4]
65	74	83	98	109

Now in merge method, i =5 , k = 7 , j = 2. (Left Length = 5, Right Length = 5)
While (i < left.length && j < right.length) Exit Comparison = 15
While(i < left.length) Exit Displacement = 0
While(j < right.length)
output[7] = 83 (j++)(k++)

Step 42 :

[0]	[1]	[2]	[3]	[4]
12	24	34	47	51

[0]	[1]	[2]	[3]	[4]
65	74	83	98	109

Now in merge method, i =5 , k = 8 , j = 3. (Left Length = 5, Right Length = 5)
While (i < left.length && j < right.length) Exit Comparison = 15
While(i < left.length) Exit Displacement = 0
While(j < right.length)
output[8] = 98 (j++)(k++)

Step 43 :

[0]	[1]	[2]	[3]	[4]
12	24	34	47	51

[0]	[1]	[2]	[3]	[4]
65	74	83	98	109

Now in merge method, i =5 , k = 9 , j =4. (Left Length = 5, Right Length = 5)
While (i < left.length && j < right.length) Exit **Comparison = 15**
While(i < left.length) Exit **Displacement = 0**
While(j < right.length)
output[9] = 109 (j++)(k++)

Step 44 :
The last counters and sorted array is below :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

Comparison = 15
Displacement = 0

Sorting 2.b Array B (Integer Array From Largo To Small-10 Elements) – Merge Sort Algorithm

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
982	842	731	654	549	439	384	264	152	46

Step 1 :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
982	842	731	654	549	439	384	264	152	46

This is the left array of Array A

[0]	[1]	[2]	[3]	[4]
982	842	731	654	549

This is the right array of Array A

[0]	[1]	[2]	[3]	[4]
439	384	264	152	46

Step 2 :

This is the left array of above left array

[0]	[1]
982	842

This is the left array of above left array

[0]	[1]	[2]
731	654	549

Step 3 :

Left Array

[0]
982

Right Array

[0]
842

Step 4 : Now in merge method, i = j = k = 0. (Left Length = Right Length = 1)
While (i < left.length && j < right.length) Comparison = 1
(Compare left[0] < right[0]) False Displacement = 1
Else output[0] = > rightArray[0] = 842 (j++)(k++)

Output :

842	842

Step 5 : In merge method, j = k = 1 , i = 0 (Left Length = Right Length = 1)
While (i < left.length && j < right.length) Exit Comparison = 1
While(i < left.length) Displacement = 2
Output[1] = > leftArray[0] = 982

[0] [1]

Output :

842	982

Step 6 :

Left Array

[0]

731

Right Array

[0] [1]

654	549

Step 7 :

Left Array

[0]

654

Right Array

[0]

549

Step 8 : Now in merge method, i = j = k = 0. (Left Length = Right Length = 1)

While (i < left.length && j < right.length) Comparison = 2

(Compare left[0] < right[0]) False Displacement = 3

Else output[0] = > rightArray[0] = 549 (j++)(k++)

Output :

549	654

Step 9 : In merge method, j = k = 1 , i = 0

While (i < left.length && j < right.length) Exit Comparison = 2

While(i < left.length) Displacement = 4

Output[1] = > leftArray[0] = 654

[0] [1]

Output :

549	654

Step 10 :

Left Array

[0]

731

Right Array

[0] [1]

549	654

Now in merge method, i = j = k = 0. (Left Length = 1, Right Length = 2)

While (i < left.length && j < right.length) Comparison = 3

(Compare left[0] < right[0]) False Displacement = 5

Else output[0] => rightArray[0] = 549(j++)(k++)

[0]

Output :

549	549	654

Step 11 : In merge method, j = k = 1 , i = 0

While (i < left.length && j < right.length) Exit — Comparison = 4

(Compare left[0] < right[1]) False — Displacement = 6

Else output[1] => rightArray[1] = 654 (j++)(k++)

Output :

[0]	[1]	[2]
549	654	654

Step 12 : In merge method, i = 0 ,k = 2 , j = 2

While (i < left.length && j < right.length) Exit — Comparison = 4

While(i < left.length) — Displacement = 7

output[2] => leftArray[0] = 731 (i++)(k++)

Output :

[0]	[1]	[2]
549	654	731

Step 13 :

[0]	[1]
842	982

[0]	[1]	[2]
549	654	731

Now in merge method, i = j = k = 0. (Left Length = 2, Right Length = 3)

While (i < left.length && j < right.length) — Comparison = 5

(Compare left[0] < right[0]) False — Displacement = 8

Else output[0] => rightArray[0] = 549 (j++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]
549	982	731	654	549

Step 14 :

[0]	[1]
842	982

[0]	[1]	[2]
549	654	731

In merge method, j = k = 1 , i = 0

While (i < left.length && j < right.length) — Comparison = 6

(Compare left[0] < right[1]) False — Displacement = 9

Else output[1] => rightArray[1] = 654 (j++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]
549	654	731	654	549

Step 15 :

[0]	[1]
842	982

[0]	[1]	[2]
549	654	731

In merge method, j = k = 2 , i = 0
While (i < left.length && j < right.length) Comparison = 7
(Compare left[0] < right[2]) False Displacement = 9
Else output[2] => right[2] = 731 (j++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]
549	654	731	654	549

Step 16 :

[0]	[1]
842	982

[0]	[1]	[2]
549	654	731

In merge method, j = k = 3 , i = 0
While (i < left.length && j < right.length) Exit Comparison = 7
While(i <left.length) Displacement = 10
Else output[3] => left[0] = 842 (i++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]
549	654	731	842	549

Step 17 :

[0]	[1]
842	982

[0]	[1]	[2]
549	654	731

In merge method,i =1, j = 3 ,k = 4
While (i < left.length && j < right.length) Exit Comparison = 7
While(i < left.length) Displacement = 11
Output[4] => left[1] = 982 (k++)(i++)

Output :

[0]	[1]	[2]	[3]	[4]
549	654	731	842	982

Step 18:

This is the left array of above left array

[0]	[1]
439	384

This is the left array of above left array

[0]	[1]	[2]
264	152	46

Step 19 :

Left Array

[0]
439

Right Array

[0]
384

Step 20 : Now in merge method, i = j = k = 0. (Left Length = Right Length = 1)

While (i < left.length && j < right.length) Comparison = 8

(Compare left[0] < right[0]) False Displacement = 12

Else output[0] => right[0] = 384 (j++)(k++)

Output :

[0]	[1]
384	384

Step 21 : In merge method, j = k = 1 , i = 0

While (i < left.length && j < right.length) Exit Comparison = 9

While(i < left.length) Displacement = 13

Output[1] => left[0] = 439

Output :

[0]	[1]
384	439

Step 22 :

Left Array

[0]
264

Right Array

[0]	[1]
152	46

Step 23:

Left Array

[0]
152

Right Array

[0]
46

Step 24 : Now in merge method, i = j = k = 0. (Left Length = Right Length = 1)

While (i < left.length && j < right.length) — Comparison = 9

(Compare left[0] < right[0]) False — Displacement = 14

Else output[0] => right[0] = 46 (j++)(k++)

	[0]	[1]
Output :	46	46

Step 25 : In merge method, j = k = 1 , i = 0 (Left Length = Right Length = 1)

While (i < left.length && j < right.length) Exit — Comparison = 9

While(i < left.length) — Displacement = 15

Output[1] = 152

	[0]	[1]
Output :	46	152

Step 26 :

Left Array

[0]
264

Right Array

[0]	[1]
46	152

Now in merge method, i = j = k = 0. (Left Length = 1, Right Length = 2)

While (i < left.length && j < right.length) — Comparison = 10

(Compare left[0] < right[0]) False — Displacement = 16

Else output[0] => right[0] = 46 (j++)(k++)

	[0]	[1]	[2]
Output :	46	152	46

Step 27 :

Left Array

[0]
264

Right Array

[0]	[1]
46	152

In merge method, j = k = 1 , i = 0

While (i < left.length && j < right.length) — Comparison = 11

(Compare left[0] < right[1]) False — Displacement = 16

Else output[1] => right[1] = 152 (k++)(j++)

	[0]	[1]	[2]
Output :	46	152	46

Step 28 : In merge method, i = 0, k = 2 , j = 2

While (i < left.length && j < right.length) Exit — Comparison = 11

While(i < left.length) — Displacement = 20

Output[2] => left[0] = 264 (k++)(i++)

Output :

46	152	264

Step 29:

[0]	[1]
384	439

[0]	[1]	[2]
46	152	264

Now in merge method, i = j = k = 0. (Left Length = 2, Right Length = 3)

While (i < left.length && j < right.length) — Comparison = 11

(Compare left[0] < right[0]) False — Displacement = 21

Else output[0] => right[0] = 46 (j++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]
46	384	264	152	46

Step 30:

[0]	[1]
384	439

[0]	[1]	[2]
46	152	264

Now in merge method, j = k = 1, i = 0. (Left Length = 2, Right Length = 3)

While (i < left.length && j < right.length) — Comparison = 12

(Compare left[0] < right[1]) False — Displacement = 22

Else output[1] => right[1] = 152 (j++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]
46	152	264	152	46

Step 31 :

[0]	[1]
384	439

[0]	[1]	[2]
46	152	264

In merge method, j = 2, k = 2 , i = 0

While (i < left.length && j < right.length) — Comparison = 14

(Compare left[0] < right[2]) False — Displacement = 22

Else output[2] => right[2] = 264 (k++)(j++)

Output :

[0]	[1]	[2]	[3]	[4]
46	152	264	152	46

Step 32 :

[0]	[1]
384	439

[0]	[1]	[2]
46	152	264

In merge method, i = 0, k = 3 , j = 3
While (i < left.length && j < right.length) Exit
While(i < left.length) Exit
Output[3] = > left[0] (k++)(i++)

Comparison = 14
Displacement = 23

Output :

[0]	[1]	[2]	[3]	[4]
46	152	264	384	46

Step 33 :

[0]	[1]
384	439

[0]	[1]	[2]
46	152	264

In merge method, i = 1, k = 4 , j = 3
While (i < left.length && j < right.length) Exit
While(i < left.length)
output[4] => left[1] = 439 (k++)(i++)

Comparison = 14
Displacement = 25

Output :

[0]	[1]	[2]	[3]	[4]
46	152	264	384	439

Step 34 :

[0]	[1]	[2]	[3]	[4]
549	654	731	842	982

[0]	[1]	[2]	[3]	[4]
46	152	264	384	439

Now in merge method, i = j = k = 0. (Left Length = 5, Right Length = 5)
While (i < left.length && j < right.length)
(Compare left[0] < right[0]) False
Else output[0] => right[0] = 46 (j++)(k++)

Comparison = 15
Displacement = 26

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	842	731	654	549	439	384	264	152	46

Step 35 :

[0]	[1]	[2]	[3]	[4]
549	654	731	842	982

[0]	[1]	[2]	[3]	[4]
46	152	264	384	439

Now in merge method, j = k = 1 , i = 0. (Left Length = 5, Right Length = 5)
While (i < left.length && j < right.length) Comparison = 16
(Compare left[0] < right[1]) False Displacement = 27
Else output[1] => right[1] = 152 (j++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	731	654	549	439	384	264	152	46

Step 36 :

[0]	[1]	[2]	[3]	[4]
549	654	731	842	982

[0]	[1]	[2]	[3]	[4]
46	152	264	384	439

Now in merge method, j = k = 2 , i = 0. (Left Length = 5, Right Length = 5)
While (i < left.length && j < right.length) Comparison = 17
(Compare left[0] < right[2]) False Displacement = 28
output[2] => right[2] = 264 (j++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	654	549	439	384	264	152	46

Step 37 :

[0]	[1]	[2]	[3]	[4]
549	654	731	842	982

[0]	[1]	[2]	[3]	[4]
46	152	264	384	439

Now in merge method, j = k = 3 , i = 0. (Left Length = 5, Right Length = 5)
While (i < left.length && j < right.length) Comparison = 18
(Compare left[0] < right[3]) False Displacement = 29
Else output[3] => right[3] = 384 (j++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	549	439	384	264	152	46

Step 38 :

[0]	[1]	[2]	[3]	[4]
549	654	731	842	982

[0]	[1]	[2]	[3]	[4]
46	152	264	384	439

Now in merge method, j = k = 4 , i = 0. (Left Length = 5, Right Length = 5)
While (i < left.length && j < right.length) Comparison = 19
(Compare left[0] < right[4]) False Displacement = 30
Else output[4] => right[4] = 439 (j++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	439	384	264	152	46

Step 39 :

[0]	[1]	[2]	[3]	[4]
549	654	731	842	982

[0]	[1]	[2]	[3]	[4]
46	152	264	384	439

Now in merge method, j = k = 5 , i = 0. (Left Length = 5, Right Length = 5)
While (i < left.length && j < right.length) Exit Comparison = 19
While(i < left.length) Displacement = 30
output[5] => left[0] = 549 (i++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	439	384	264	152	46

Step 40 :

[0]	[1]	[2]	[3]	[4]
549	654	731	842	982

[0]	[1]	[2]	[3]	[4]
46	152	264	384	439

Now in merge method, j = 5 , k = 6 , i = 1. (Left Length = 5, Right Length = 5)
While (i < left.length && j < right.length) Exit Comparison = 19
While(i < left.length) Displacement = 31
output[6] => left[1] = 654 (i++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	439	654	264	152	46

Step 41 :

[0]	[1]	[2]	[3]	[4]
549	654	731	842	982

[0]	[1]	[2]	[3]	[4]
46	152	264	384	439

Now in merge method, j =5 , k = 7 , i = 2. (Left Length = 5, Right Length = 5)
While (i < left.length && j < right.length) Exit Comparison = 19
While(i < left.length) Displacement = 32
output[7] => left[2] = 731 (i++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	439	654	731	152	46

Step 42 :

[0]	[1]	[2]	[3]	[4]
549	654	731	842	982

[0]	[1]	[2]	[3]	[4]
46	152	264	384	439

Now in merge method, j =5 , k = 8 , i = 3. (Left Length = 5, Right Length = 5)
While (i < left.length && j < right.length) Exit Comparison = 19
While(i < left.length) Displacement = 33
output[8] => left[3] = 842 (i++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	439	654	731	842	46

Step 43 :

[0]	[1]	[2]	[3]	[4]
549	654	731	842	982

[0]	[1]	[2]	[3]	[4]
46	152	264	384	439

Now in merge method, j =5 , k = 9 , i = 4. (Left Length = 5, Right Length = 5)
While (i < left.length && j < right.length) Exit Comparison = 19
While(i < left.length) Displacement = 34
output[9] => left[4] = 982 (i++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	439	654	731	842	982

Step 44:

The last counters and sorted array is below :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	549	654	731	842	982

Comparison = 19

Displacement = 34

Sorting 2.c Array C (Integer Array - 12 Elements) – Merge Sort Algorithm

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
5	2	13	9	1	7	6	8	1	15	4	11

Step 1 :

This is the left array of Array A

[0]	[1]	[2]	[3]	[4]	[5]
5	2	13	9	1	7

This is the right array of Array A

[6]	[7]	[8]	[9]	[10]	[11]
6	8	1	15	4	11

Step 2 :

This is the left array of above left array

[0]	[1]	[2]
5	2	13

This is the left array of above left array

[0]	[1]	[2]
9	1	7

Step 3 :

Left Array

[0]
5

Right Array

[0]	[1]
2	13

Step 4 :

Left Array

[0]
2

Right Array

[0]
13

Step 5 : Now in merge method, i = j = k = 0. (Left Length = Right Length = 1)

While (i < left.length && j < right.length) **Comparison = 1**

(Compare left[0] < right[0]) True **Displacement = 0**

Output :

2	13

Step 6 : In merge method, i = k = 1 , j = 0 (Left Length = Right Length = 1)
While (i < left.length && j < right.length) Exit — Comparison = 1
While(i < right.length) — Displacement = 0
Output[1] = > right[0] = 13 (j++)(k++)

Output :

[0]	[1]
2	13

Step 7:

Left Array

[0]
5

Right Array

[0]	[1]
2	13

Step 8 :

Left Array

[0]
5

Right Array

[0]	[1]
2	13

Step 9 : Now in merge method, i = j = k = 0. (Left Length = 1, Right Length = 2)
While (i < left.length && j < right.length) — Comparison = 2
(Compare left[0] < right[0]) False — Displacement = 1
Else output[0] => right[0] = 2 (k++)(j++)

Output :

[0]	[1]	[2]
2	2	13

Step 10 :

Left Array

[0]
5

Right Array

[0]	[1]
2	13

Now in merge method, i=0, j = k = 1. (Left Length = 1,Right Length = 2)
While (i < left.length && j < right.length) — Comparison = 3
(Compare left[0] < right[1]) True — Displacement = 2
output[1] = > left[0] = 2 (i++)(k++)

Output :

[0]	[1]	[2]
2	5	13

Step 11 :

Left Array [0]	Right Array [0]	[1]
5	2	13

Now in merge method, i =1, j = 1, k = 2. (Left Length = 1,Right Length = 2)
While (i < left.length && j < right.length) Exit Comparison = 3
While(i < right.length) Displacement = 2
output[2] = > right[1] = 2 (j++)(k++)

Output :

2	5	13

Step 12 :

Left Array [0]	Right Array [0]	[1]
9	1	7

Step 13 :

Left Array [0]	Right Array [0]
1	7

Step 14 : Now in merge method, i = j = k = 0. (Left Length = Right Length = 1)
While (i < left.length && j < right.length) Comparison = 4
(Compare left[0] < right[0]) True Displacement = 2
output[0] => left[0] = 1 (i++)(k++)

Output :

1	7

Step 15 : In merge method, i = k = 1 , j = 0 (Left Length = Right Length = 1)
While (i < left.length && j < right.length) Exit Comparison = 4
While(i < right.length) Displacement = 2
Output[1] = > right[0] = 7 (j++)(k++)

Output :

[0]	[1]
1	7

Step 16 :

Left Array [0]	Right Array [0]	[1]
9	1	7

Step 17 :

Left Array

[0]
9

Right Array

[0]	[1]
1	7

Step 18 : Now in merge method, i = j = k = 0. (Left Length = 1, Right Length = 2)

While (i < left.length && j < right.length) Comparison = 5

(Compare left[0] < right[0]) False Displacement = 3

Else output[0] => right[0] = 1 (k++)(j++)

Output :

[0]	[1]	[2]
1	1	7

Step 19 : Now in merge method, i=0, j = k = 1. (Left Length = 1,Right Length = 2)

While (i < left.length && j < right.length) Comparison = 6

(Compare left[0] < right[1]) False Displacement = 4

Else output[1] = > right[2] = 7 (j++)(k++)

Output :

[0]	[1]	[2]
1	7	7

Step 20 :

Left Array

[0]
9

Right Array

[0]	[1]
1	7

Now in merge method, i =0, j = 1, k = 2. (Left Length = 1,Right Length = 2)

While (i < left.length && j < right.length) Exit Comparison = 6

While(i < right.length) Displacement = 5

output[2] = > left[0] = 9 (i++)(k++)

Output :

1	7	9

Step 21 :

This is the left array of above left array

[0]	[1]	[2]
2	5	13

This is the left array of above left array

[0]	[1]	[2]
1	7	9

Step 22 : In merge method, j = k = 1 , i = 0 (Left Length = Right Length = 3)

While (i < left.length && j < right.length) Comparison = 7

(Compare left[0] < right[0]) False Displacement = 6

Else output[0] = > right[0] = 1 (j++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]	[5]
1	2	13	9	1	7

Step 23 : Now in merge method, i = 0, j = k = 1. (Left Length = Right Length = 3)

While (i < left.length && j < right.length) Comparison = 8

(Compare left[0] < right[1]) True Displacement = 6

Else output[1] = > left[0] = 2 (i++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]	[5]
1	2	13	9	1	7

Step 24 :

[0]	[1]	[2]
2	5	13

[0]	[1]	[2]
1	7	9

Now in merge method, i = 1, j = 1, k = 2. (Left Length = Right Length = 3)

While (i < left.length && j < right.length) Comparison = 9

(Compare left[1] < right[1]) True Displacement = 7

output[2] = > left[1] = 5 (i++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]	[5]
1	2	5	9	1	7

Step 25 :

[0]	[1]	[2]
2	5	13

[0]	[1]	[2]
1	7	9

Now in merge method, i = 2, j = 1,k = 3. (Left Length = Right Length = 3)

While (i < left.length && j < right.length) Comparison = 10

(Compare left[2] < right[1]) False Displacement = 8

Else output[3] = > right[1] = 7 (j++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]	[5]
1	2	5	7	1	7

Step 26 :

[0]	[1]	[2]
2	5	13

[0]	[1]	[2]
1	7	9

Now in merge method, i = 2, j = 2,k = 4. (Left Length = Right Length = 3)
While (i < left.length && j < right.length) Comparison = 11
(Compare left[2] < right[2]) False Displacement = 9
output[4] = > right[2] = 7 (j++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]	[5]
1	2	5	7	9	7

Step 27 :

[0]	[1]	[2]
2	5	13

[0]	[1]	[2]
1	7	9

Now in merge method, i = 2, j = 3,k = 5. (Left Length = Right Length = 3)
While (i < left.length && j < right.length) Exit Comparison = 11
While (i < left.length) Displacement = 10
output[4] = > left[2] = 13 (i++)(k++)

Output :

[0]	[1]	[2]	[3]	[4]	[5]
1	2	5	7	9	13

Step 28 :

[0]	[1]	[2]	[3]	[4]	[5]
6	8	1	15	4	11

Step 29 :

This is the left array of above left array

[0]	[1]	[2]
6	8	1

This is the left array of above left array

[0]	[1]	[2]
15	4	11

Step 30 :

Left Array

[0]
6

Right Array

[0]	[1]
8	1

Step 31 :

Left Array	Right Array
[0]	[0]
8	1

Step 32: Now in merge method, i = j = k = 0. (Left Length = Right Length = 1)

While (i < left.length && j < right.length) Comparison = 12

(Compare left[0] < right[0]) False Displacement = 11

Else output[0] = right[0] = 1 (j++)(k++)

Output :

1	1

Step 33 : In merge method, j = k = 1 , i = 0 (Left Length = Right Length = 1)

While (i < left.length && j < right.length) Exit Comparison = 12

While(i < left.length) Displacement = 12

Output[1] = > left[0] = 8 (i++)(k++)

Output :

[0]	[1]
1	8

Step 34 :

Left Array	Right Array	
[0]	[0]	[1]
6	1	8

Step 35 : Now in merge method, i = j = k = 0. (Left Length =1, Right Length = 2)

