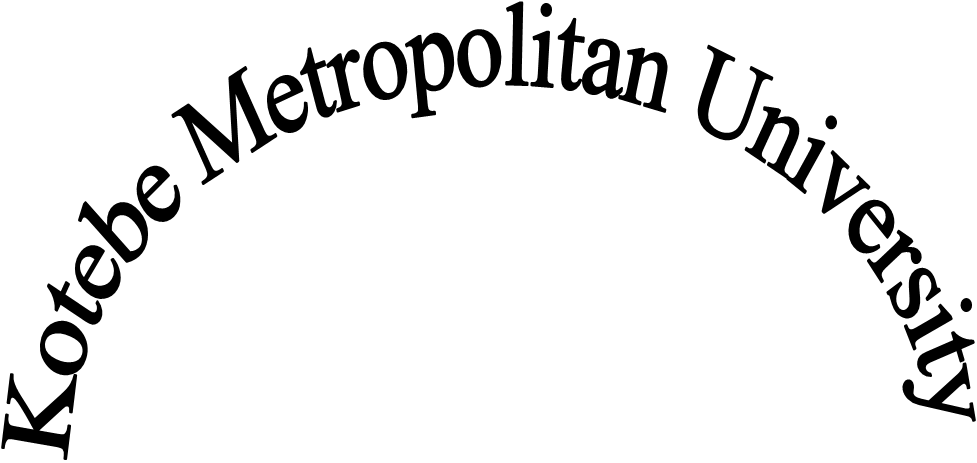
****

****

**DEPARTMENT: COMPUTER SCIENCE**

**COURSE TITTLE: Computer Organization and Architecture**

**COURSE CODE: CoSc 2092**

**GROUP ASSIGNMENT**

**STUDENT NAME ID ID NO**

***🡪Tilahun Tegenaw 🡪CNCS/UR17772/12***

***🡪 Lema Bewket 🡪CNCS/UR15660/12***

***🡪 Lamesgen Enawgaw🡪CNCS/UR15659/12***

***🡪Asmari Enquanhone 🡪CNCS/UR14958/12***

***🡪Ataklti Solomon 🡪CNCS/UR14962/12***

**Shell Sort Algorithm**

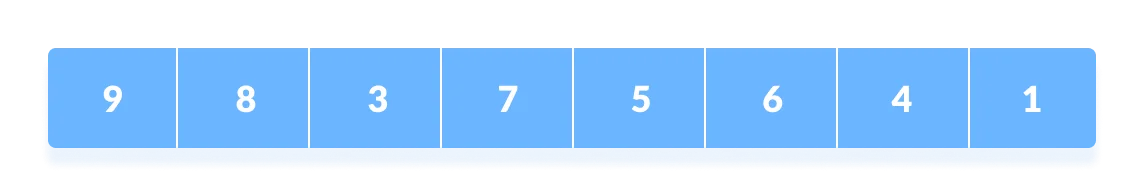
shell sort algorithm and its implementation in C++.

Shell sort is a highly efficient sorting algorithm and is based on insertion sort algorithm. This algorithm avoids large shifts as in case of insertion sort, if the smaller value is to the far right and has to be moved to the far left.

This algorithm uses insertion sort on a widely spread elements, first to sort them and then sorts the less widely spaced elements. This spacing is termed as interval.

Shell sort is a generalized version of the [insertion sort algorithm](https://www.programiz.com/dsa/insertion-sort). It first sorts elements that are far apart from each other and successively reduces the interval between the elements to be sorted.

The interval between the elements is reduced based on the sequence used. Some of the optimal sequences that can be used in the shell sort algorithm are:



**Shell's original sequence**

:

N/2 , N/4 , …, 1

**Knuth's increments**

:

1

, 4, 13, …,

(3

– 1) / 2

**Sedgewick's increments**

:

1

, 8, 23, 77, 281, 1073, 4193, 16577...4j+1+ 3·2j+

1

**Hibbard's increments**

:

1

, 3, 7, 15, 31, 63, 127, 255, 511…

**Papernov & Stasevich increment**

:

1

, 3, 5, 9, 17, 33,

65,...

**Pratt**

:

1

, 2, 3, 4, 6, 9, 8, 12, 18, 27, 16, 24, 36,

54, 81....

The performance of the shell sort depends on the type of sequence used for

a given input array.

k

# Working of Shell Sort

1. Suppose, we need to sort the following array.

Initial array

1. We are using the shell's original sequence (N/2, N/4, ...1 ) as intervals in our algorithm.

In the first loop, if the array size is N = 8 then, the elements lying at the interval of N/2 = 4 are compared and swapped if they are not in order.

a

.