While (i < left.length && j < right.length) Comparison = 13

(Compare left[0] < right[0]) False Displacement = 13

Else output[0] => right[0] = 1 (k++)(j++)

Output :

[0]	[1]	[2]
1	8	1

Step 36 :

Left Array		Right Array	
[0]		[0]	[1]
6		1	8

Now in merge method, i = 0, j = k = 1. (Left Length =1, Right Length = 2)
While (i < left.length && j < right.length) Comparison = 14
(Compare left[0] < right[1]) True Displacement = 14
output[1] => left[0] = 6 (k++)(i++)

Output :

[0]	[1]	[2]
1	6	1

Step 37 : Now in merge method, i = 1, j = 1, k = 2. (Left Length =1, Right Length = 2)
While (i < left.length && j < right.length) Exit Comparison = 14
While(j < right.length) Displacement = 15
output[2] => right[1] = 8 (k++)(j++)

Output :

[0]	[1]	[2]
1	6	8

Step 38 :

Left Array		Right Array	
[0]		[0]	[1]
15		4	11

Step 39 :

Left Array	Right Array
[0]	[0]
4	11

Step 40 : Now in merge method, i = j = k = 0. (Left Length = Right Length = 1)
While (i < left.length && j < right.length) Comparison = 15
(Compare left[0] < right[0]) True Displacement = 15
output[0] = left[0] = 4 (i++)(k++)

Output :

4	11

Step 41 : In merge method, i = k = 1 , j = 0 (Left Length = Right Length = 1)
While (i < left.length && j < right.length) Exit Comparison = 15
While(i < right.length) Displacement = 15
Output[1] = > right[0] = 11 (i++)(k++)

Output :

[0]	[1]
4	11

Step 42 :

Left Array

[0]
15

Right Array

[0]	[1]
4	11

Step 43 :

Left Array

[0]
15

Right Array

[0]	[1]
4	11

Step 44 : Now in merge method, i = j = k = 0. (Left Length =1, Right Length = 2)
While (i < left.length && j < right.length) Comparison = 16
(Compare left[0] < right[0]) False Displacement = 16
Else output[0] => right[0] = 4 (k++)(j++)

Output :

[0]	[1]	[2]
4	4	11

Step 45 :

Left Array

[0]
15

Right Array

[0]	[1]
4	11

Step 46 : Now in merge method, i = 0, j = k = 1. (Left Length =1, Right Length = 2)
While (i < left.length && j < right.length) Comparison = 17
(Compare left[0] < right[1]) False Displacement = 17
Else output[1] => right[1] = 11 (k++)(j++)

Output :

[0]	[1]	[2]
4	11	11

Step 47 :

Left Array

[0]
15

Right Array

[0]	[1]
4	11

Now in merge method, i = 0, j = 2, k = 2. (Left Length =1, Right Length = 2)
While (i < left.length && j < right.length) Exit
While(j < left.length)
output[2] => left[0] = 15 (k++)(i++)

Comparison = 17
Displacement = 18

Output :

[0]	[1]	[2]
4	11	15

Step 48 :

[0]	[1]	[2]
1	6	8

[0]	[1]	[2]
4	11	15

Step 49 :

[0]	[1]	[2]
1	6	8

[0]	[1]	[2]
4	11	15

Output:

[0]	[1]	[2]	[3]	[4]	[5]
1	8	1	15	4	11

Comparison = 18
Displacement =19

Step 50 :

[0]	[1]	[2]
1	6	8

[0]	[1]	[2]
4	11	15

Output:

[0]	[1]	[2]	[3]	[4]	[5]
1	4	1	15	4	11

Comparison = 19
Displacement =20

Step 51 :

[0]	[1]	[2]
1	6	8

[0]	[1]	[2]
4	11	15

Output:

[0]	[1]	[2]	[3]	[4]	[5]
1	4	6	15	4	11

Comparison = 20
Displacement =21

Step 52 :

[0]	[1]	[2]
1	6	8

[0]	[1]	[2]
4	11	15

Output:

[0]	[1]	[2]	[3]	[4]	[5]
1	4	6	8	4	11

Comparison = 21
Displacement =22

Step 53 :

[0]	[1]	[2]
1	6	8

[0]	[1]	[2]
4	11	15

Output:

[0]	[1]	[2]	[3]	[4]	[5]
1	4	6	8	11	11

Comparison = 21
Displacement =23

Step 54 :

[0]	[1]	[2]
1	6	8

[0]	[1]	[2]
4	11	15

Output:

[0]	[1]	[2]	[3]	[4]	[5]
1	4	6	8	11	15

Comparison = 21
Displacement =24

Step 55: This is the left array of Array A

[0]	[1]	[2]	[3]	[4]	[5]
1	2	5	7	9	13

This is the right array of Array A

[6]	[7]	[8]	[9]	[10]	[11]
1	4	6	8	11	15

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	2	13	9	1	7	6	8	1	15	4	11

Comparison = 22
Displacement =25

Step56 : This is the left array of Array A

[0]	[1]	[2]	[3]	[4]	[5]
1	2	5	7	9	13

This is the right array of Array A

[6]	[7]	[8]	[9]	[10]	[11]
1	4	6	8	11	15

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	13	9	1	7	6	8	1	15	4	11

Comparison = 23
Displacement =26

Step 57: This is the left array of Array A

[0]	[1]	[2]	[3]	[4]	[5]
1	2	5	7	9	13

This is the right array of Array A

[6]	[7]	[8]	[9]	[10]	[11]
1	4	6	8	11	15

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	9	1	7	6	8	1	15	4	11

Comparison = 24
Displacement =28

Step 58: This is the left array of Array A

[0]	[1]	[2]	[3]	[4]	[5]
1	2	5	7	9	13

This is the right array of Array A

[6]	[7]	[8]	[9]	[10]	[11]
1	4	6	8	11	15

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	1	7	6	8	1	15	4	11

Comparison = 25
Displacement =29

Step 59: This is the left array of Array A

[0]	[1]	[2]	[3]	[4]	[5]
1	2	5	7	9	13

This is the right array of Array A

[6]	[7]	[8]	[9]	[10]	[11]
1	4	6	8	11	15

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	7	6	8	1	15	4	11

Comparison = 26
Displacement =30

Step 60: This is the left array of Array A

[0]	[1]	[2]	[3]	[4]	[5]
1	2	5	7	9	13

This is the right array of Array A

[6]	[7]	[8]	[9]	[10]	[11]
1	4	6	8	11	15

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	6	6	8	1	15	4	11

Comparison = 27
Displacement =31

Step 61: This is the left array of Array A

[0]	[1]	[2]	[3]	[4]	[5]
1	2	5	7	9	13

This is the right array of Array A

[6]	[7]	[8]	[9]	[10]	[11]
1	4	6	8	11	15

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	6	7	8	1	15	4	11

Comparison = 28
Displacement =32

Step 62: This is the left array of Array A

[0]	[1]	[2]	[3]	[4]	[5]
1	2	5	7	9	13

This is the right array of Array A

[6]	[7]	[8]	[9]	[10]	[11]
1	4	6	8	11	15

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	6	7	8	1	15	4	11

Comparison = 29
Displacement =33

Step 63: This is the left array of Array A

[0]	[1]	[2]	[3]	[4]	[5]
1	2	5	7	9	13

This is the right array of Array A

[6]	[7]	[8]	[9]	[10]	[11]
1	4	6	8	11	15

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	6	7	8	9	15	4	11

Comparison = 30
Displacement =34

Step 64: This is the left array of Array A

[0]	[1]	[2]	[3]	[4]	[5]
1	2	5	7	9	13

This is the right array of Array A

[6]	[7]	[8]	[9]	[10]	[11]
1	4	6	8	11	15

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	6	7	8	9	11	4	11

Comparison = 31
Displacement =35

Step 65: This is the left array of Array A

[0]	[1]	[2]	[3]	[4]	[5]
1	2	5	7	9	13

This is the right array of Array A

[6]	[7]	[8]	[9]	[10]	[11]
1	4	6	8	11	15

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	6	7	8	9	11	13	11

Comparison = 32
Displacement =36

Step 66:

[0]	[1]	[2]	[3]	[4]	[5]
1	2	5	7	9	13

[6]	[7]	[8]	[9]	[10]	[11]
1	4	6	8	11	15

Output :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	6	7	8	9	11	13	15

Comparison = 32
Displacement =37

Step 67 : The last counters and sorted array is below :
Comparison : 32
Displacement:37

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	6	7	8	9	11	13	15

Sorting 2.d Array D (Char Array - 12 Elements) – Merge Sort Algorithm

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
S	B	I	M	H	Q	C	L	R	E	P	K

Step 1 : This is the left array of Array A

[0]	[1]	[2]	[3]	[4]	[5]
S	B	I	M	H	Q

This is the right array of Array A

[6]	[7]	[8]	[9]	[10]	[11]
C	L	R	E	P	K

Step 2 :

[0]	[1]	[2]
S	B	I

[0]	[1]	[2]
M	H	Q

Step 3 :

Left Array

[0]
S

Right Array

[0]	[1]
B	I

Step 4 :

Left Array

[0]
B

Right Array

[0]
I

Step 5:

[0]
B

[0]
I

Output :

[0]	[1]
B	I

Comparison :1
Displacement :0

Step 6:

[0]
B

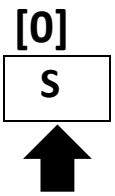
[0]
I

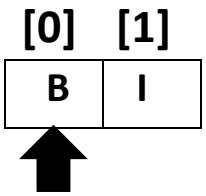
Output :

[0]	[1]
B	I

Comparison :2
Displacement :0

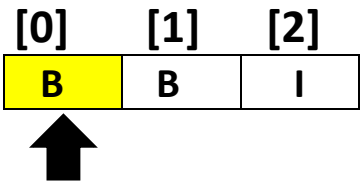
Step 7:



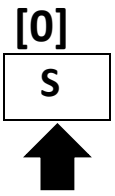


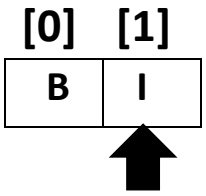
Output :

Comparison :3
Displacement : 1

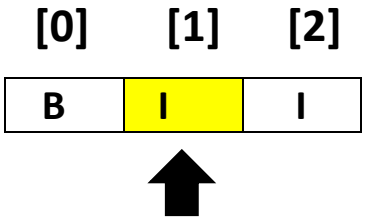


Step 8:



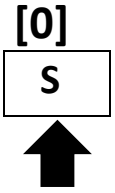


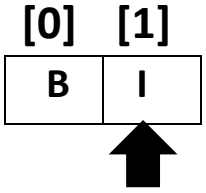
Output :



Comparison :4
Displacement :2

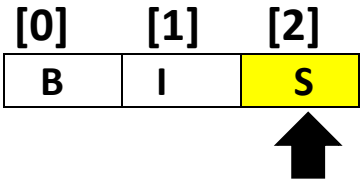
Step 9:





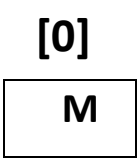
Output :

Comparison :4
Displacement :3

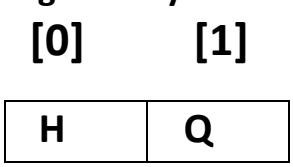


Step 10 :

Left Array



Right Array



Step 11 :

Left Array

[0]

H

Right Array

[0]

Q

Step 12:

[0]

H

[0]

Q

Output :

Comparison :5
Displacement :3

[0]	[1]
H	Q

Step 13:

[0]

H

[0]

Q

Output :

Comparison : 6
Displacement : 3

[0]	[1]
H	Q

Step 14:

[0]

M

[0] [1]

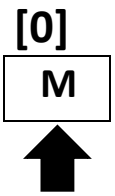
H	Q

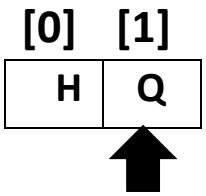
Output :

Comparison : 7
Displacement : 4

[0]	[1]	[2]
H	H	Q

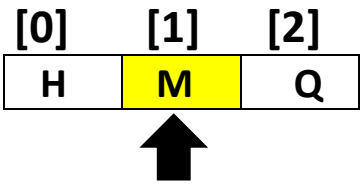
Step 15:



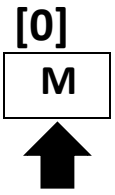


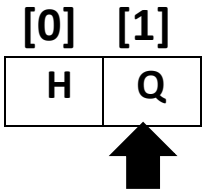
Output :

Comparison : 8
Displacement : 4



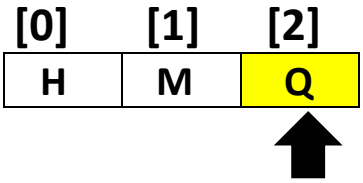
Step 16:



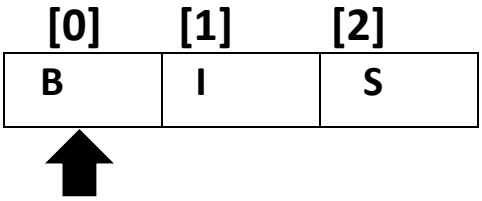


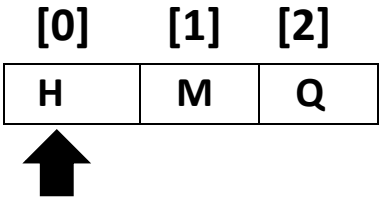
Output :

Comparison : 8
Displacement : 5



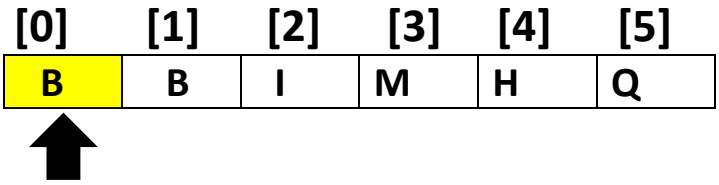
Step 17 :



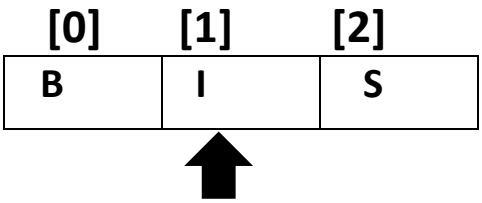


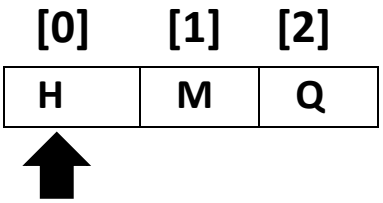
Output :

Comparison : 9
Displacement : 6



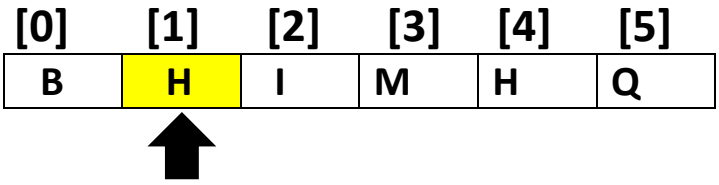
Step 18 :



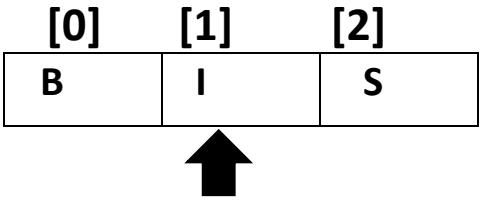


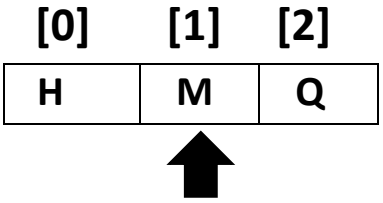
Output :

Comparison : 10
Displacement : 7



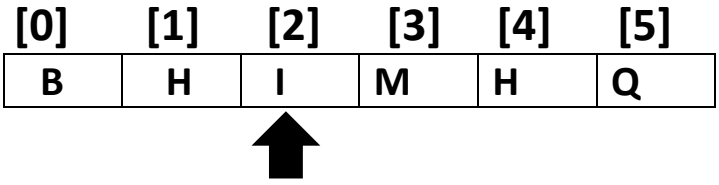
Step 19 :



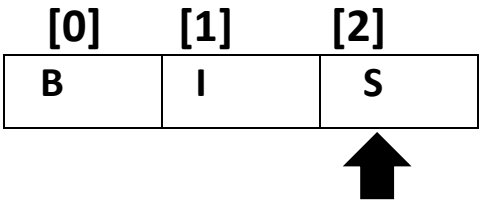


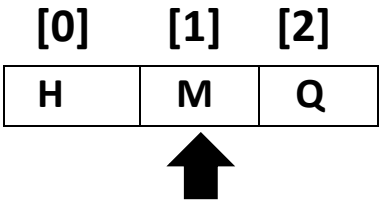
Output :

Comparison : 11
Displacement : 7



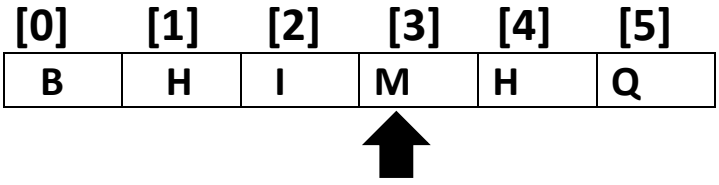
Step 20 :



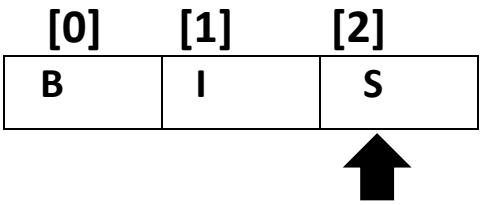


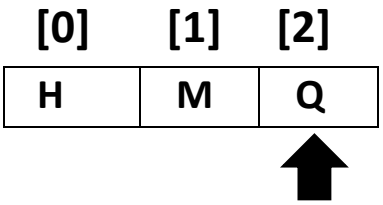
Output :

Comparison : 12
Displacement : 7



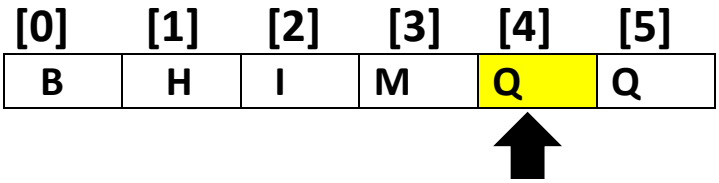
Step 21 :



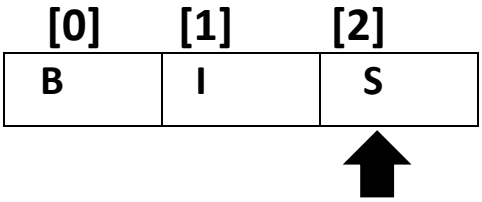


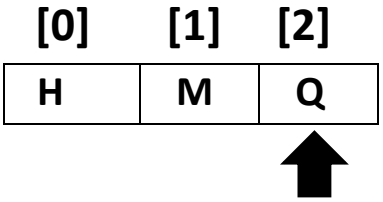
Output :

Comparison : 13
Displacement : 8



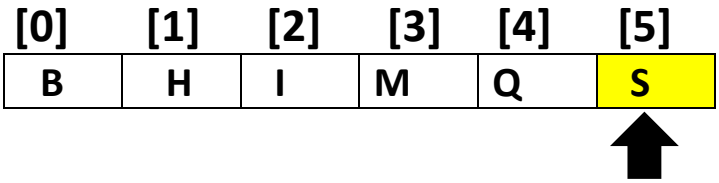
Step 22 :



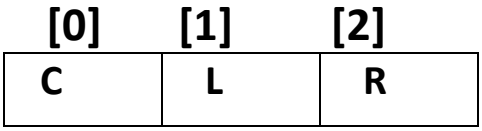


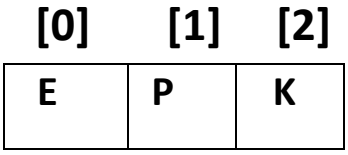
Output :

Comparison : 13
Displacement : 9



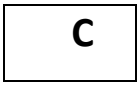
Step 23 :



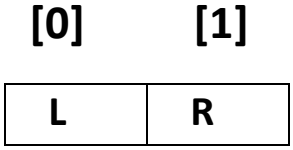


Step 24 :

Left Array



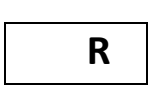
Right Array



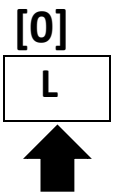
Step 25 :

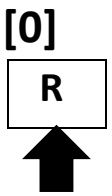
Left Array

Right Array



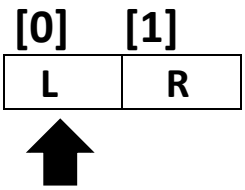
Step 26:



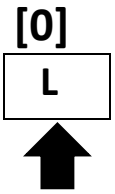


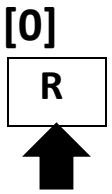
Output :

Comparison : 14
Displacement : 9



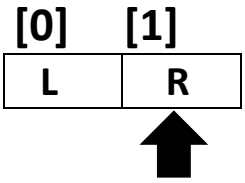
Step 27:



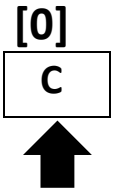


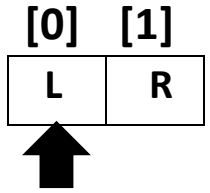
Output :

Comparison : 14
Displacement : 9



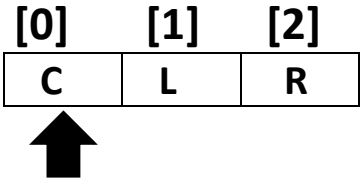
Step 28:



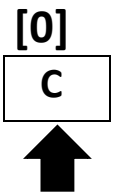


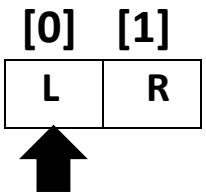
Output :

Comparison : 15
Displacement : 9



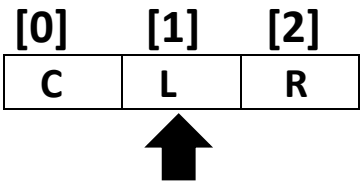
Step 30:



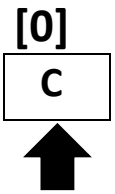


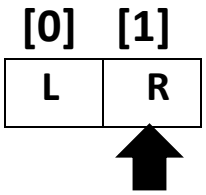
Output :

Comparison : 16
Displacement : 9



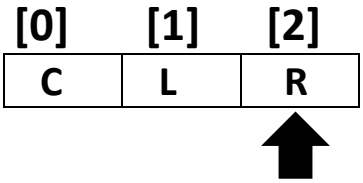
Step 31 :





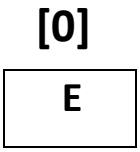
Output :

Comparison : 17
Displacement : 9

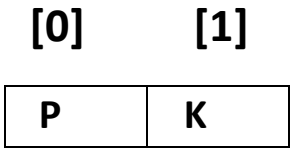


Step 32 :

Left Array

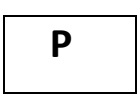


Right Array

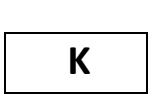


Step 33 :

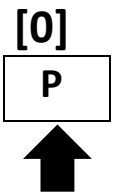
Left Array

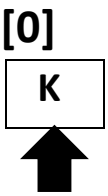


Right Array



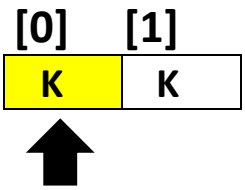
Step34 :



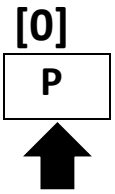


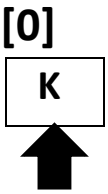
Output :

Comparison : 18
Displacement : 10



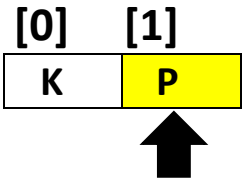
Step 35 :



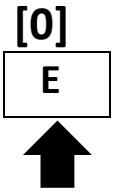


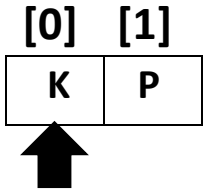
Output :

Comparison : 18
Displacement : 11



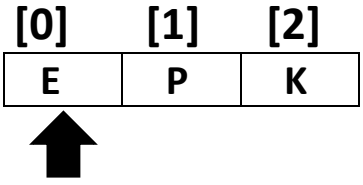
Step 36 :



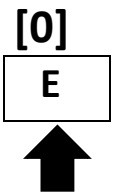


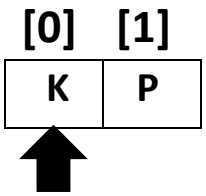
Output :

Comparison : 19
Displacement : 11



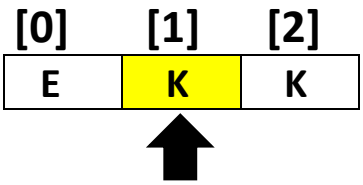
Step 37:



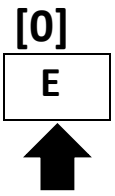


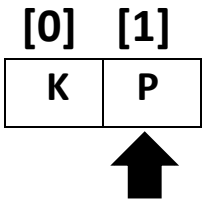
Output :

Comparison : 19
Displacement : 12



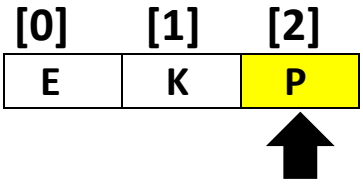
Step 38:



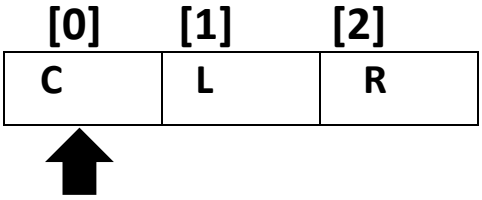


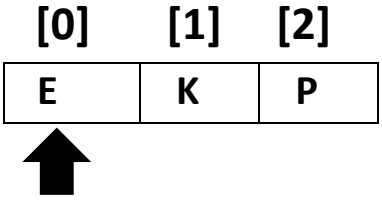
Output :

Comparison : 19
Displacement : 13



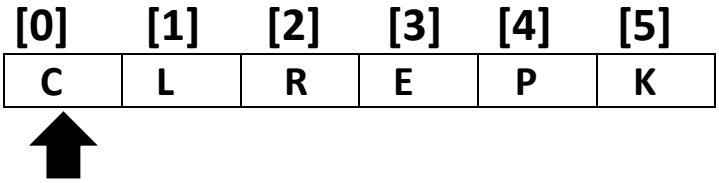
Step 39 :



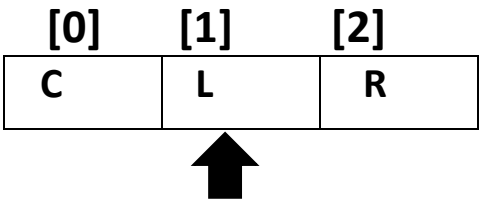


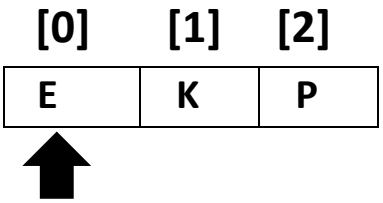
Output :

Comparison : 20
Displacement : 13



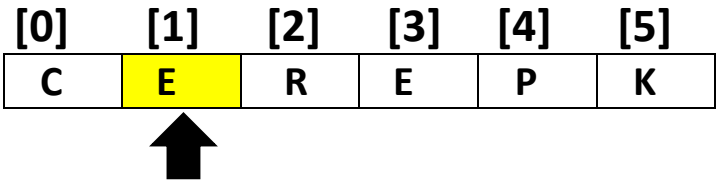
Step 40 :



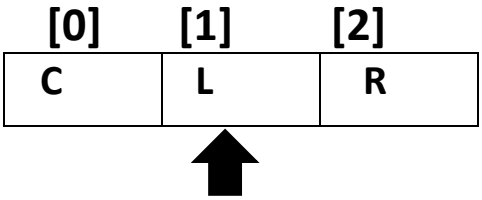


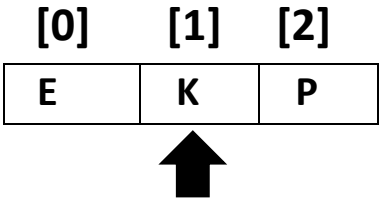
Output :

Comparison : 21
Displacement : 14



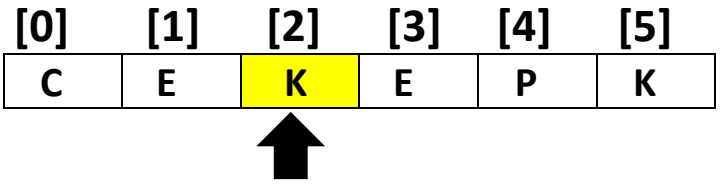
Step 41 :



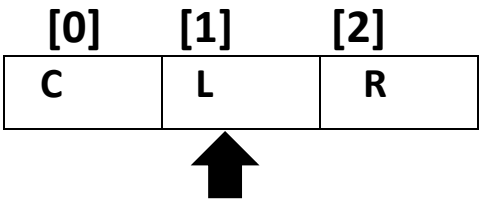


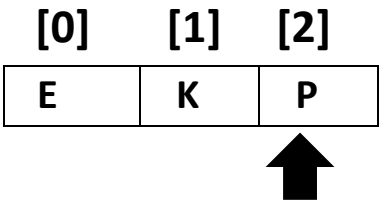
Output :

Comparison : 22
Displacement : 15



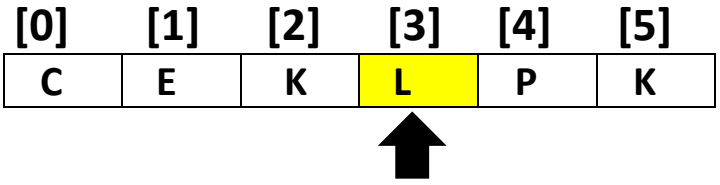
Step 42 :



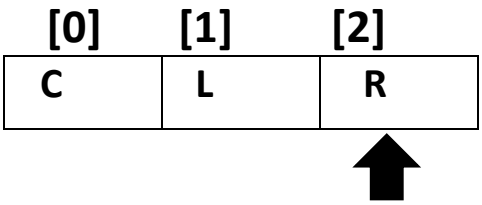


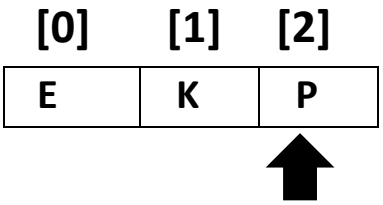
Output :

Comparison : 23
Displacement : 16



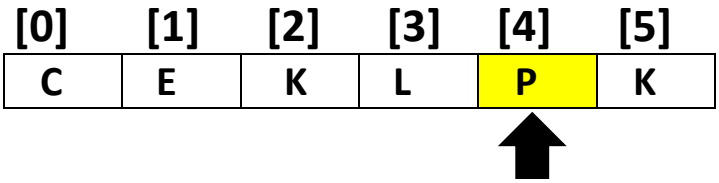
Step 43 :



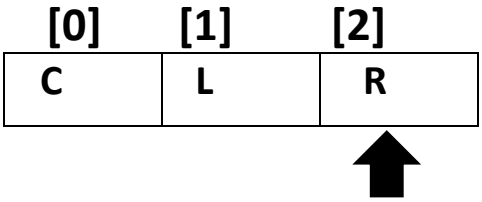


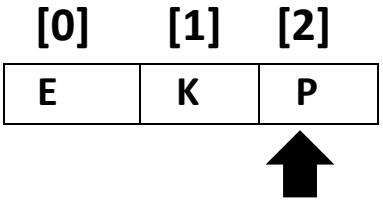
Output :

Comparison : 24
Displacement : 17



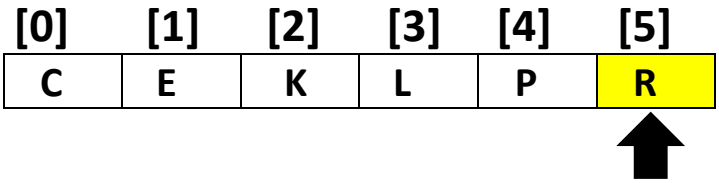
Step 44 :



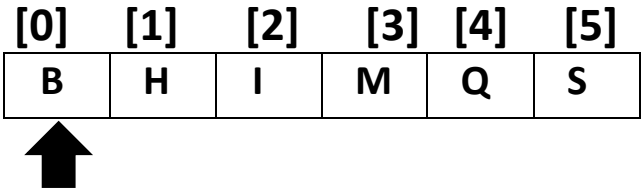


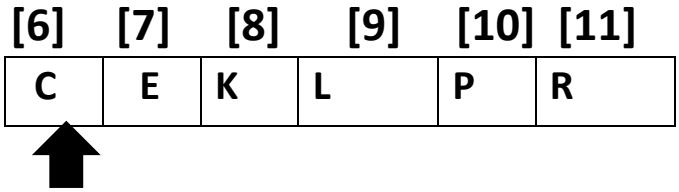
Output :

Comparison : 24
Displacement : 18



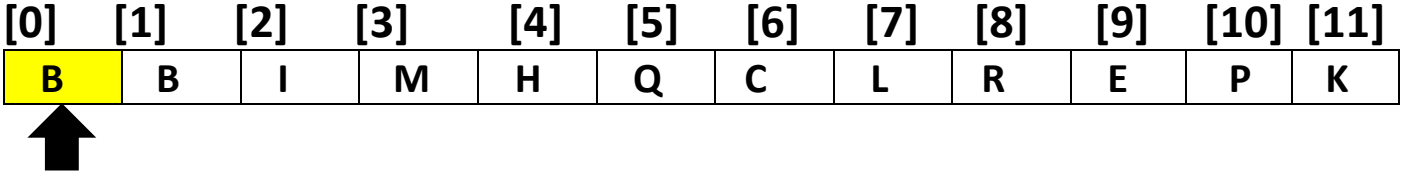
Step 45 :



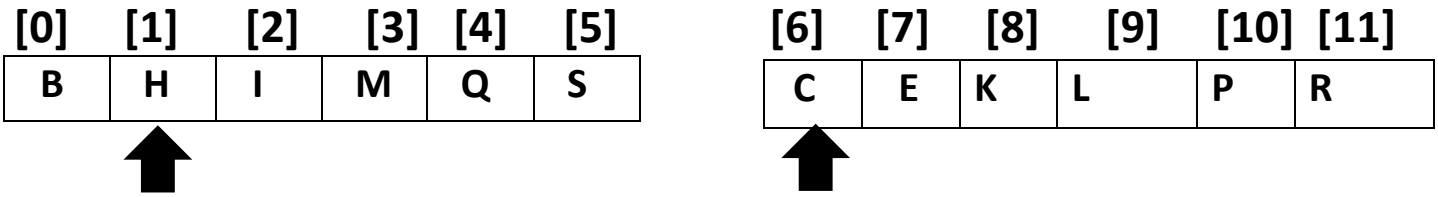


Comparison : 25
Displacement : 19

Output :

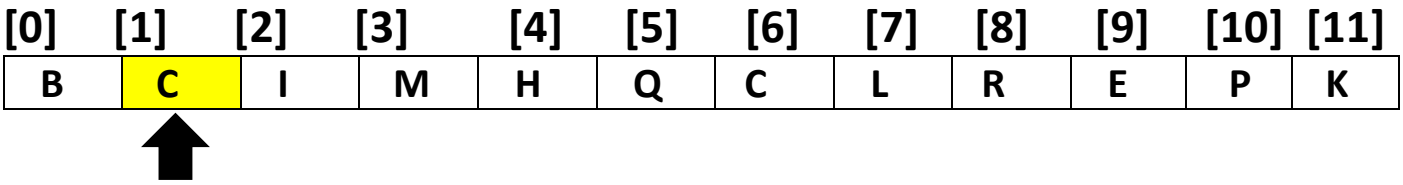


Step 46 :

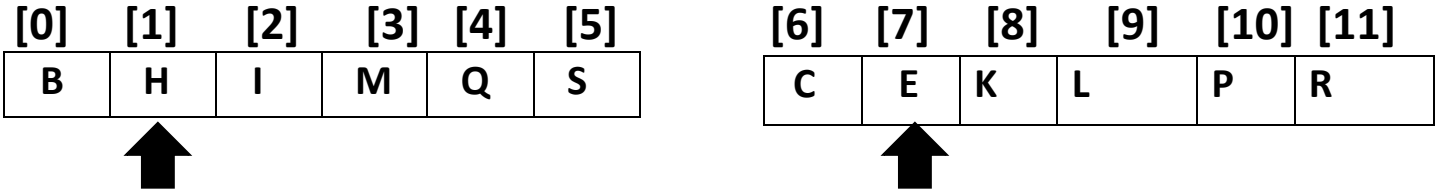


Output :

Comparison : 26
Displacement : 20

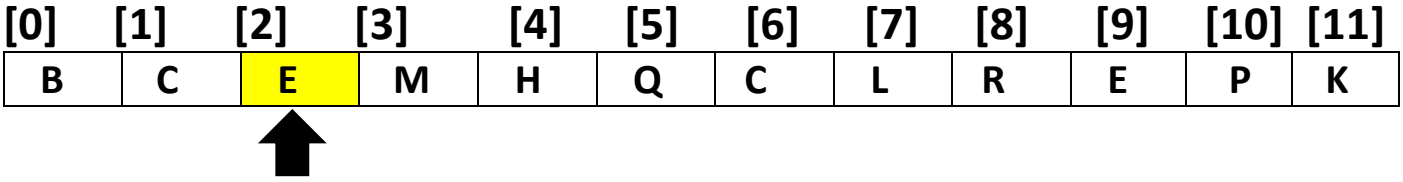


Step 47 :

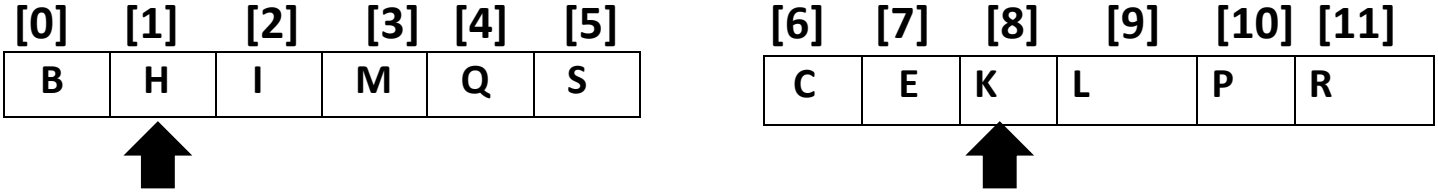


Output :

Comparison : 27
Displacement : 21

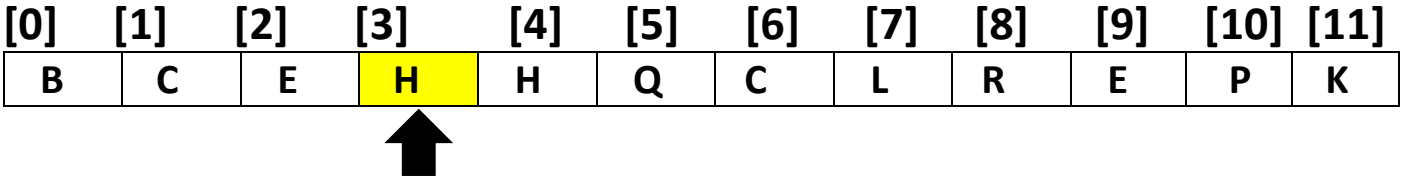


Step 48 :

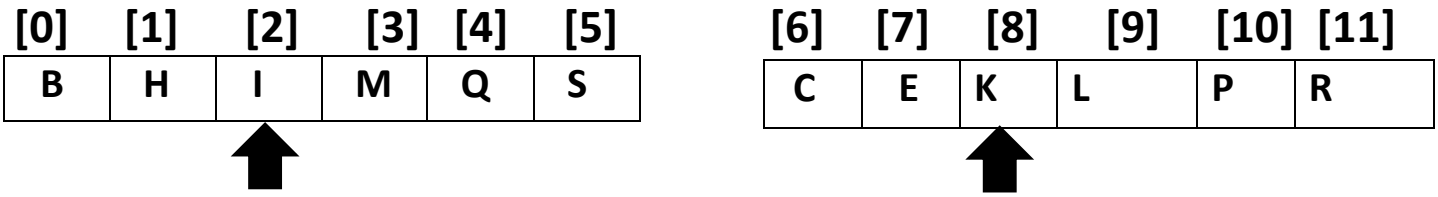


Output :

Comparison : 28
Displacement : 22

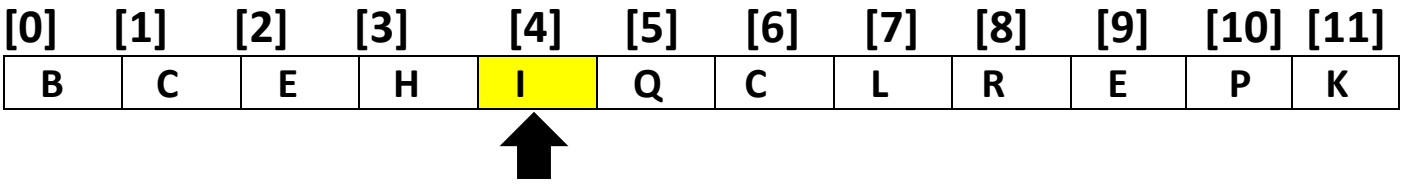


Step 49 :

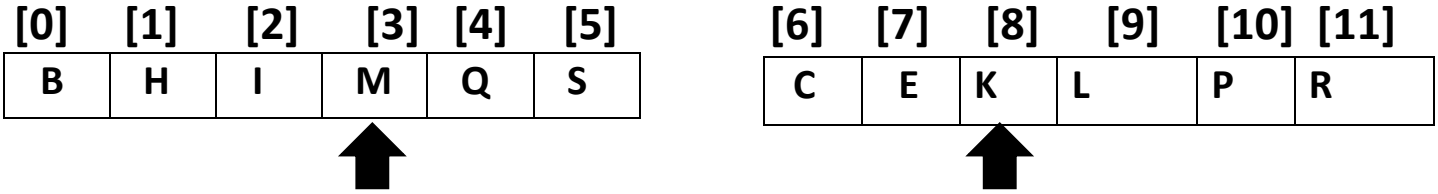


Output :

Comparison : 29
Displacement : 23

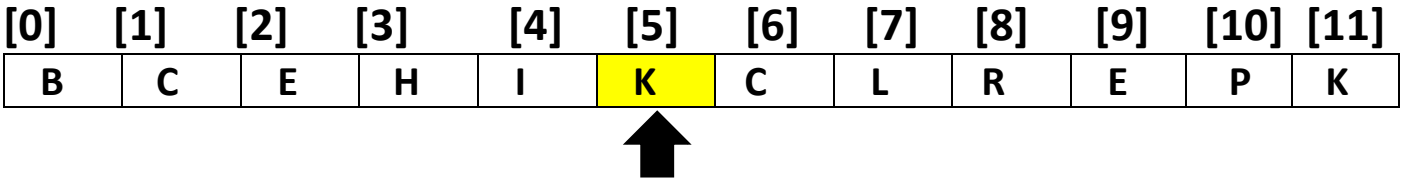


Step 50 :

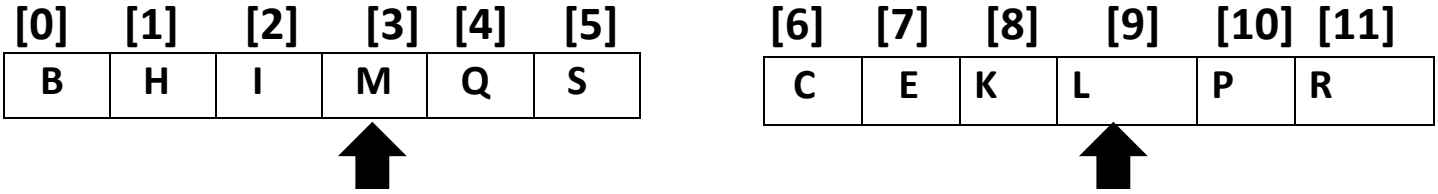


Output :