The 0th element is compared with the

th

4

element.

b

.

If the 0th element is greater than the

4

th

one then, the

4

th

element is first stored

in

temp

variable and the

0

th

element (ie. greater element) is stored in the

4

th

position and the element stored in

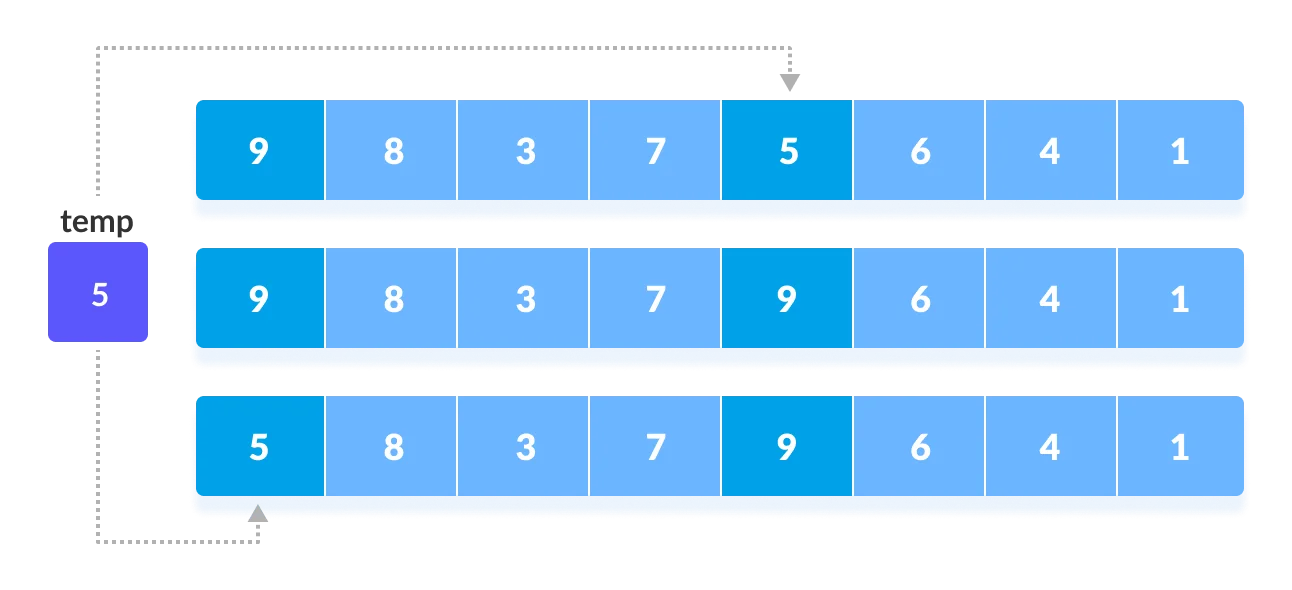
temp

is stored in the

0

th

position.



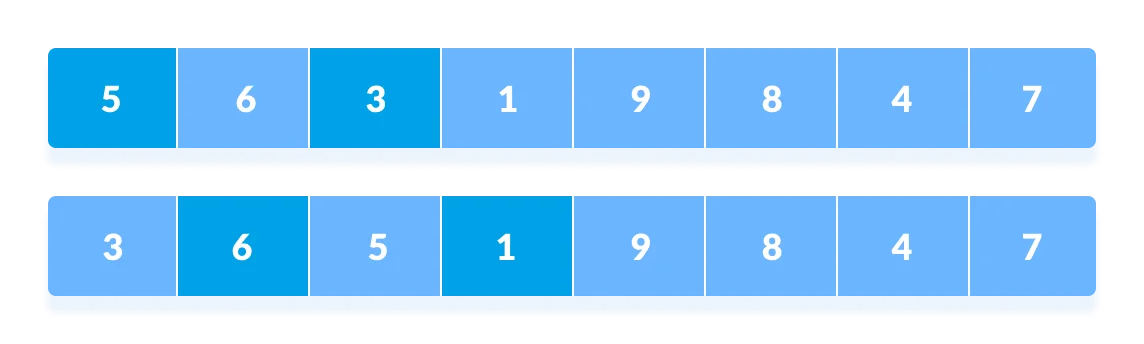
Rearrange the elements at n/2 interval

This process goes on for all the remaining elements.