Comparison : 30
Displacement : 24

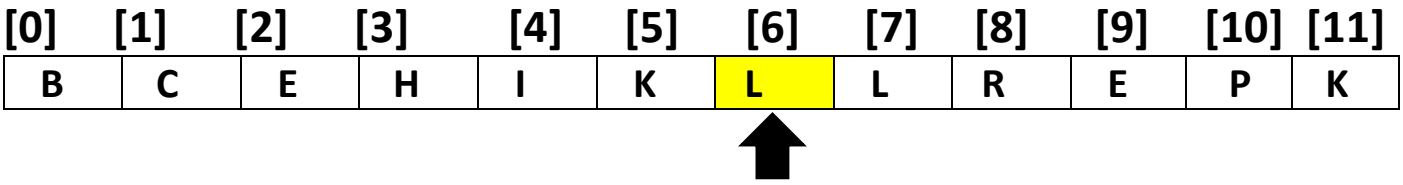


Step 51 :

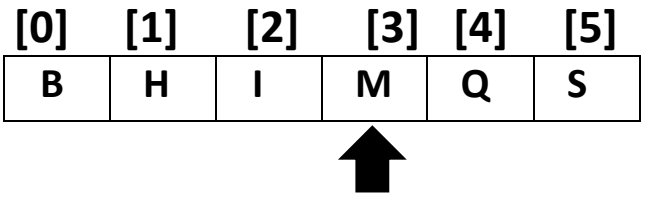


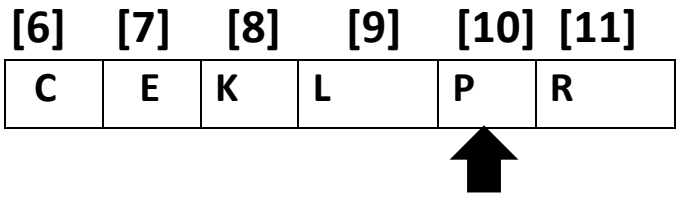
Output :

Comparison : 31
Displacement : 25



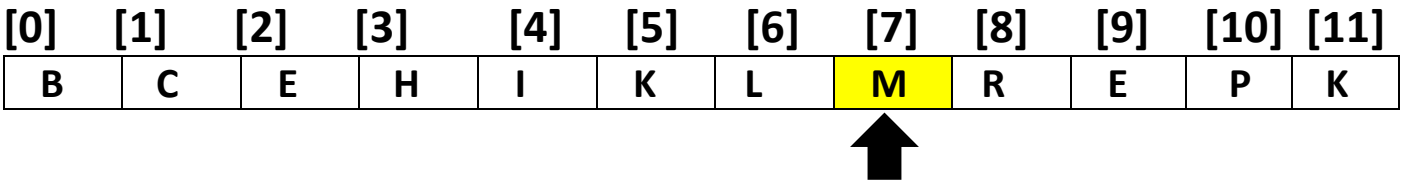
Step 52 :



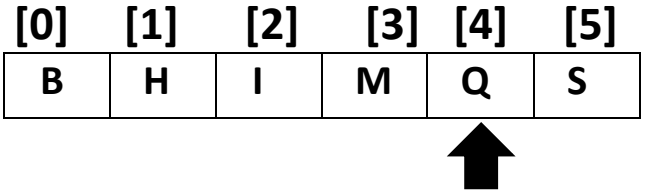


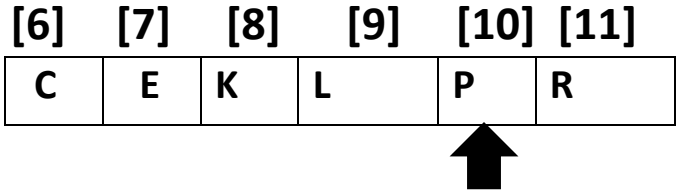
Output :

Comparison : 32
Displacement : 26



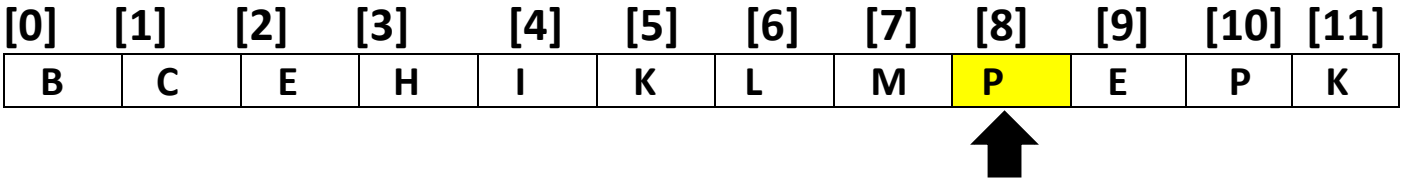
Step 53 :



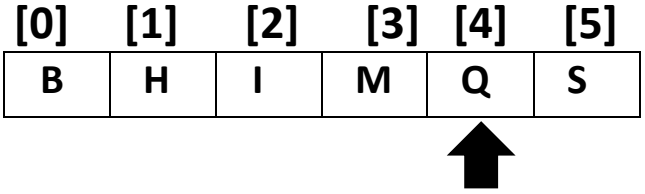


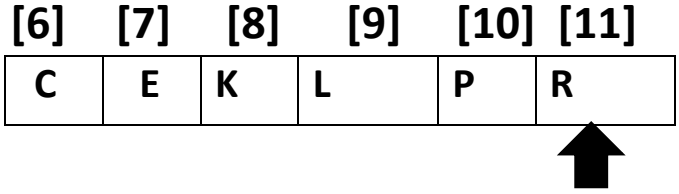
Output :

Comparison : 33
Displacement : 27



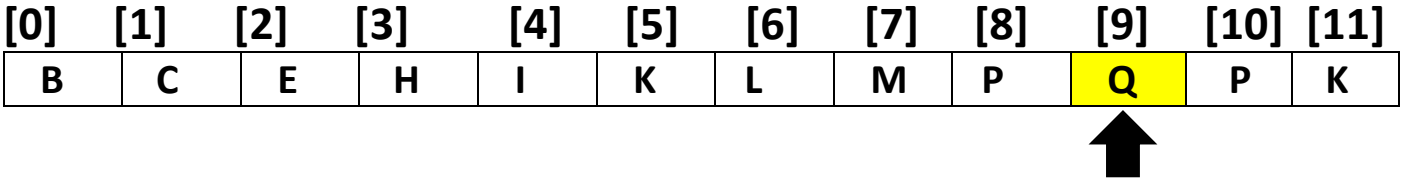
Step 54 :



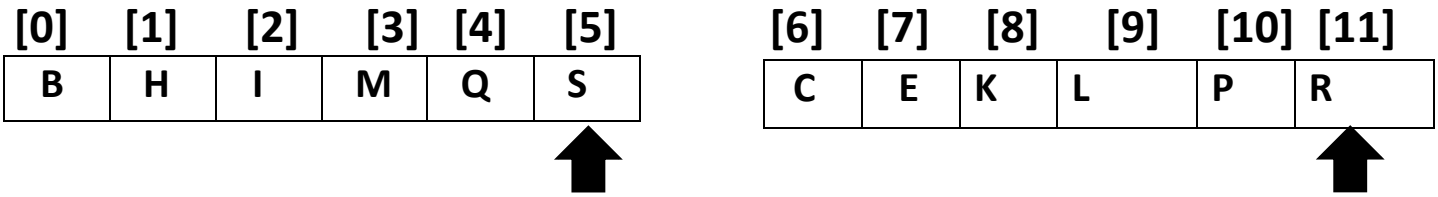


Output :

Comparison : 34
Displacement : 28

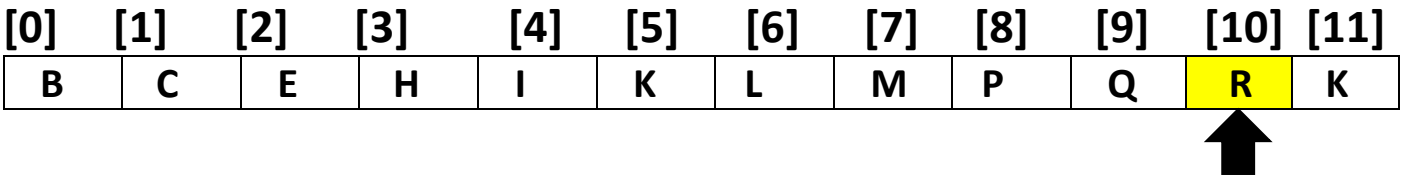


Step 55 :

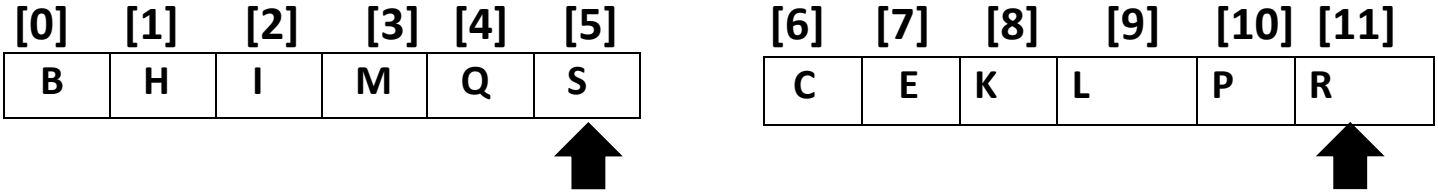


Output :

Comparison : 35
Displacement : 29

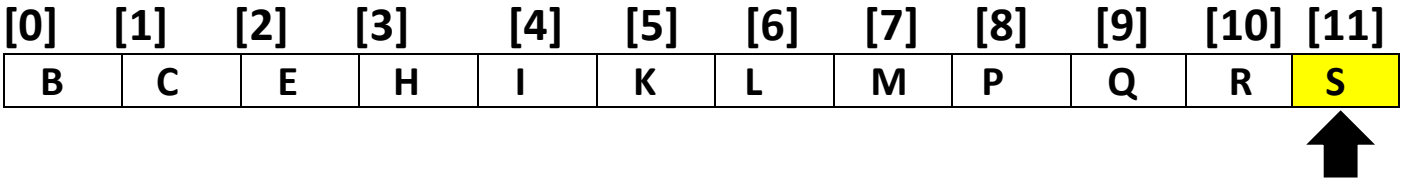


Step 56 :



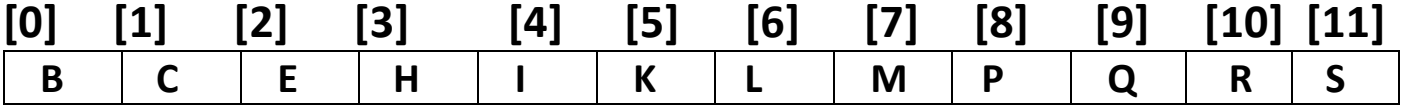
Output :

Comparison : 35
Displacement : 30



Step 57 : **The last counters and sorted array is below :**

Comparison : 35
Displacement : 30



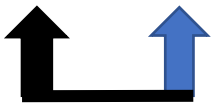
Sorting 3.a Array A (Integer Array From Small To Large - 10 Elements) – Heap Sort Algorithm

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

I will use the Heap Sort Algorithm in book :

Step 1 : Build a Heap
Comparison : 1
Displacement : 2

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
24	12	34	47	51	65	74	83	98	109



Step 2 : Build a Heap
Comparison : 2
Displacement : 4

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
34	12	24	47	51	65	74	83	98	109

Step 3 : Build a Heap
Comparison : 4
Displacement : 8

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
34	47	24	12	51	65	74	83	98	109

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
47	34	24	12	51	65	74	83	98	109

Step 4 : **Build a Heap**
Comparison : 6
Displacement : 12

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
47	51	24	12	34	65	74	83	98	109

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
51	47	24	12	34	65	74	83	98	109

Step 5 : **Build a Heap**
Comparison : 8
Displacement : 16

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
47	51	65	12	34	24	74	83	98	109

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
65	47	51	12	34	24	74	83	98	109

Step 6 : **Build a Heap**
Comparison : 10
Displacement : 20

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
65	47	74	12	34	24	51	83	98	109

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
74	47	65	12	34	24	51	83	98	109

Step 7 : **Build a Heap**

Comparison : 13

Displacement : 26

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
74	47	65	83	34	24	51	12	98	109

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
74	83	65	47	34	24	51	12	98	109

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
83	74	65	47	34	24	51	12	98	109

Step 8 : **Build a Heap**

Comparison : 16

Displacement : 32

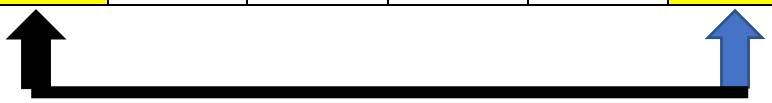
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
83	74	65	98	34	24	51	12	47	109

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
83	98	65	74	34	24	51	12	47	109

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
98	83	65	74	34	24	51	12	47	109

Step 9 : **Build a Heap**
Comparison : 19
Displacement : 38

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
98	83	65	74	109	24	51	12	47	34

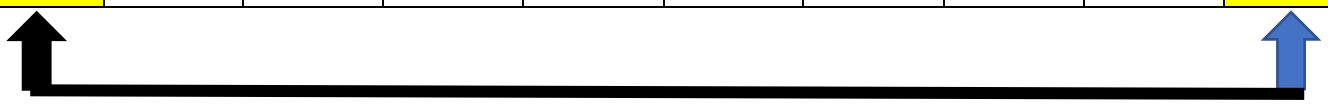


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
98	109	65	74	83	24	51	12	47	34

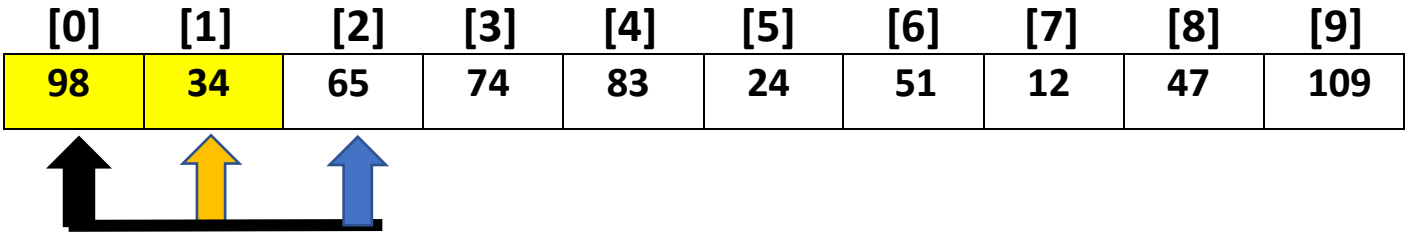
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
109	98	65	74	83	24	51	12	47	34

Step 10 : **Now I am in shrink method (n = table.length = 10) Swap (0,n-1)**
Comparison : 19
Displacement : 40

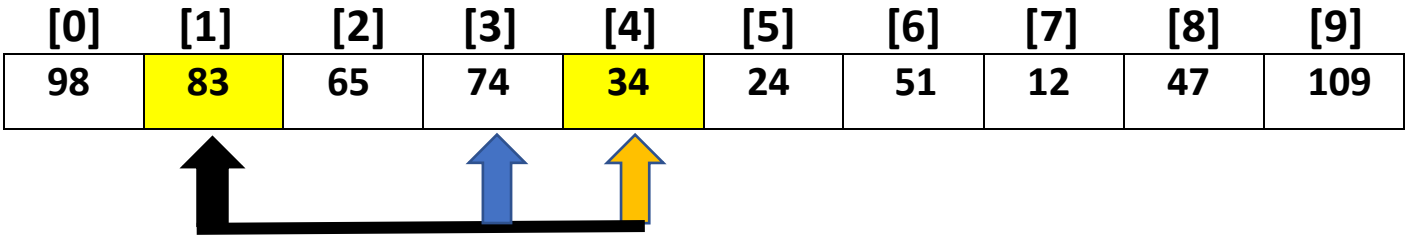
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
34	98	65	74	83	24	51	12	47	109



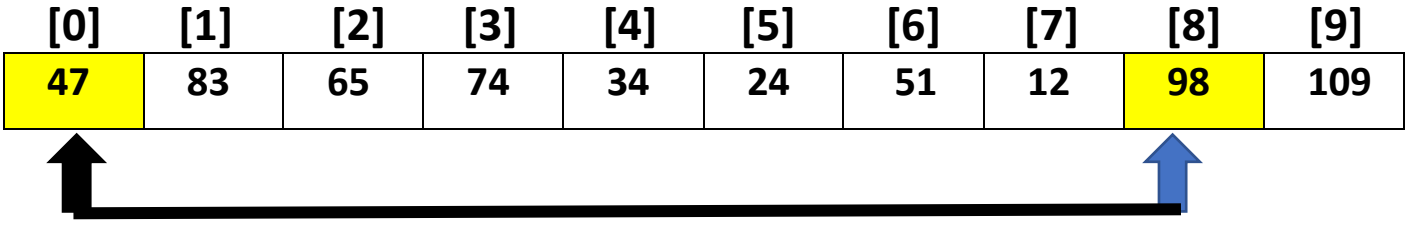
Step 11 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. In this step the left child (Index 1 is bigger than other child) is max Child.
Comparison : 20
Displacement : 42



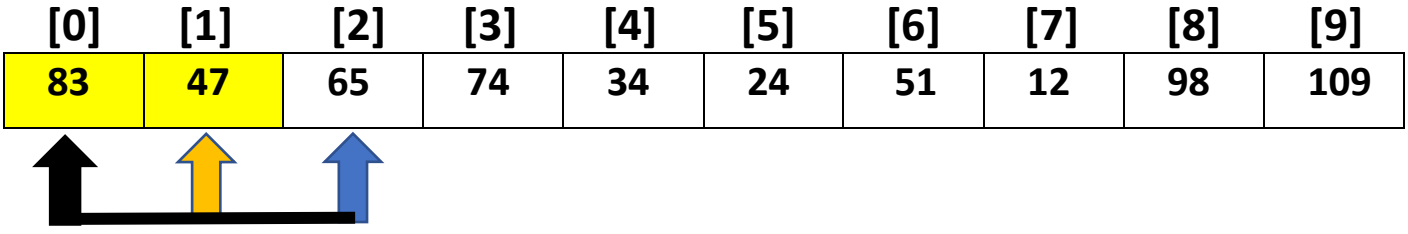
Step 12 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. In this step, the max child is right child whose index is 4.
Comparison : 21
Displacement : 44



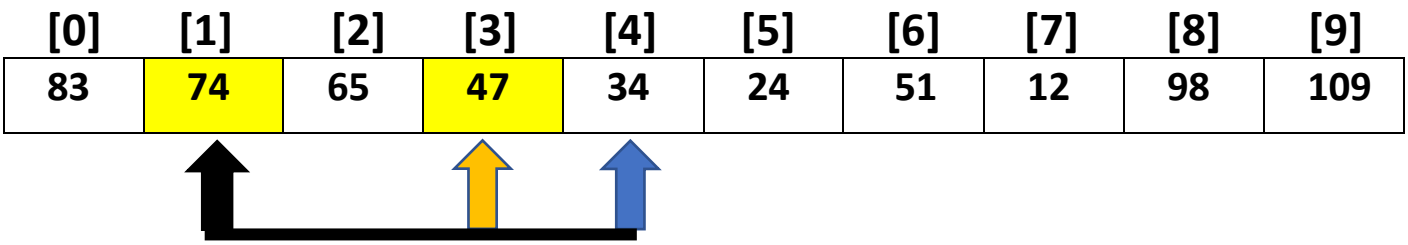
Step 12 : n = 8 , Swap (0,n-1)
Comparison : 21
Displacement : 46



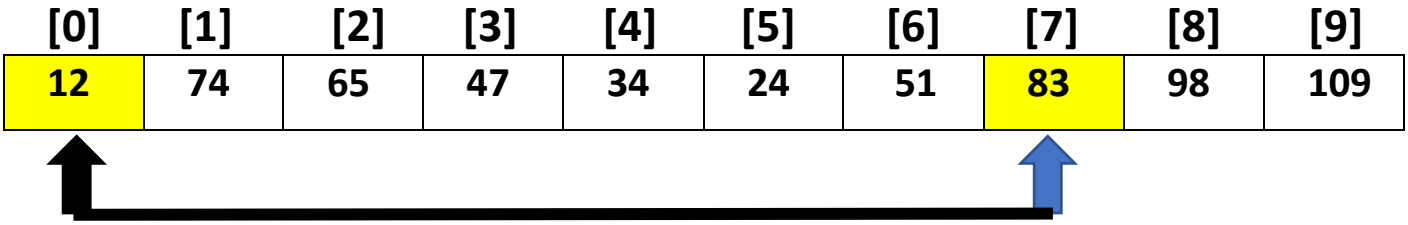
Step 13 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. In this step, the max child is left child whose index is 1
Comparison : 22
Displacement : 48



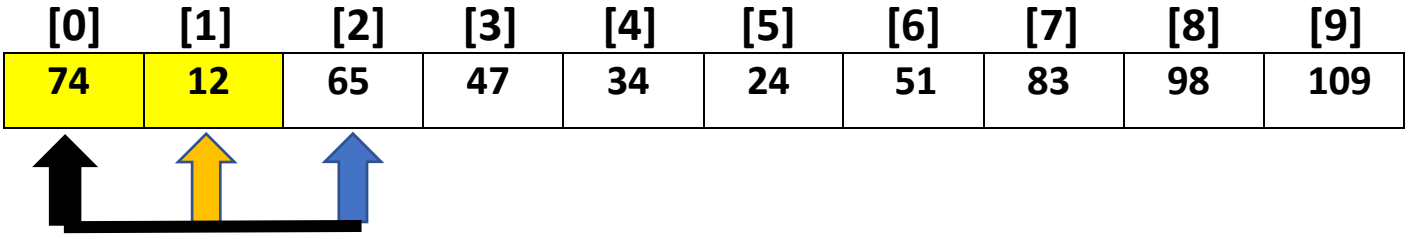
Step 14 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. In this step, the max child is left child whose index is 3.
Comparison : 23
Displacement : 50



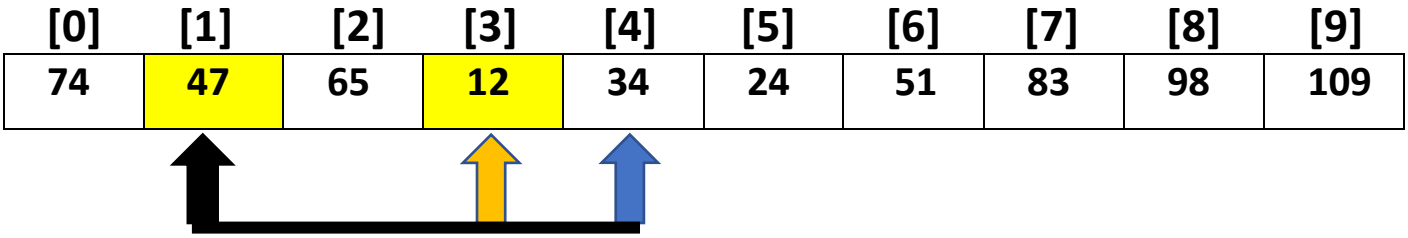
Step 15 : n = 7 , Swap (0,n-1)
Comparison : 23
Displacement : 52



Step 16 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. In this step, the max child is left child whose index is 1.
Comparison : 23
Displacement : 54



Step 17 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. Now the parent sets to the 1. index, so this parent compares with its children. The max child (left child) is bigger than parent so swap them.
Comparison : 24
Displacement : 56

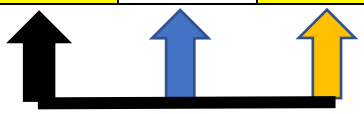


Step 18 : n = 6 , Swap (0,n-1)
Comparison : 24
Displacement : 58

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
51	47	65	12	34	24	74	83	98	109

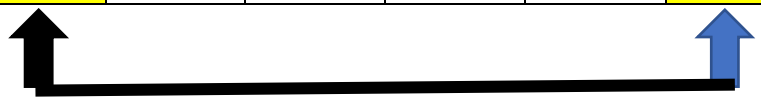
Step 19 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is right child. I don't check again (Parent 2, children 5,6) because the n is 5 in this method. So it is not our bounds
Comparison : 24
Displacement : 60

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
65	47	51	12	34	24	74	83	98	109



Step 20 : n = 5 , Swap (0,n-1)
Comparison : 24
Displacement : 62

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
24	47	51	12	34	65	74	83	98	109



Step 21 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is right child. And now the parent is 2.index but I can't compare because its children are 5,6 index but they are out of bounds (N = 5)
Comparison : 24
Displacement : 64

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
51	47	24	12	34	65	74	83	98	109

Step 22 : n = 4 , Swap (0,n-1)
Comparison : 24
Displacement : 66

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
34	47	24	12	51	65	74	83	98	109

Step 23 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child. (Index 1)
Comparison : 25
Displacement : 68

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
47	34	24	12	51	65	74	83	98	109

Step 24 : n = 3 , Swap (0,n-1)
Comparison : 25
Displacement : 70

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	34	24	47	51	65	74	83	98	109

Step 25 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child. (Index 1)
Comparison : 26
Displacement : 72

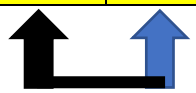
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
34	12	24	47	51	65	74	83	98	109

Step 26 : **n = 2 , Swap (0,n-1)**
Comparison : 26
Displacement : 74

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
24	12	34	47	51	65	74	83	98	109

Step 27 : **n = 1, Swap (0,n-1). This is the last part of sorting. So the counters and sorted array is below :**
Comparison : 26
Displacement : 76

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109



Sorting 3.b Array B (Integer Array Large To Small - 10 Elements) – Heap Sort Algorithm

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
982	842	731	654	549	439	384	264	152	46

Step 1 : Build a Heap
Comparison : 1
Displacement : 0

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
982	842	731	654	549	439	384	264	152	46

Step 2 : Build a Heap
Comparison : 2
Displacement : 0

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
982	842	731	654	549	439	384	264	152	46

Step 3 : Build a Heap
Comparison : 3
Displacement : 0

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
982	842	731	654	549	439	384	264	152	46

Step 4 : Build a Heap
Comparison : 4
Displacement : 0

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
982	842	731	654	549	439	384	264	152	46

Step 5 : Build a Heap
Comparison : 5
Displacement : 0

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
982	842	731	654	549	439	384	264	152	46

Step 6 : Build a Heap
Comparison : 6
Displacement : 0

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
982	842	731	654	549	439	384	264	152	46

Step 7 : Build a Heap
Comparison : 7
Displacement : 0

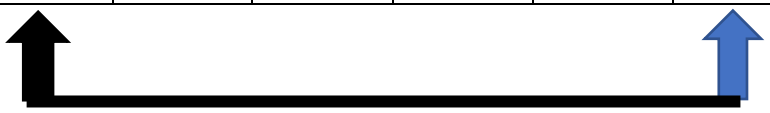
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
982	842	731	654	549	439	384	264	152	46

Step 8 : Build a Heap
Comparison : 8
Displacement : 0

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
982	842	731	654	549	439	384	264	152	46

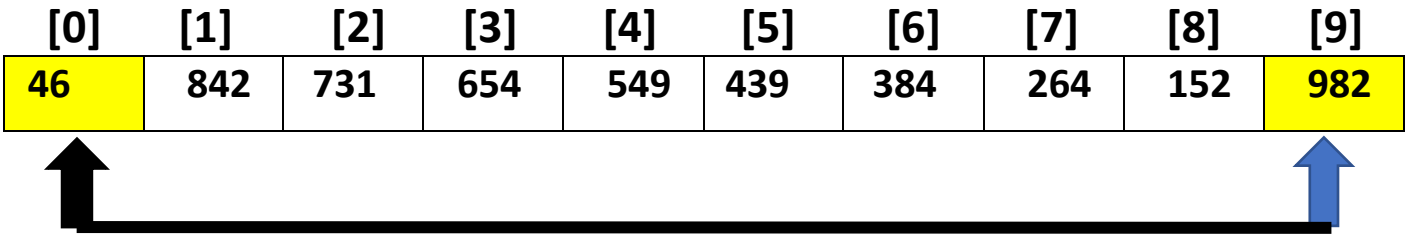
Step 9 : Build a Heap
Comparison : 9
Displacement : 0

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
982	842	731	654	549	439	384	264	152	46

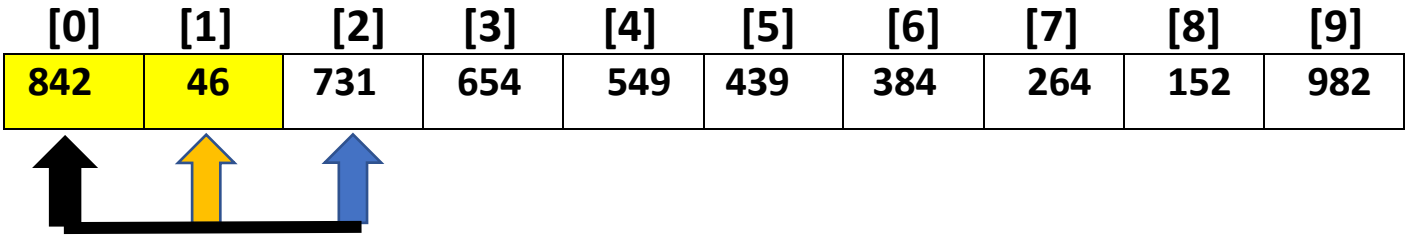


Step 10 : Now I am in shrink method (n = array.length = 10) Swap (0,n-1) => (0,9)

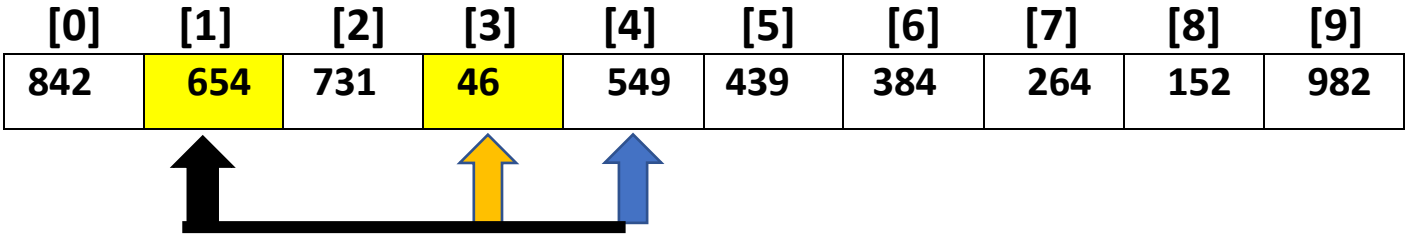
Comparison : 9
Displacement : 2



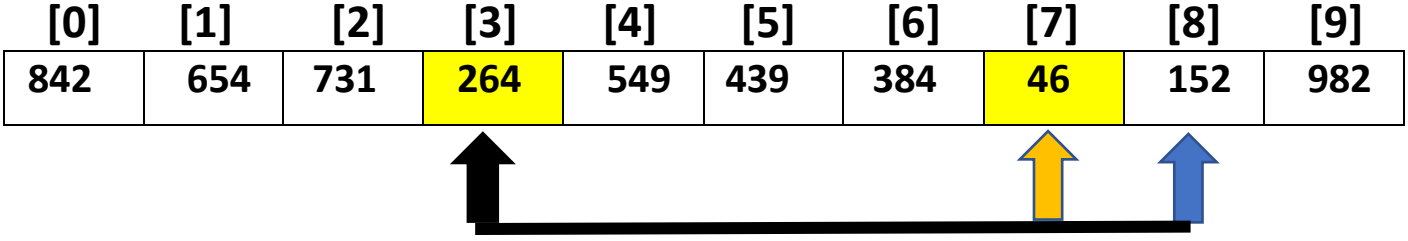
Step 11 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child. (Index 1)
Comparison : 10
Displacement : 4



Step 12 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. Now the left child becomes parent (Index3). So compare with children. The max child is bigger than parent. So swap them.
Comparison : 11
Displacement : 6



Step 13 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child(Index 7). Swap them. Now the parent is 7.index but I can't compare with its children because of out of bounds.
Comparison : 12
Displacement : 8

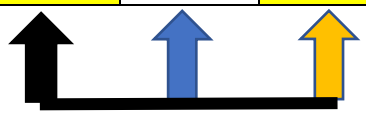


Step 14 : n = 8 , Swap (0,n-1)
Comparison : 12
Displacement : 10

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
152	654	731	264	549	439	384	46	842	982

Step 15 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is right child (Index 2). Swap them and now 1.index becomes parent and then check again with its children.
Comparison : 13
Displacement : 12

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
731	654	152	264	549	439	384	46	842	982

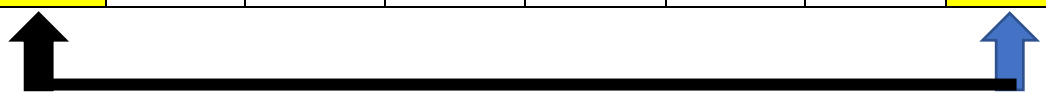


Step 16 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 5). Now the 5.index becomes parent but I can't compare with its children because of out of bounds
Comparison : 14
Displacement : 14

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
731	654	439	264	549	152	384	46	842	982

Step 17 : n = 7 , Swap (0,n-1)
Comparison : 14
Displacement : 16

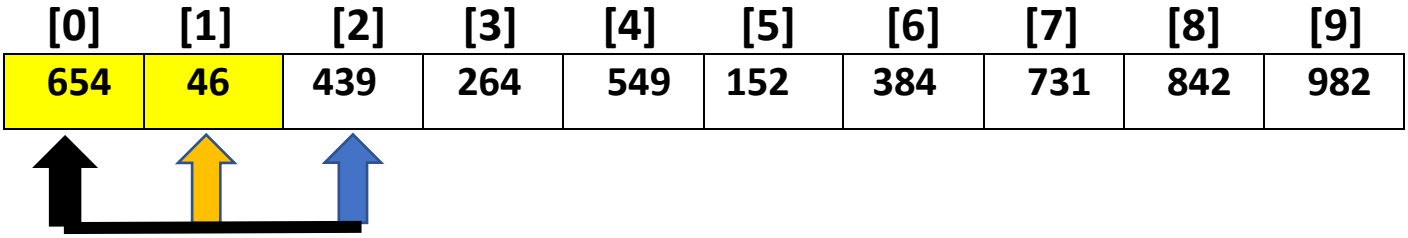
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	654	439	264	549	152	384	731	842	982



Step 18 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 1). Now index 1 becomes parent and I compare with its children

Comparison : 15

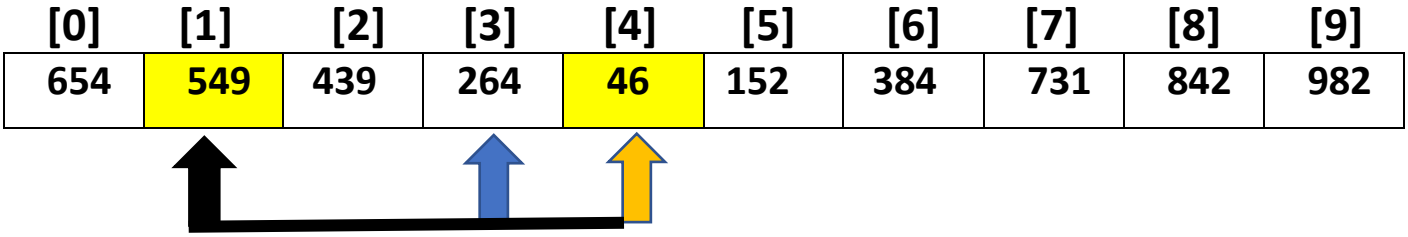
Displacement : 18



Step 19 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is right child (Index 4). Now the parent is 4.index but its children are not in bounds. So we can't compare.

Comparison : 16

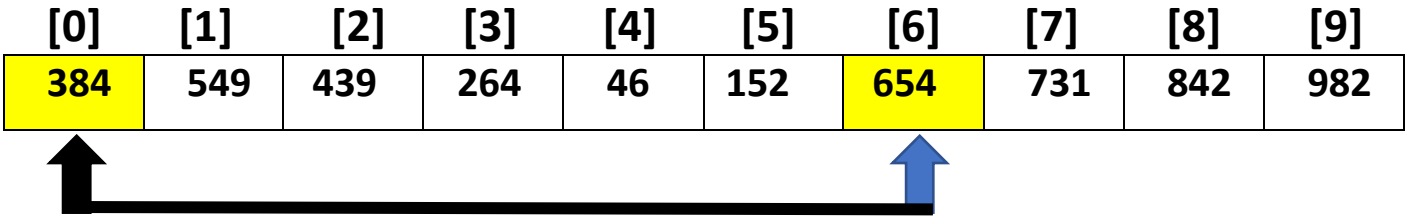
Displacement : 20



Step 20 : n = 6 , Swap (0,n-1)

Comparison : 16

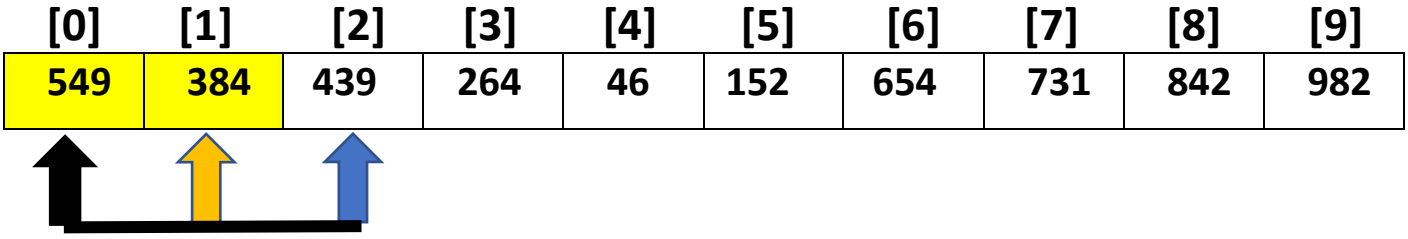
Displacement : 22



Step 21 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 1). Now the parent will be index 1. Its children are not bigger than parent. So I won't swap.

Comparison : 17

Displacement : 24

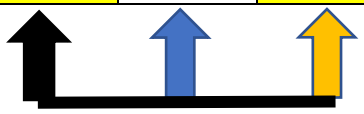


Step 22 : n = 5
Comparison : 17
Displacement : 26

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
152	384	439	264	46	549	654	731	842	982

Step 23 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is right child(Index 2). Now the parent is index 2 but I can't compare because its children's index 5,6 and they are not in bounds
Comparison : 18
Displacement : 28

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
439	384	152	264	46	549	654	731	842	982



Step 24 : n = 4
Comparison : 18
Displacement : 30

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	384	152	264	439	549	654	731	842	982

Step 25 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 1). Now the parent is index 1. I compare with its children.
Comparison : 19
Displacement : 32

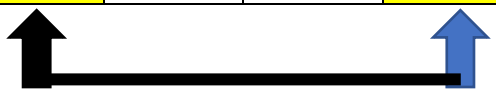
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
384	46	152	264	439	549	654	731	842	982

Step 26 : Set parent and max child to leftchildren(Right child is equal to n). After that compare. If the parent is smaller than the max child, then swap again.
Comparison : 20
Displacement : 34

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
384	264	152	46	439	549	654	731	842	982

Step 27 : n = 3
Comparison : 20
Displacement : 36

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	264	152	384	439	549	654	731	842	982



Step 28 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 1). Now the parent is index 1 but I can't compare because its children are out of bounds
Comparison : 21
Displacement : 38

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
264	46	152	384	439	549	654	731	842	982

Step 29 : n = 2
Comparison : 21
Displacement : 40

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
152	46	264	384	439	549	654	731	842	982

Step 30 : **n = 1**
Comparison : 21
Displacement : 42

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	549	654	731	842	982

Step 31 : **n = 0. Left child is equal to n. So the sorting is done. The last counters and sorted array is below :**
Comparison : 21
Displacement : 42

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	549	654	731	842	982

Sorting 3.c Array C (Integer Array - 12 Elements) – Heap Sort Algorithm

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
5	2	13	9	1	7	6	8	1	15	4	11

Step 1 : **Build a Heap**
Comparison : 1
Displacement : 0

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
5	2	13	9	1	7	6	8	1	15	4	11

Step 2 : **Build a Heap**
Comparison : 2
Displacement : 2

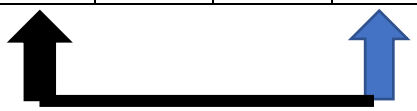
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
13	2	5	9	1	7	6	8	1	15	4	11

Step 3 : **Build a Heap**
Comparison : 3
Displacement : 4

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
13	9	5	2	1	7	6	8	1	15	4	11

Step 4 : **Build a Heap**
Comparison : 4
Displacement : 4

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
13	9	5	2	1	7	6	8	1	15	4	11

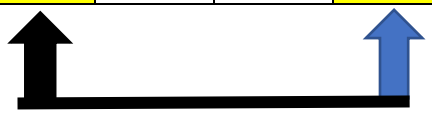


Step 5 : **Build a Heap**

Comparison : 5

Displacement : 6

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
13	9	7	2	1	5	6	8	1	15	4	11



Step 6 : **Build a Heap**

Comparison : 6

Displacement : 6

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
13	9	7	2	1	5	6	8	1	15	4	11

Step 7 : **Build a Heap**

Comparison : 7

Displacement : 8

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
13	9	7	8	1	5	6	2	1	15	4	11

Step 8 : **Build a Heap**

Comparison : 8

Displacement : 8

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
13	9	7	8	1	5	6	2	1	15	4	11

Step 9 : **Build a Heap**

Comparison : 9

Displacement : 10

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
13	9	7	8	15	5	6	2	1	1	4	11

Step 10 : **Build a Heap**
Comparison : 10
Displacement : 12

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
13	15	7	8	9	5	6	2	1	1	4	11

Step 11 : **Build a Heap**
Comparison : 11
Displacement : 14

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
15	13	7	8	9	5	6	2	1	1	4	11

Step 12 : **Build a Heap**
Comparison : 12
Displacement : 14

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
15	13	7	8	9	5	6	2	1	1	4	11

Step 13 : **Build a Heap**
Comparison : 13
Displacement : 16

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
15	13	7	8	9	11	6	2	1	1	4	5

Step 14 : **Build a Heap**
Comparison : 14
Displacement : 18

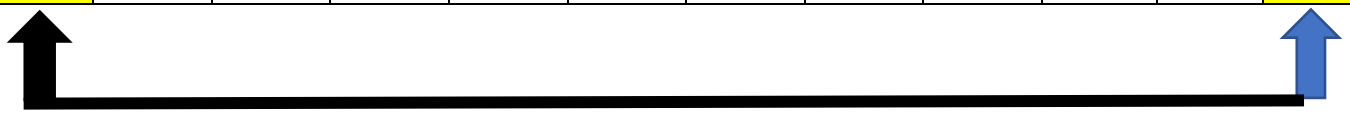
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
15	13	11	8	9	7	6	2	1	1	4	5

Step 15 : **Build a Heap**
Comparison : 15
Displacement : 18

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
15	13	11	8	9	7	6	2	1	1	4	5

Step 16 : **Now I am in shrink method. N = 12 which is array length. In while loop swap (0,n-1) = swap(0,11)**
Comparison : 15
Displacement : 20

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
5	13	11	8	9	7	6	2	1	1	4	15



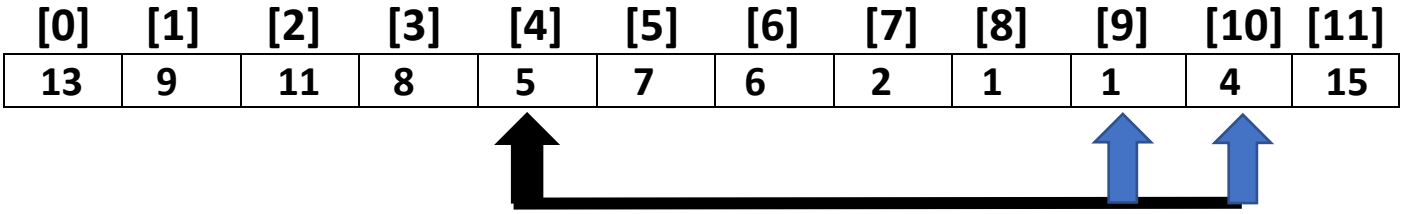
Step 17 : **Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 1). Now index 1 become parent and compare its children.**
Comparison : 16
Displacement : 22