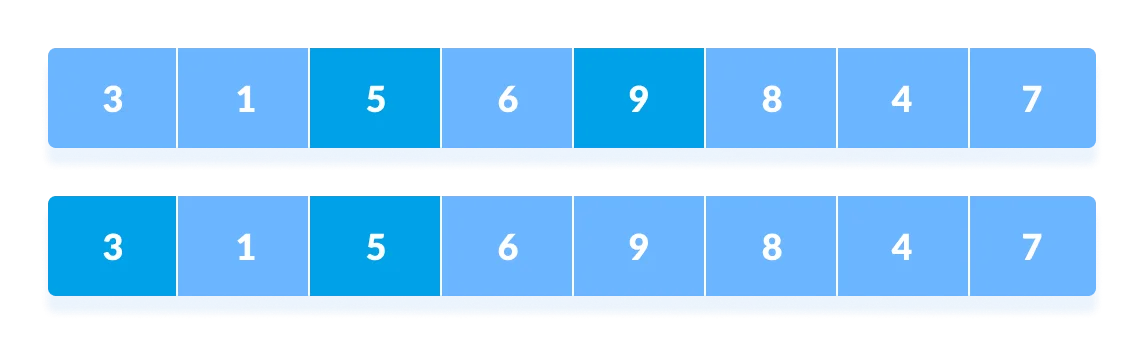
Rearrange all the elements at n/2 interval

1. In the second loop, an interval of N/4 = 8/4 = 2 is taken and again the elements lying at these intervals are sorted.



Rearrange the elements at n/4 interval

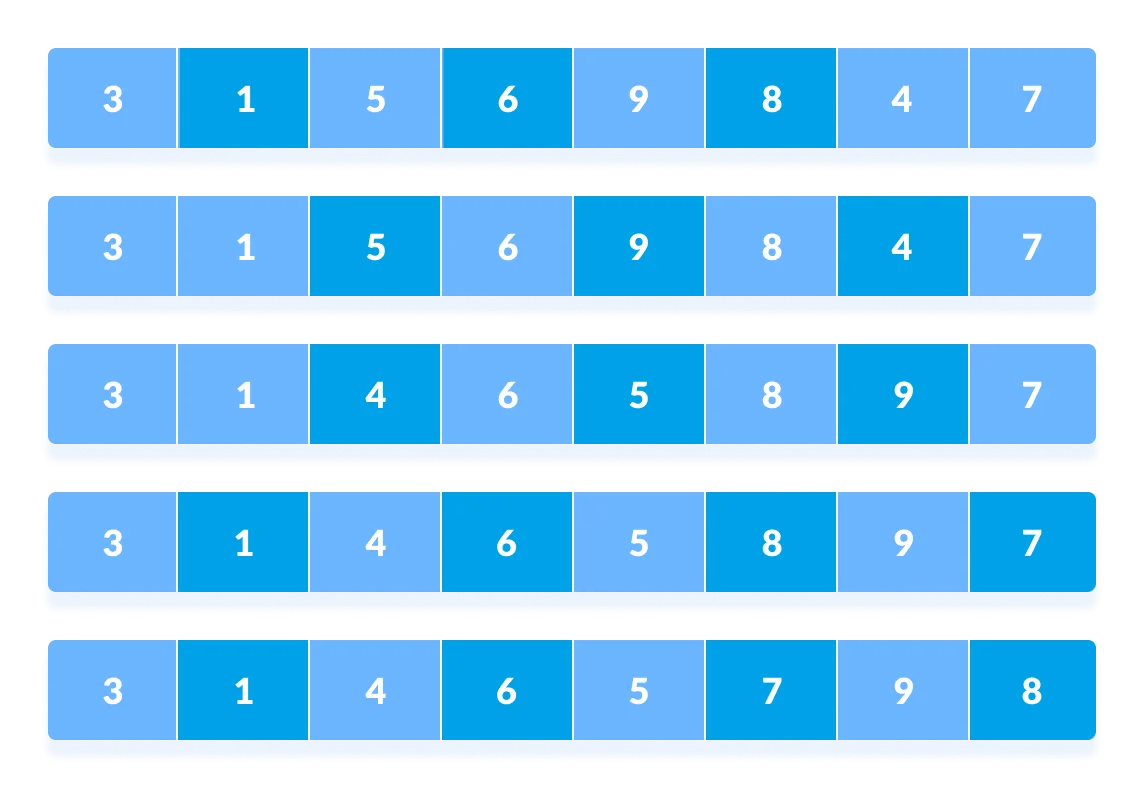
You might get confused at this point.



All the elements in the array lying at the current interval are compared.

The elements at 4th and 2nd position are compared. The elements at 2nd and 0th position are also compared. All the elements in the array lying at the current interval are compared.

1. The same process goes on for remaining elements.



Rearrange all the elements at n/4 interval

5

.

Finally, when the interval is

N/8 = 8/8 =1

then the array elements lying at the interval of

1 are sorted. The array is now completely sorted.



Rearrange the elements at n/8 interval