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
13	5	11	8	9	7	6	2	1	1	4	15

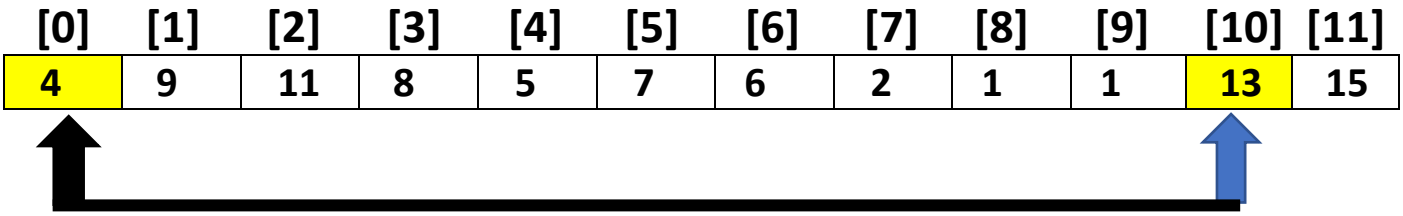
Step 18 : **Set parent and children. After that compare. If the parent is smaller than the max child, then swap again.The max child is right child (Index 4). Now index 4 become parent and compare its children**
Comparison : 17
Displacement : 24

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
13	9	11	8	5	7	6	2	1	1	4	15

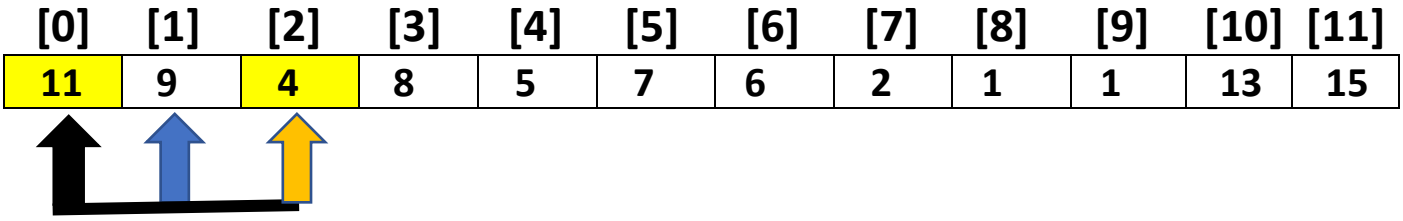
Step 19 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again.
Comparison : 18
Displacement : 24



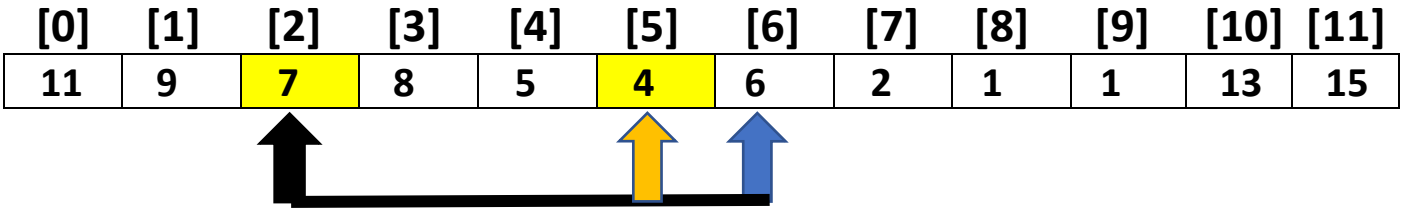
Step 20 : N = 11 swap (0,n--) = swap(0,10)
Comparison : 18
Displacement : 26



Step 21 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is right child (Index 2). Now index 2 becomes parent and compare with its children.
Comparison : 19
Displacement : 28



Step 22 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 5). Now index 5 becomes parent but I can't compare because its children are out of bounds (N = 10)
Comparison : 20
Displacement : 30



Step 23 : N = 10 swap (0,n--) = swap(0,9)
Comparison : 20
Displacement : 32

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	9	7	8	5	4	6	2	1	11	13	15

Step 24 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 1). Now index 1 become parent and I compare with its children.
Comparison : 21
Displacement : 34

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
9	1	7	8	5	4	6	2	1	11	13	15

Step 25 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 3). Now the index 3 becomes parent and I compare with its children.
Comparison : 22
Displacement : 36

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
9	8	7	1	5	4	6	2	1	11	13	15

Step 26 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 7). Now index 7 becomes parent but I can't compare with its children because they are out of bounds.
Comparison : 23
Displacement : 38

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
9	8	7	2	5	4	6	1	1	11	13	15

Step 27 : N = 9 swap (0,n--) = swap(0,8)
Comparison : 23
Displacement : 40

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	8	7	2	5	4	6	1	9	11	13	15

Step 28 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 1). Now index 1 becomes parent and I compare with its children.
Comparison : 24
Displacement : 42

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
8	1	7	2	5	4	6	1	9	11	13	15

Step 29 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is right child (Index 4). Now index 4 becomes parent but I can't compare with its children because they are out of bounds.

Comparison : 25
Displacement : 44

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
8	5	7	2	1	4	6	1	9	11	13	15

Step 30 : N = 8 swap (0,n--) = swap(0,7)
Comparison : 25
Displacement : 46

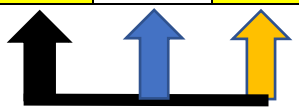
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	5	7	2	1	4	6	8	9	11	13	15

Step 31 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is right child (Index 2). Now index 2 becomes parent and I compare with its children.

Comparison : 26

Displacement : 48

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
7	5	1	2	1	4	6	8	9	11	13	15

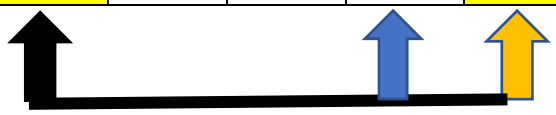


Step 32 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is right child (Index 6). Now index 6 becomes parent but I can't compare with its children beause they are out of bounds.

Comparison : 27

Displacement : 50

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
7	5	6	2	1	4	1	8	9	11	13	15



Step 33 : N = 7 swap (0,n--) = swap(0,6)

Comparison : 27

Displacement : 52

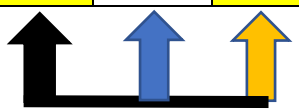
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	5	6	2	1	4	7	8	9	11	13	15

Step 34 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is right child (Index 2). Now index 2 becomes parent and I compare with its children.

Comparison : 28

Displacement : 54

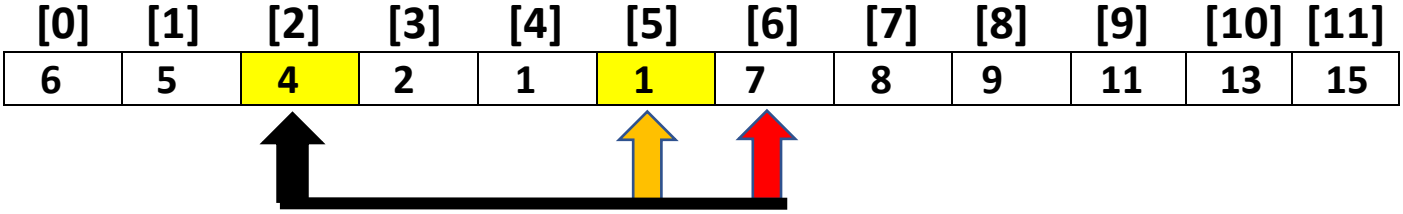
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
6	5	1	2	1	4	7	8	9	11	13	15



Step 35 : Set parent and max children to left child (Because right child is out of bounds because of n). If the parent is smaller than the max child, then swap again. The max child is right child (Index 5). Now index 5 becomes parent but I can't compare with its children beause they are out of bounds.

Comparison : 29

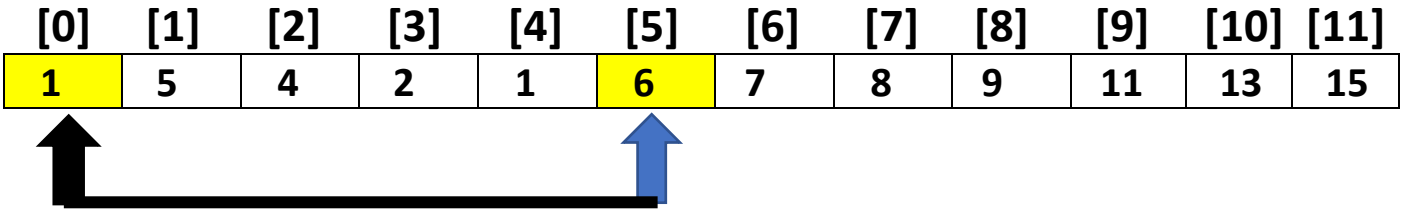
Displacement : 56



Step 36 : N = 6 swap (0,n--) = swap(0,5)

Comparison : 29

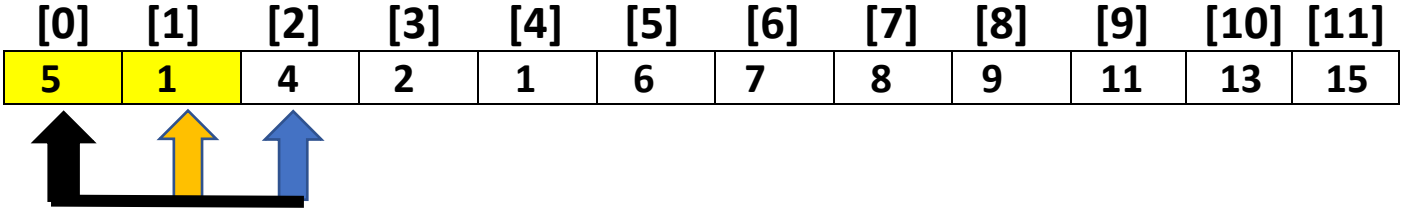
Displacement : 58



Step 37 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 1). Now index 1 becomes parent and I compare with its children.

Comparison : 30

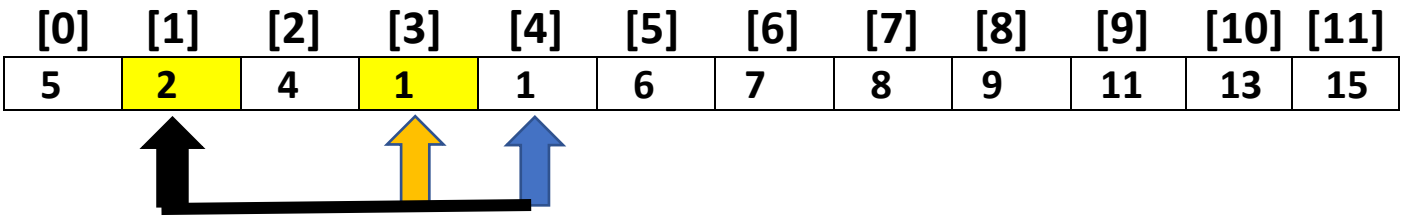
Displacement : 60



Step 38 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 3). Now index 3 becomes parent but I can't compare with its children beause they are out of bounds.

Comparison : 31

Displacement : 62

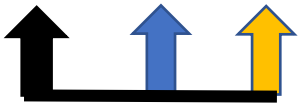


Step 39 : N = 5 swap (0,n--) = swap(0,4)
Comparison : 31
Displacement : 64

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	2	4	1	5	6	7	8	9	11	13	15

Step 40 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is right child (Index 2). Now index 2 becomes parent but I can't compare with its children beause they are out of bounds.
Comparison : 32
Displacement : 66

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
4	2	1	1	5	6	7	8	9	11	13	15



Step 41 : N = 4 swap (0,n--) = swap(0,3)
Comparison : 32
Displacement : 68

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	2	1	4	5	6	7	8	9	11	13	15

Step 42 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 1). Now index 1 becomes parent but I can't compare with its children beause they are out of bounds.
Comparison : 33
Displacement : 70

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
2	1	1	4	5	6	7	8	9	11	13	15

Step 43 : N = 3 swap (0,n--) = swap(0,2)
Comparison : 33
Displacement : 72

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	6	7	8	9	11	13	15

Step 44 : N = 2 swap (0,n--) = swap(0,1)
Comparison : 33
Displacement : 74

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	6	7	8	9	11	13	15

Step 45 : N = 1 swap (0,n--) = swap(0,0) So the sorting is done. The last counters and sorted array are below :
Comparison : 33
Displacement : 74

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	6	7	8	9	11	13	15

Sorting 3.d Array D (Char Array - 12 Elements) – Heap Sort Algorithm

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
S	B	I	M	H	Q	C	L	R	E	P	K

Step 1 : **Build a Heap**
Comparison : 1
Displacement : 0

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
S	B	I	M	H	Q	C	L	R	E	P	K

Step 2 : **Build a Heap**
Comparison : 2
Displacement : 0

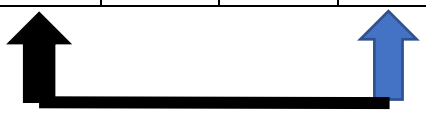
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
S	B	I	M	H	Q	C	L	R	E	P	K

Step 3 : **Build a Heap**
Comparison : 3
Displacement : 2

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
S	M	I	B	H	Q	C	L	R	E	P	K

Step 4 : **Build a Heap**
Comparison : 4
Displacement : 2

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
S	M	I	B	H	Q	C	L	R	E	P	K

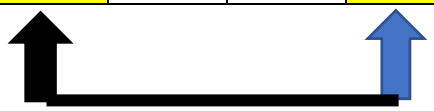


Step 5 : **Build a Heap**

Comparison : 5

Displacement : 4

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
S	M	Q	B	H	I	C	L	R	E	P	K

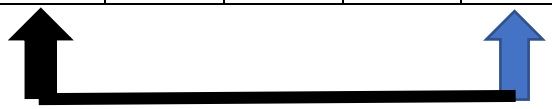


Step 6 : **Build a Heap**

Comparison : 6

Displacement : 4

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
S	M	Q	B	H	I	C	L	R	E	P	K

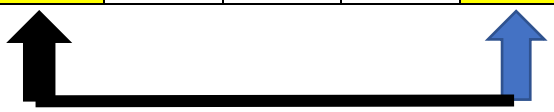


Step 7 : **Build a Heap**

Comparison : 8

Displacement : 6

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
S	M	Q	L	H	I	C	B	R	E	P	K

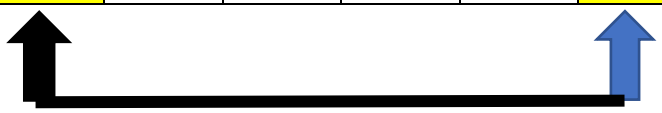


Step 8 : **Build a Heap**

Comparison : 9

Displacement : 8

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
S	M	Q	R	H	I	C	B	L	E	P	K



Step 9 : **Build a Heap**

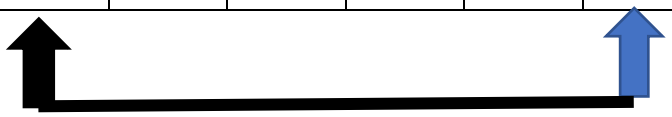
Comparison : 10

Displacement : 10

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
S	R	Q	M	H	I	C	B	L	E	P	K

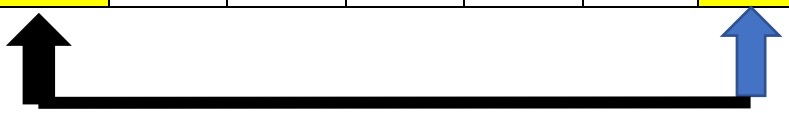
Step 10 : Build a Heap
Comparison : 11
Displacement : 10

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
S	R	Q	M	H	I	C	B	L	E	P	K



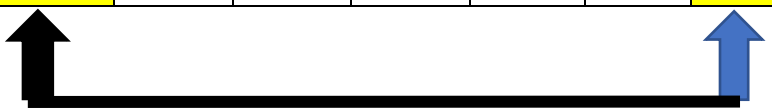
Step 11 : Build a Heap
Comparison : 12
Displacement : 12

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
S	R	Q	M	P	I	C	B	L	E	H	K



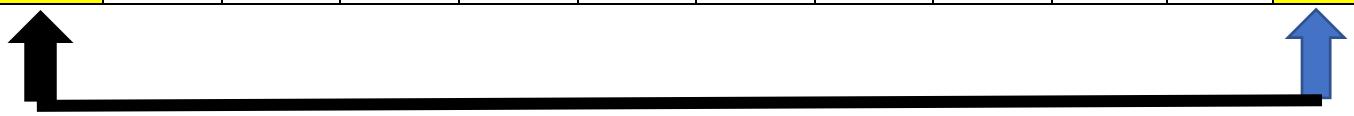
Step 12 : Build a Heap
Comparison : 13
Displacement : 14

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
S	R	Q	M	P	K	C	B	L	E	H	I



Step 13 : Now I am in shrink method. N = 12. So swap starts from 11 (0,n-1)
Comparison : 13
Displacement : 16

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
I	R	Q	M	P	K	C	B	L	E	H	S



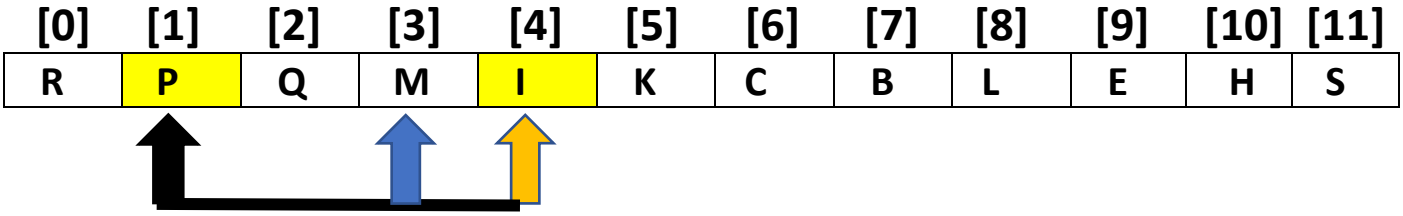
Step 14 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 1). Now index 1 becomes parent and I compare with its children.
Comparison : 14
Displacement : 18

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
R	I	Q	M	P	K	C	B	L	E	H	S

Step 15 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is right child (Index 4). Now index 4 becomes parent and I compare with its children.

Comparison : 15

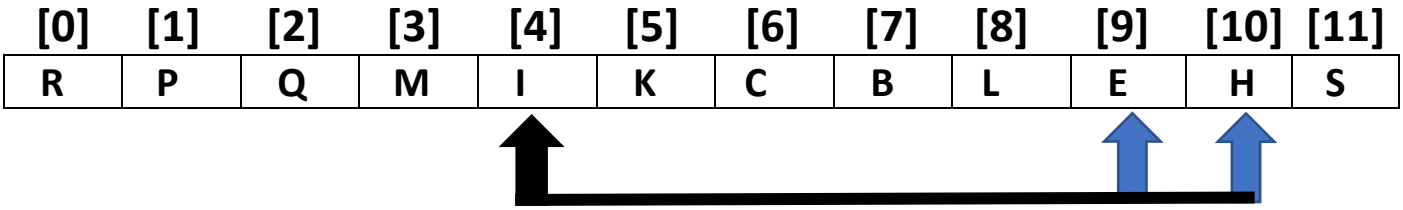
Displacement : 20



Step 16 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is right child (Index 6). Now index 6 becomes parent and I compare with its children.

Comparison : 16

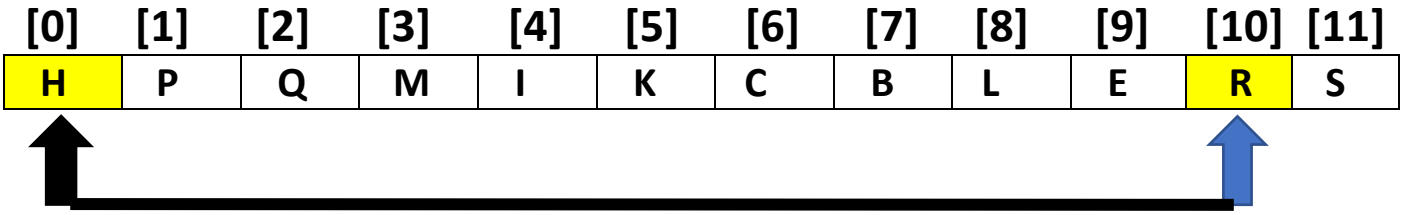
Displacement : 20



Step 17 : N = 11. So swap starts from 10 (0,n-1)

Comparison : 16

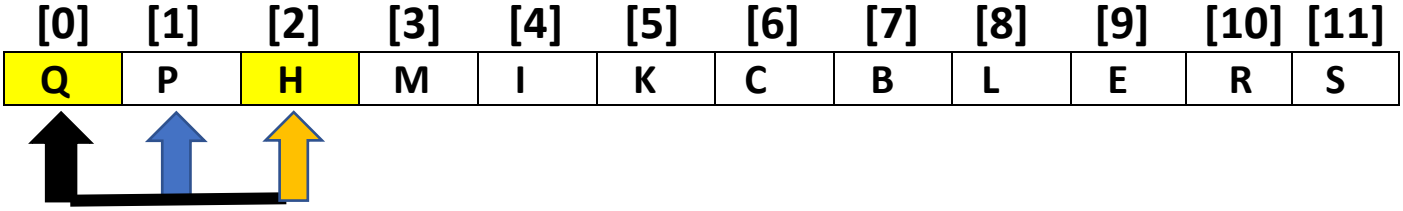
Displacement : 22



Step 18 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is right child (Index 2). Now index 2 becomes parent and I compare with its children.

Comparison : 17

Displacement : 24

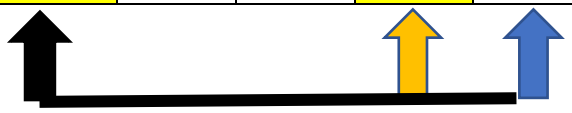


Step 19 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 5). Now index 5 becomes parent but I can't compare with its children because they are out of bounds.

Comparison : 18

Displacement : 26

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
Q	P	K	M	I	H	C	B	L	E	R	S



Step 20 : N = 10. So swap starts from 9 (0,n-1)

Comparison : 18

Displacement : 28

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
E	P	K	M	I	H	C	B	L	Q	R	S

Step 21 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 1). Now index 1 becomes parent and I compare with its children.

Comparison : 19

Displacement : 30

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
P	E	K	M	I	H	C	B	L	Q	R	S

Step 22 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 3). Now index 3 becomes parent and I compare with its children.

Comparison : 20

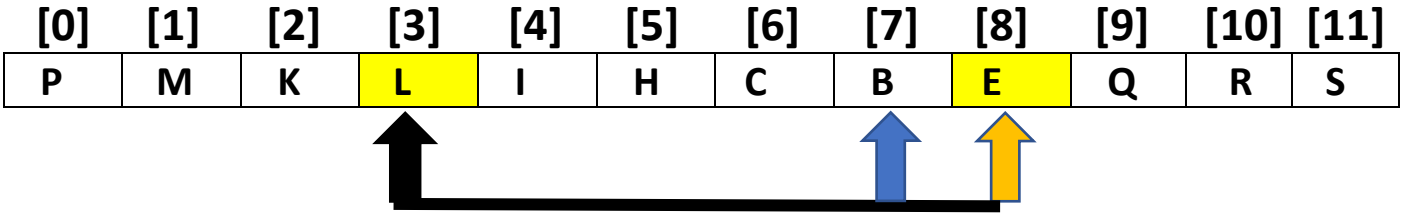
Displacement : 32

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
P	M	K	E	I	H	C	B	L	Q	R	S

Step 23 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is right child (Index 8). Now index 8 becomes parent but I can't compare with its children because they are out of bounds.

Comparison : 21

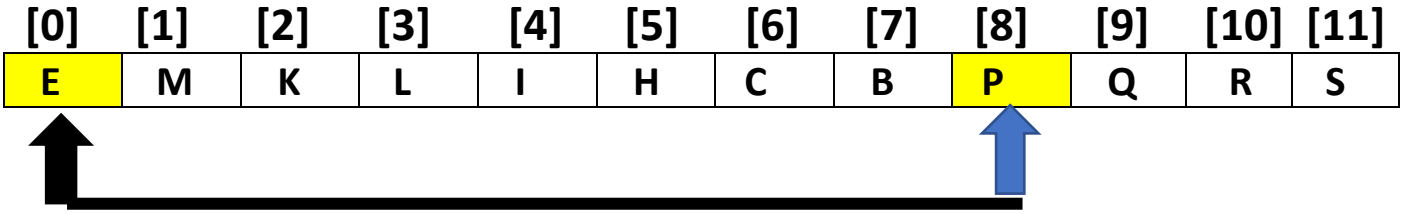
Displacement : 34



Step 24 : N = 9. So swap starts from 8 (0,n-1)

Comparison : 21

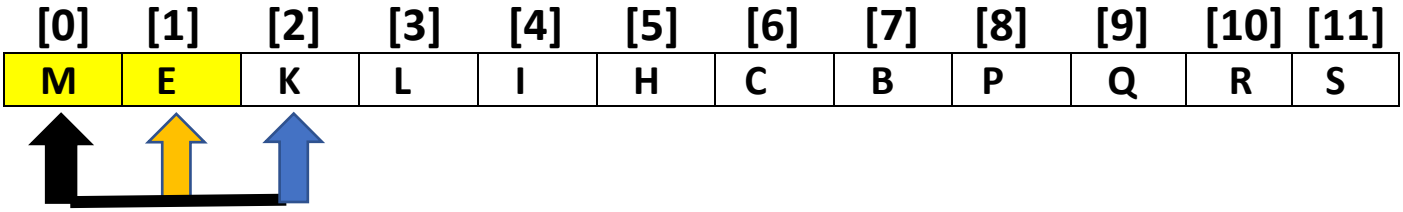
Displacement : 36



Step 25 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 1). Now index 1 becomes parent and I compare with its children.

Comparison : 22

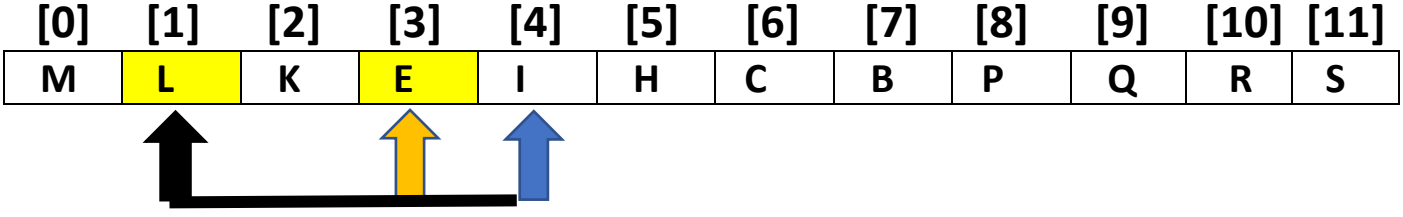
Displacement : 38



Step 26 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 3). Now index 3 becomes parent and I compare with left child because right child is out of bounds.

Comparison : 23

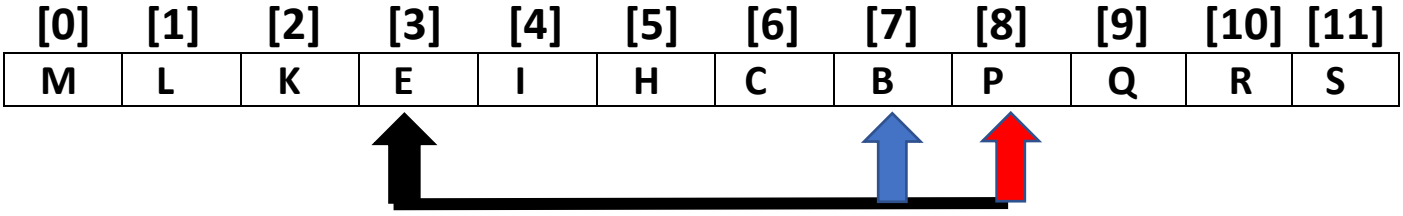
Displacement : 40



Step 27 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. Right child is out of bounds.

Comparison : 24

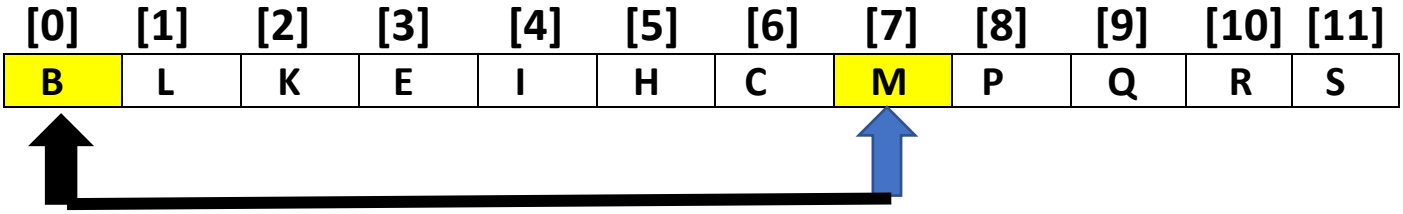
Displacement : 40



Step 28 : N = 8. So swap starts from 7 (0,n-1)

Comparison : 24

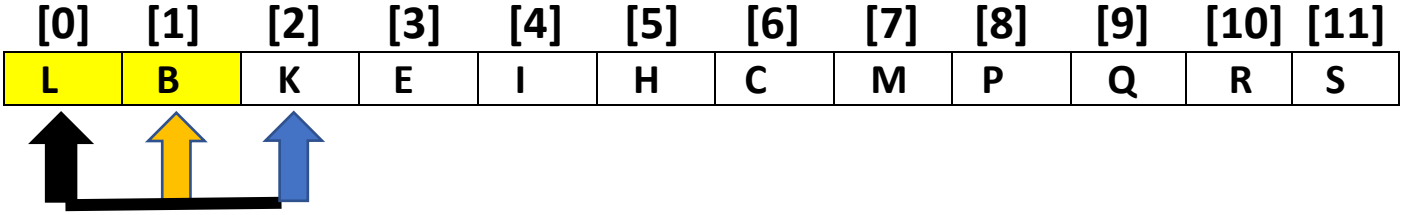
Displacement : 42



Step 29 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 1). Now index 1 becomes parent and I compare with its children.

Comparison : 25

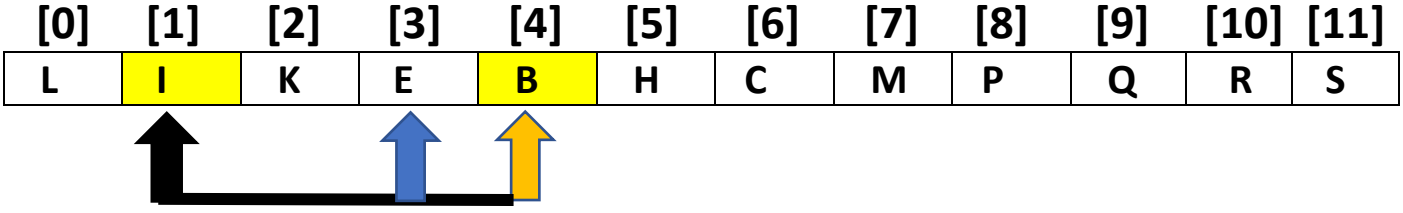
Displacement : 44



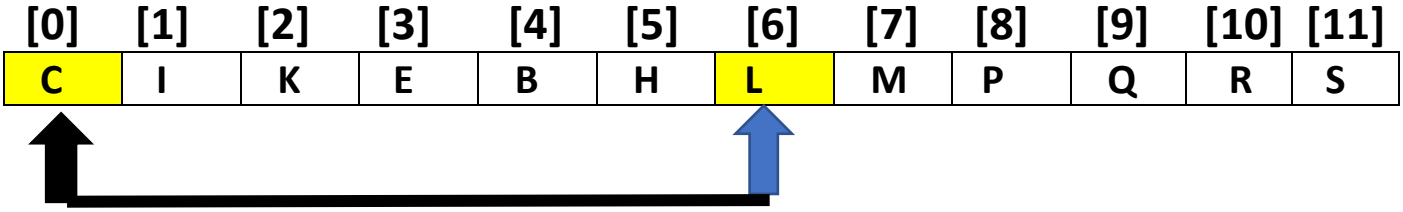
Step 30 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again.

Comparison : 26

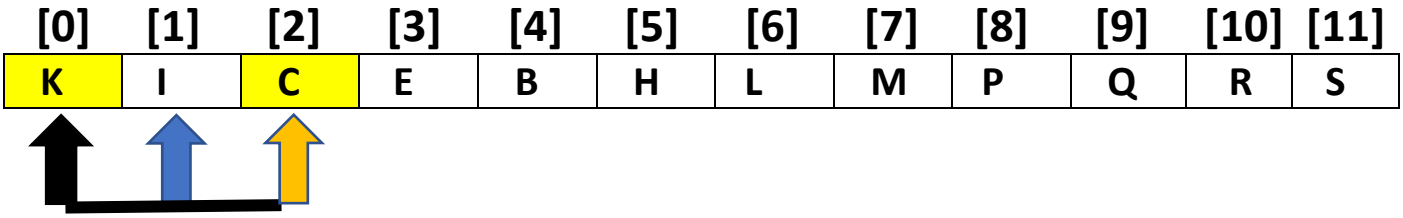
Displacement : 48



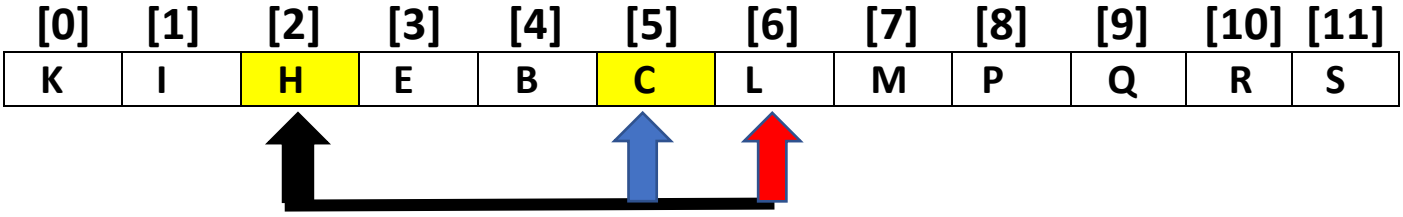
Step 31 : N = 7. So swap starts from 6 (0,n-1)
Comparison : 26
Displacement : 50



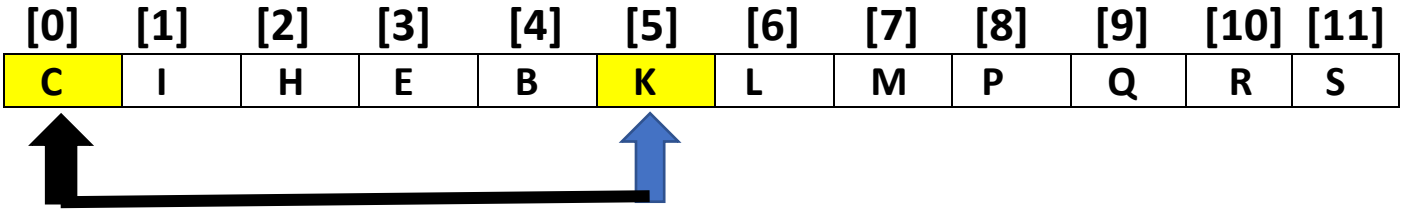
Step 32 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is right child (Index 2). Now index 2 becomes parent and I compare with its children.
Comparison : 27
Displacement : 52



Step 33 : Set parent and max child to left children(Right child is equal to n index). After that compare. If the parent is smaller than the max child, then swap again.
Comparison : 28
Displacement : 54



Step 34 : N = 6. So swap starts from 5 (0,n-1)
Comparison : 28
Displacement : 56



Step 35 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 1). Now index 1 becomes parent and I compare with its children.

Comparison : 29

Displacement : 58

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
I	C	H	E	B	K	L	M	P	Q	R	S

Step 36 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is right child (Index 2). Now index 2 becomes parent and I can't compare with its children because they are out of bounds.

Comparison : 30

Displacement : 60

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
I	E	H	C	B	K	L	M	P	Q	R	S

Step 37 : N = 5. So swap starts from 4 (0,n-1)

Comparison : 30

Displacement : 62

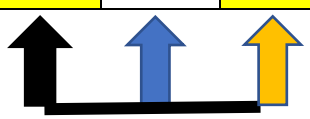
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	E	H	C	I	K	L	M	P	Q	R	S

Step 38 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is right child (Index 2). Now index 2 becomes parent and I compare with its children.

Comparison : 31

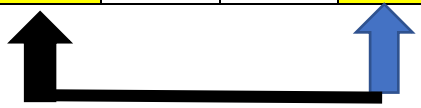
Displacement : 64

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
H	E	B	C	I	K	L	M	P	Q	R	S



Step 39 : N = 4. So swap starts from 3 (0,n-1)
Comparison : 31
Displacement : 66

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	E	B	H	I	K	L	M	P	Q	R	S



Step 40 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 1). Now index 1 becomes parent and I can't compare with its children because they are out of bounds.
Comparison : 32
Displacement : 68

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
E	C	B	H	I	K	L	M	P	Q	R	S

Step 41 : N = 3. So swap starts from 2 (0,n-1)
Comparison : 32
Displacement : 70

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	L	M	P	Q	R	S

Step 42 : Set parent and children. After that compare. If the parent is smaller than the max child, then swap again. The max child is left child (Index 1). Now index 1 becomes parent and I can't compare with its children because they are out of bounds.
Comparison : 33
Displacement : 72

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	B	E	H	I	K	L	M	P	Q	R	S

Step 1 : N = 2. So swap starts from 1 (0,n-1)
Comparison : 33
Displacement : 74

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	L	M	P	Q	R	S

Step 1 : N = 1. So the sorting is done. The last counters and sorted array are below :
Comparison : 33
Displacement : 74

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	L	M	P	Q	R	S

Sorting 4.a Array A (Integer Array From Small To Large-10 Elements) – Quick Sort Algorithm

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

Step 1: In this sorting, I will use “up” and “down” for partition.Pivot is now table[first] = 12

Method	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	Comp.	Displace
quickSort	0(First)	9(Last)	12	24	34	47	51	65	74	83	98	109	0	0
partition	1	9(Last)	12	24	34	47	51	65	74	83	98	109	1	0
partition	1	8	12	24	34	47	51	65	74	83	98	109	2	0
partition	1	7	12	24	34	47	51	65	74	83	98	109	3	0
partition	1	6	12	24	34	47	51	65	74	83	98	109	4	0
partition	1	5	12	24	34	47	51	65	74	83	98	109	5	0
partition	1	4	12	24	34	47	51	65	74	83	98	109	6	0
partition	1	3	12	24	34	47	51	65	74	83	98	109	7	0
partition	1	2	12	24	34	47	51	65	74	83	98	109	8	0
partition	1	1	12	24	34	47	51	65	74	83	98	109	9	0
partition	1	0	12	24	34	47	51	65	74	83	98	109	10	0
swap	0(F)	0	12	24	34	47	51	65	74	83	98	109	10	0

Step 2 : **This method returns the down which is 0. Now in partition method, my pivot is table[1] = 24. I compare according to this value**

Method	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	Comp.	Displace
quickSort	1(First)	9	12	24	34	47	51	65	74	83	98	109	10	0
partition	2	9(Last)	12	24	34	47	51	65	74	83	98	109	11	0
partition	2	8	12	24	34	47	51	65	74	83	98	109	12	0
partition	2	7	12	24	34	47	51	65	74	83	98	109	13	0
partition	2	6	12	24	34	47	51	65	74	83	98	109	14	0
partition	2	5	12	24	34	47	51	65	74	83	98	109	15	0
partition	2	4	12	24	34	47	51	65	74	83	98	109	16	0
partition	2	3	12	24	34	47	51	65	74	83	98	109	17	0
partition	2	2	12	24	34	47	51	65	74	83	98	109	18	0
partition	2	1	12	24	34	47	51	65	74	83	98	109	19	0
swap	1(First)	1	12	24	34	47	51	65	74	83	98	109	19	0

Step 3 : **This method returns the down which is 1. After swap in partition method, we are going to quickSort method. Now my pivot is table[2] = 34. I compare according to this value**

Method	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	Comp.	Displace
quickSort	2(First)	9(Last)	12	24	34	47	51	65	74	83	98	109	19	0
partition	3	9(Last)	12	24	34	47	51	65	74	83	98	109	20	0
partition	3	8	12	24	34	47	51	65	74	83	98	109	21	0
partition	3	7	12	24	34	47	51	65	74	83	98	109	22	0
partition	3	6	12	24	34	47	51	65	74	83	98	109	23	0
partition	3	5	12	24	34	47	51	65	74	83	98	109	24	0
partition	3	4	12	24	34	47	51	65	74	83	98	109	25	0
partition	3	3	12	24	34	47	51	65	74	83	98	109	26	0
partition	3	2	12	24	34	47	51	65	74	83	98	109	27	0
swap	2(First)	2	12	24	34	47	51	65	74	83	98	109	27	0

Step 4 : **This method returns the down which is 2. . After swap in partition method, we are going to quickSort method. Now in partition method, my pivot is table[3] = 47. I compare according to this value**

Method	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	Comp.	Displace
quickSort	3(First)	9(Last)	12	24	34	47	51	65	74	83	98	109	27	0
partition	4	9(Last)	12	24	34	47	51	65	74	83	98	109	28	0
partition	4	8	12	24	34	47	51	65	74	83	98	109	29	0
partition	4	7	12	24	34	47	51	65	74	83	98	109	30	0
partition	4	6	12	24	34	47	51	65	74	83	98	109	31	0
partition	4	5	12	24	34	47	51	65	74	83	98	109	32	0
partition	4	4	12	24	34	47	51	65	74	83	98	109	33	0
partition	4	3	12	24	34	47	51	65	74	83	98	109	34	0
swap	3(First)	3	12	24	34	47	51	65	74	83	98	109	3	0

Step 5 : **This method returns the down which is 3. After swap in partition method, we are going to quickSort method. Now in partition method, my pivot is table[4] = 51. I compare according to this value**

Method	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	Comp.	Displace
quickSort	4(First)	9(Last)	12	24	34	47	51	65	74	83	98	109	34	0
partition	5	8(Last)	12	24	34	47	51	65	74	83	98	109	35	0
partition	5	7	12	24	34	47	51	65	74	83	98	109	36	0
partition	5	6	12	24	34	47	51	65	74	83	98	109	37	0
partition	5	5	12	24	34	47	51	65	74	83	98	109	38	0
partition	5	4	12	24	34	47	51	65	74	83	98	109	39	0
swap	4(First)	4	12	24	34	47	51	65	74	83	98	109	39	0

Step 6 : This method returns the down which is 4. After swap in partition method, we are going to quickSort method. Now in partition method, my pivot is table[5] = 65. I compare according to this value.

Method	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	Comp.	Displace
quickSort	5(First)	9(Last)	12	24	34	47	51	65	74	83	98	109	39	0
partition	6	9(Last)	12	24	34	47	51	65	74	83	98	109	40	0
partition	6	8	12	24	34	47	51	65	74	83	98	109	41	0
partition	6	7	12	24	34	47	51	65	74	83	98	109	42	0
partition	6	6	12	24	34	47	51	65	74	83	98	109	43	0
partition	6	5	12	24	34	47	51	65	74	83	98	109	44	0
swap	5(First)	5	12	24	34	47	51	65	74	83	98	109	44	0

Step 7: This method returns the down which is 5. After swap in partition method, we are going to quickSort method. Now in partition method, my pivot is table[6] = 74. I compare according to this value

Method	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	Comp.	Displace
quickSort	6(First)	9(Last)	12	24	34	47	51	65	74	83	98	109	44	0
partition	7	9(Last)	12	24	34	47	51	65	74	83	98	109	45	0
partition	7	8	12	24	34	47	51	65	74	83	98	109	46	0
partition	7	7	12	24	34	47	51	65	74	83	98	109	47	0
partition	7	6	12	24	34	47	51	65	74	83	98	109	48	0
swap	6(First)	6	12	24	34	47	51	65	74	83	98	109	48	0

Step 8: This method returns the down which is 6. After swap in partition method, we are going to quickSort method. Now in partition method, my pivot is table[7] = 83. I compare according to this value.

Method	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	Comp.	Displace
quickSort	7(First)	9(Last)	12	24	34	47	51	65	74	83	98	109	48	0
partition	8	9(Last)	12	24	34	47	51	65	74	83	98	109	49	0
partition	8	8	12	24	34	47	51	65	74	83	98	109	50	0
partition	8	7	12	24	34	47	51	65	74	83	98	109	51	0
swap	7(First)	7	12	24	34	47	51	65	74	83	98	109	51	0

Step 9: This method returns the down which is 7. After swap in partition method, we are going to quickSort method. Now in partition method, my pivot is table[8] = 98. I compare according to this value

Method	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	Comp.	Displace
quickSort	8(First)	9(Last)	12	24	34	47	51	65	74	83	98	109	51	0
partition	9	9(Last)	12	24	34	47	51	65	74	83	98	109	52	0
partition	9	8	12	24	34	47	51	65	74	83	98	109	53	0
swap	8(First)	8	12	24	34	47	51	65	74	83	98	109	53	0

Step 10: This method returns the down which is 7. After swap in partition method, we are going to quickSort method. But when we send the partition method again, the first and last index is equal. So the sorting is done. The last counters and sorted array is below :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	24	34	47	51	65	74	83	98	109

Comparison : 53
Displacement : 0

Sorting 4.b Array B (Integer Array From Large To Small -10 Elements) – Quick Sort Algorithm

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
982	842	731	654	549	439	384	264	152	46

Step 1: In this sorting, I will use “up” and “down” for partition.
Pivot is now table[first] = table[0] = 982
First = 0, Last = 9

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	Comp.	Displ.
quick.	0(First)	9(Last)	982	842	731	654	549	439	384	264	152	46	0	0
part.	1	9(Last)	982	842	731	654	549	439	384	264	152	46	1	0
part.	2	9(Last)	982	842	731	654	549	439	384	264	152	46	2	0
part.	3	9(Last)	982	842	731	654	549	439	384	264	152	46	3	0
part.	4	9(Last)	982	842	731	654	549	439	384	264	152	46	4	0
part.	5	9(Last)	982	842	731	654	549	439	384	264	152	46	5	0
part.	6	9(Last)	982	842	731	654	549	439	384	264	152	46	6	0
part.	7	9(Last)	982	842	731	654	549	439	384	264	152	46	7	0
part.	8	9(Last)	982	842	731	654	549	439	384	264	152	46	8	0
part.	9	9(Last)	982	842	731	654	549	439	384	264	152	46	9	0
swap	0(First)	9	46	842	731	654	549	439	384	264	152	982	9	2

Step 2: The piv Index is returned down which is 9. After the swap, we are going to back the quickSort. Now our pivot is table[0] = 46.
First = 0, Last = 8

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	Comp.	Displ.
quick.	0(First)	8(Last)	46	842	731	654	549	439	384	264	152	982	9	2
part.	1	8	46	842	731	654	549	439	384	264	152	982	10	2
part.	1	7	46	842	731	654	549	439	384	264	152	982	11	2
part.	1	6	46	842	731	654	549	439	384	264	152	982	12	2
part.	1	5	46	842	731	654	549	439	384	264	152	982	13	2
part.	1	4	46	842	731	654	549	439	384	264	152	982	14	2
part.	1	3	46	842	731	654	549	439	384	264	152	982	15	2
part.	1	2	46	842	731	654	549	439	384	264	152	982	16	2
part.	1	1	46	842	731	654	549	439	384	264	152	982	17	2
part.	1	0	46	842	731	654	549	439	384	264	152	982	18	2
swap	0(First)	0	46	842	731	654	549	439	384	264	152	982	18	2

Step 3: The piv Index is returned down which is 0. So swap first and down index.
After the swap, we are going to back the quickSort. Now our pivot is table[1] = 842.
First = 1, Last = 8

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	Comp.	Displ.
quick.	1(First)	8(Last)	46	842	731	654	549	439	384	264	152	982	18	2
part.	2	8	46	842	731	654	549	439	384	264	152	982	19	2
part.	3	8	46	842	731	654	549	439	384	264	152	982	20	2
part.	4	8	46	842	731	654	549	439	384	264	152	982	21	2
part.	5	8	46	842	731	654	549	439	384	264	152	982	22	2
part.	6	8	46	842	731	654	549	439	384	264	152	982	23	2
part.	7	8	46	842	731	654	549	439	384	264	152	982	24	2
part.	8	8	46	842	731	654	549	439	384	264	152	982	25	2
swap	1(First)	8	46	152	731	654	549	439	384	264	842	982	25	4

Step 4: The piv Index is returned down which is 8. So swap first and down index.
After the swap, we are going to back the quickSort. Now our pivot is table[1] = 152
First = 1, Last = 7

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	Comp.	Displ.
quick.	1(First)	7(Last)	46	152	731	654	549	439	384	264	842	982	25	4
part.	2	7	46	152	731	654	549	439	384	264	842	982	26	4
part.	2	6	46	152	731	654	549	439	384	264	842	982	27	4
part.	2	5	46	152	731	654	549	439	384	264	842	982	28	4
part.	2	4	46	152	731	654	549	439	384	264	842	982	29	4
part.	2	3	46	152	731	654	549	439	384	264	842	982	30	4
part.	2	2	46	152	731	654	549	439	384	264	842	982	31	4
part.	2	1	46	152	731	654	549	439	384	264	842	982	32	4
swap	1(First)	1	46	152	731	654	549	439	384	264	842	982	32	4

Step 5: The piv Index is returned down which is 1. So swap first and down index.
After the swap, we are going to back the quickSort. Now our pivot is table[2] = 731
First = 2 Last = 7

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	Comp.	Displ.
quick.	2(First)	7(Last)	46	152	731	654	549	439	384	264	842	982	32	4
part.	3	7	46	152	731	654	549	439	384	264	842	982	33	4
part.	4	7	46	152	731	654	549	439	384	264	842	982	34	4
part.	5	7	46	152	731	654	549	439	384	264	842	982	35	4
part.	6	7	46	152	731	654	549	439	384	264	842	982	36	4
part.	7	7	46	152	731	654	549	439	384	264	842	982	37	4
swap	2(First)	7	46	152	264	654	549	439	384	731	842	982	37	6

Step 6: The piv Index is returned down which is 7. So swap first and down index.
After the swap, we are going to back the quickSort. Now our pivot is table[2] = 264
First = 2 Last = 6

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	Comp.	Displ.
quick.	2(First)	6(Last)	46	152	264	654	549	439	384	731	842	982	37	6
part.	3	6(Last)	46	152	264	654	549	439	384	731	842	982	38	6
part.	3	5	46	152	264	654	549	439	384	731	842	982	39	6
part.	3	4	46	152	264	654	549	439	384	731	842	982	40	6
part.	3	3	46	152	264	654	549	439	384	731	842	982	41	6
part.	3	2	46	152	264	654	549	439	384	731	842	982	42	6
swap	2(First)	2	46	152	264	654	549	439	384	731	842	982	42	6

Step 7: The piv Index is returned down which is 2. So swap first and down index.
After the swap, we are going to back the quickSort. Now our pivot is table[3] = 654
First = 3 Last = 6

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	Comp.	Displ.
quick.	3(First)	6(Last)	46	152	264	654	549	439	384	731	842	982	42	6
part.	4	6(Last)	46	152	264	654	549	439	384	731	842	982	43	6
part.	4	6(Last)	46	152	264	654	549	439	384	731	842	982	44	6
part.	5	6(Last)	46	152	264	654	549	439	384	731	842	982	45	6
part.	6	6(Last)	46	152	264	654	549	439	384	731	842	982	46	6
swap	3(First)	6	46	152	264	384	549	439	654	731	842	982	46	8

Step 8: The piv Index is returned down which is 6. So swap first and down index.
After the swap, we are going to back the quickSort. Now our pivot is table[3] = 384
First = 3 , Last = 5

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	Comp.	Displ.
part.	3(First)	5(Last)	46	152	264	384	549	439	654	731	842	982	46	8
quick.	4	5	46	152	264	384	549	439	654	731	842	982	47	8
part.	4	4	46	152	264	384	549	439	654	731	842	982	48	8
part.	4	3	46	152	264	384	549	439	654	731	842	982	49	8
swap	3(First)	3	46	152	264	384	549	439	654	731	842	982	49	8

Step 9: The piv Index is returned down which is 3. So swap first and down index.
After the swap, we are going to back the quickSort. Now our pivot is table[4] = 549
First = 4, Last = 5

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	Comp.	Displ.
quick.	4(First)	5(Last)	46	152	264	384	549	439	654	731	842	982	49	8
part.	5	5	46	152	264	384	549	439	654	731	842	982	50	8
swap	4(First)	5	46	152	264	384	439	549	654	731	842	982	50	10

Step 10: The piv Index is returned down which is 5. So swap first and down index.
After the swap, we are going to back the quickSort. Now

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	Comp.	Displ.
part.	5(First)	5(Last)	46	152	264	384	439	549	654	731	842	982	50	10
quick.	5	5	46	152	264	384	439	549	654	731	842	982	51	10
swap	5(First)	5	46	152	264	384	439	549	654	731	842	982	51	10

Step 11: After swap in partition method, we are going to quickSort method. But when we send the partition method again, the first and last index is equal. So the sorting is done. The last counters and sorted array is below :

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
46	152	264	384	439	549	654	731	842	982

Comparison : 51
Displacement : 10

Sorting 4.c Array C (Integer Array -12 Elements) – Quick Sort Algorithm

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
5	2	13	9	1	7	6	8	1	15	4	11

Step 1: In this sorting, I will use “up” and “down” for partition.
Pivot is now table[first] = 5
First = 0, Last = 11

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	C	D
quick.	0(First)	11	5	2	13	9	1	7	6	8	1	15	4	11	0	0
part.	1	11	5	2	13	9	1	7	6	8	1	15	4	11	1	0
part.	2	11	5	2	13	9	1	7	6	8	1	15	4	11	2	0
part.	2	10	5	2	13	9	1	7	6	8	1	15	4	11	3	0
swap	2	10	5	2	4	9	1	7	6	8	1	15	13	11	3	2
part.	3	11(Last)	5	2	4	9	1	7	6	8	1	15	13	11	4	2
part.	3	9	5	2	4	9	1	7	6	8	1	15	13	11	5	2
part.	3	8	5	2	4	9	1	7	6	8	1	15	13	11	6	2
swap	3	8	5	2	4	1	1	7	6	8	9	15	13	11	6	4
part.	4	11(Last)	5	2	4	1	1	7	6	8	9	15	13	11	7	4
part.	5	11(Last)	5	2	4	1	1	7	6	8	9	15	13	11	8	4
part.	5	7	5	2	4	1	1	7	6	8	9	15	13	11	9	4
part.	5	6	5	2	4	1	1	7	6	8	9	15	13	11	10	4
part.	5	5	5	2	4	1	1	7	6	8	9	15	13	11	11	4
part.	5	4	5	2	4	1	1	7	6	8	9	15	13	11	12	4
swap	0(F)	4	1	2	4	1	5	7	6	8	9	15	13	11	12	6

Step 2: The piv Index is returned down which is 4. So swap first and down index.
After the swap, we are going to back the quickSort. Now our pivot is table[0] = 1.
First = 0, Last = 3

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	C	D
quick.	0(First)	3(Last)	1	2	4	1	5	7	6	8	9	15	13	11	12	6
part.	1	3	1	2	4	1	5	7	6	8	9	15	13	11	13	6
swap	1	3	1	1	4	2	5	7	6	8	9	15	13	11	13	8
part.	2	3(Last)	1	1	4	2	5	7	6	8	9	15	13	11	14	8
part.	2	2	1	1	4	2	5	7	6	8	9	15	13	11	15	8
part.	2	1	1	1	4	2	5	7	6	8	9	15	13	11	16	8
swap	0(First)	1	1	1	4	2	5	7	6	8	9	15	13	11	16	8

Step 3: The piv Index is returned down which is 1. So swap first and down index.
After the swap, we are going to back the quickSort. Now our pivot is table[2] = 4.
First = 2, Last = 3

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	C	D
quick	2(First)	3(Last)	1	1	4	2	5	7	6	8	9	15	13	11	16	8
part.	2	3(Last)	1	1	4	2	5	7	6	8	9	15	13	11	17	8
part.	3	3	1	1	2	4	5	7	6	8	9	15	13	11	18	8
swap	2(First)	3	1	1	2	4	5	7	6	8	9	15	13	11	18	10

Step 4: The piv Index is returned down which is 3. So swap first and down index.
After the swap, we are going to back the quickSort. Now our pivot is table[5] = 7.
First = 5 ,Last = 11 (The other quickSort recursive method)

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	C	D
quick	5(First)	11(Last)	1	1	2	4	5	7	6	8	9	15	13	11	18	10
part.	6	11(Last)	1	1	2	4	5	7	6	8	9	15	13	11	19	10
part.	7	11(Last)	1	1	2	4	5	7	6	8	9	15	13	11	20	10
part.	7	10	1	1	2	4	5	7	6	8	9	15	13	11	21	10
part.	7	9	1	1	2	4	5	7	6	8	9	15	13	11	22	10
part.	7	8	1	1	2	4	5	7	6	8	9	15	13	11	23	10
part.	7	7	1	1	2	4	5	7	6	8	9	15	13	11	24	10
part.	7	6	1	1	2	4	5	7	6	8	9	15	13	11	25	10
swap	5(First)	6	1	1	2	4	5	6	7	8	9	15	13	11	25	12

Step 5: The piv Index is returned down which is 6. So swap first and down index.
After the swap, we are going to back the quickSort. Now our pivot is table[7] = 8.
First = 7 Last = 11

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	C	D
quick	7(First)	11(Last)	1	1	2	4	5	6	7	8	9	15	13	11	25	12
part.	8	11	1	1	2	4	5	6	7	8	9	15	13	11	26	12
part.	8	10	1	1	2	4	5	6	7	8	9	15	13	11	27	12
part.	8	9	1	1	2	4	5	6	7	8	9	15	13	11	28	12
part.	8	8	1	1	2	4	5	6	7	8	9	15	13	11	29	12
part.	8	7	1	1	2	4	5	6	7	8	9	15	13	11	30	12
swap	7(First)	7	1	1	2	4	5	6	7	8	9	15	13	11	30	12

Step 6: The piv Index is returned down which is 7. So swap first and down index.
After the swap, we are going to back the quickSort. Now our pivot is table[8] = 9.
First = 8 Last = 11

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	C	D
quick	8(First)	11(Last)	1	1	2	4	5	6	7	8	9	15	13	11	30	12
part.	9	11	1	1	2	4	5	6	7	8	9	15	13	11	31	12
part.	9	10	1	1	2	4	5	6	7	8	9	15	13	11	32	12
part.	9	9	1	1	2	4	5	6	7	8	9	15	13	11	33	12
part.	9	8	1	1	2	4	5	6	7	8	9	15	13	11	34	12
swap	8(First)	8	1	1	2	4	5	6	7	8	9	15	13	11	34	12

Step 7: The piv Index is returned down which is 8. So swap first and down index.
After the swap, we are going to back the quickSort. Now our pivot is table[9] = 15.
First = 9 Last = 11

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	C	D
quick	9(First)	11(Last)	1	1	2	4	5	6	7	8	9	15	13	11	34	12
part.	10	11	1	1	2	4	5	6	7	8	9	15	13	11	35	12
part.	11	11	1	1	2	4	5	6	7	8	9	15	13	11	36	12
swap	9(First)	11	1	1	2	4	5	6	7	8	9	11	13	15	36	14

Step 8: The piv Index is returned down which is 11. So swap first and down index.
After the swap, we are going to back the quickSort. Now our pivot is table[9] = 11.
First = 9 Last = 10

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	C	D
quick	9(First)	10(Last)	1	1	2	4	5	6	7	8	9	11	13	15	36	14
part.	10	10	1	1	2	4	5	6	7	8	9	11	13	15	37	14
part.	10	9	1	1	2	4	5	6	7	8	9	11	13	15	38	14
swap	9(First)	9	1	1	2	4	5	6	7	8	9	11	13	15	38	14

Step 9: And final operation is, I send the (arr,first,last). So the privIndex is 9 which is returned down in partition method. So our base case is if first > last, do not anything and exit the method. So sorting is done. The last counters and sorted array are below :
Comparison : 38
Displacement : 14

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
1	1	2	4	5	6	7	8	9	11	13	15

Sorting 4.d Array D (Char Array -12 Elements) – Quick Sort Algorithm

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
S	B	I	M	H	Q	C	L	R	E	P	K

Step 1: In this sorting, I will use “up” and “down” for partition.
Pivot is now table[first] = S
First = 0 Last = 11

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	C	D
quick.	0	11(Last)	S	B	I	M	H	Q	C	L	R	E	P	K	0	0
part.	1	11(Last)	S	B	I	M	H	Q	C	L	R	E	P	K	1	0
part.	2	11(Last)	S	B	I	M	H	Q	C	L	R	E	P	K	2	0
part.	3	11(Last)	S	B	I	M	H	Q	C	L	R	E	P	K	3	0
part.	4	11(Last)	S	B	I	M	H	Q	C	L	R	E	P	K	4	0
part.	5	11(Last)	S	B	I	M	H	Q	C	L	R	E	P	K	5	0
part.	6	11(Last)	S	B	I	M	H	Q	C	L	R	E	P	K	6	0
part.	7	11(Last)	S	B	I	M	H	Q	C	L	R	E	P	K	7	0
part.	8	11(Last)	S	B	I	M	H	Q	C	L	R	E	P	K	8	0
part.	9	11(Last)	S	B	I	M	H	Q	C	L	R	E	P	K	9	0
part.	10	11(Last)	S	B	I	M	H	Q	C	L	R	E	P	K	10	0
part.	11	11(Last)	S	B	I	M	H	Q	C	L	R	E	P	K	11	0
swap	0(F)	11	K	B	I	M	H	Q	C	L	R	E	P	S	11	2

Step 2: The piv Index is returned down which is 11. So swap first and down index. After the swap, we are going to back the quickSort. Now our pivot is table[0] = K
First = 0 Last = 10

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	C	D
quick.	0	10(Last)	K	B	I	M	H	Q	C	L	R	E	P	S	11	2
part.	1	10(Last)	K	B	I	M	H	Q	C	L	R	E	P	S	12	2
part.	2	10(Last)	K	B	I	M	H	Q	C	L	R	E	P	S	13	2
part.	3	9	K	B	I	M	H	Q	C	L	R	E	P	S	14	2
swap	3	9	K	B	I	E	H	Q	C	L	R	M	P	S	14	4
part.	4	10(Last)	K	B	I	E	H	Q	C	L	R	M	P	S	15	4
part.	5	10(Last)	K	B	I	E	H	Q	C	L	R	M	P	S	16	4
part.	5	8	K	B	I	E	H	Q	C	L	R	M	P	S	17	4
part.	5	7	K	B	I	E	H	Q	C	L	R	M	P	S	18	4
part.	5	6	K	B	I	E	H	Q	C	L	R	M	P	S	19	4
swap	5	6	K	B	I	E	H	C	Q	L	R	M	P	S	19	6
part.	6	10(Last)	K	B	I	E	H	C	Q	L	R	M	P	S	20	6
part.	6	5	K	B	I	E	H	C	Q	L	R	M	P	S	21	6
swap	0(F)	5	C	B	I	E	H	K	Q	L	R	M	P	S	21	8

Step 3: The piv Index is returned down which is 15. So swap first and down index. After the swap, we are going to back the quickSort. Now our pivot is table[0] = C
First = 0 Last = 4

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	C	D
quick.	0(F)	4(Last)	C	B	I	E	H	K	Q	L	R	M	P	S	21	8
part.	1	4(Last)	C	B	I	E	H	K	Q	L	R	M	P	S	22	8
part.	2	4(Last)	C	B	I	E	H	K	Q	L	R	M	P	S	23	8
part.	2	3	C	B	I	E	H	K	Q	L	R	M	P	S	24	8
part.	2	2	C	B	I	E	H	K	Q	L	R	M	P	S	25	8
part.	2	1	C	B	I	E	H	K	Q	L	R	M	P	S	26	8
swap	0(F)	1	B	C	I	E	H	K	Q	L	R	M	P	S	26	10

Step 4: The piv Index is returned down which is 1. So swap first and down index. After the swap, we are going to back the quickSort. Now we enter the 2.quick recursive method. And our pivot is table[2] = I
First = 2 Last = 4

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	C	D
quick.	2(F)	4(Last)	B	C	I	E	H	K	Q	L	R	M	P	S	26	10
part.	3	4(Last)	B	C	I	E	H	K	Q	L	R	M	P	S	27	10
part.	4	4	B	C	I	E	H	K	Q	L	R	M	P	S	28	10
swap	2(F)	4	B	C	H	E	I	K	Q	L	R	M	P	S	28	12

Step 5: The piv Index is returned down which is 4. So swap first and down index.
After the swap, we are going to back the quickSort. Now we enter the 1.quick recursive method. And our pivot is table[2] = H
First = 2 Last = 3

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	C	D
quick.	2(F)	3(Last)	B	C	H	E	I	K	Q	L	R	M	P	S	28	12
part.	3	3(Last)	B	C	H	E	I	K	Q	L	R	M	P	S	29	12
part.	3	3	B	C	H	E	I	K	Q	L	R	M	P	S	30	12
swap	2(F)	3	B	C	E	H	I	K	Q	L	R	M	P	S	30	14

Step 6: The piv Index is returned down which is 3. So swap first and down index.
After the swap, we are going to back the quickSort. Now we enter the 2.quick recursive method. And our pivot is table[6] = Q
First = 6 Last = 10

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	C	D
quick.	6(F)	10(Last)	B	C	E	H	I	K	Q	L	R	M	P	S	30	14
part.	7	10(Last)	B	C	E	H	I	K	Q	L	R	M	P	S	31	14
part.	8	10(Last)	B	C	E	H	I	K	Q	L	R	M	P	S	32	14
swap	8	10	B	C	E	H	I	K	Q	L	P	M	R	S	32	16
part.	9	10(Last)	B	C	E	H	I	K	Q	L	P	M	R	S	33	16
part.	10	10(Last)	B	C	E	H	I	K	Q	L	P	M	R	S	34	16
part.	10	9	B	C	E	H	I	K	Q	L	P	M	R	S	35	16
swap	6(F)	9	B	C	E	H	I	K	M	L	P	Q	R	S	35	18

Step 7: The piv Index is returned down which is 9. So swap first and down index.
After the swap, we are going to back the quickSort. Now we enter the 1.quick recursive method. And our pivot is table[6] = M
First = 6 Last = 8

Met.	Up	Down	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	C	D
quick.	6(F)	8(Last)	B	C	E	H	I	K	M	L	P	Q	R	S	35	18
part.	7	8(Last)	B	C	E	H	I	K	M	L	P	Q	R	S	36	18
part.	8	8(Last)	B	C	E	H	I	K	M	L	P	Q	R	S	37	18
part.	8	7	B	C	E	H	I	K	M	L	P	Q	R	S	38	18
swap	6(F)	7	B	C	E	H	I	K	L	M	P	Q	R	S	38	20

Step 8: The piv Index is returned down which is 7. So swap first and down index.
After the swap, we are going to back the quickSort. And all recursive methods are done because first is greater or equal than last. So the last counters and sorted array are below:

Comparison : 38
Displacement : 20

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
B	C	E	H	I	K	L	M	P	Q	R	S