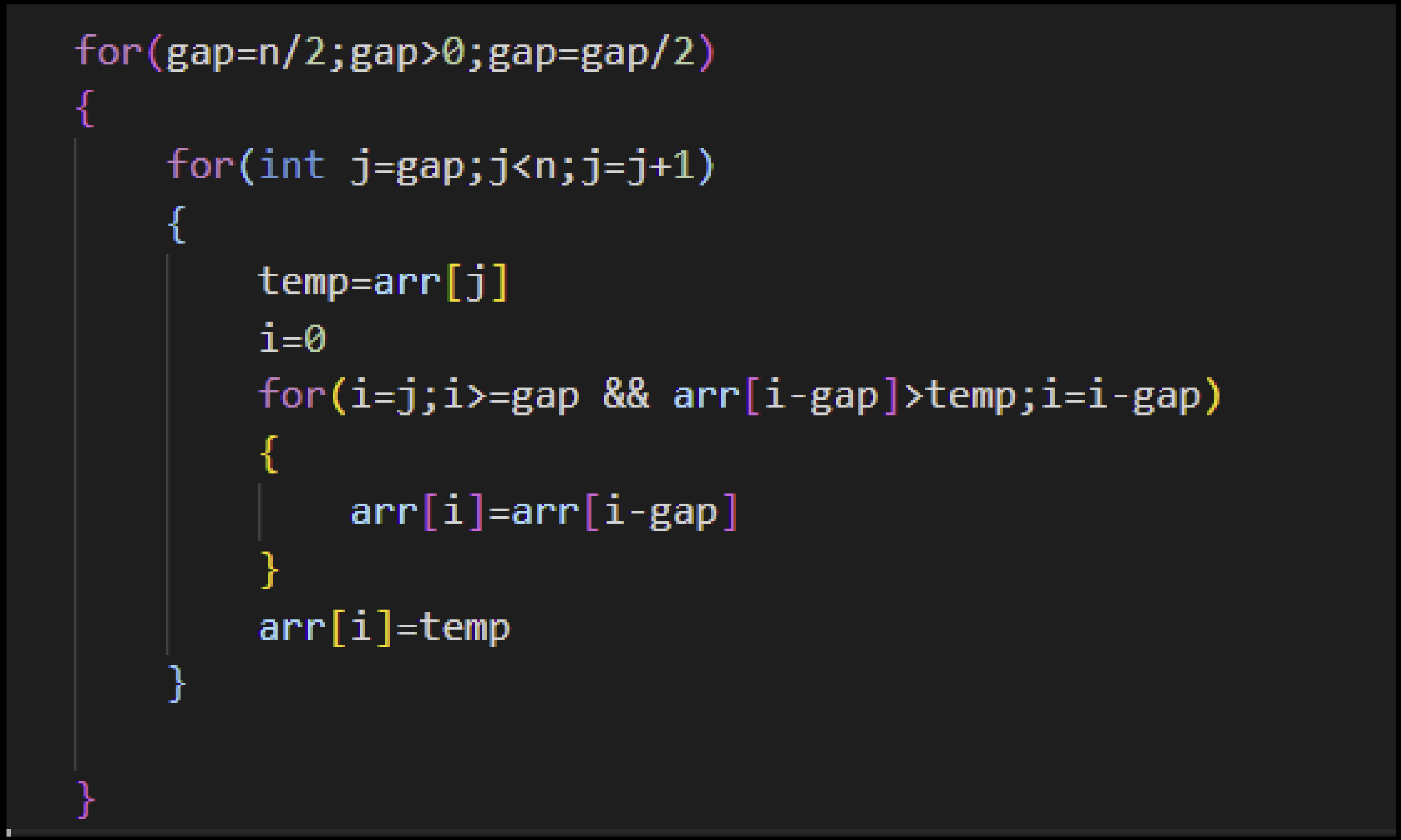
Shell Sort .

Introduction

- The shell sort is an improved version of the straight insertion sort. In shell sorting algorithm we compare elements that are a distance apart rather than adjacent. We start by comparing elements that are at a certain distance apart which is the “gap” so if there are N elements then we start the value gap < N. Meaning in the next pass the gap will be N/2 and so on. In each pass we keep reducing the value of the gap until we reach the last pass when gap is 1. Note that gap is equal to floor(N/2). In the last pass shell sort is like an insertion sort and like this eventually the list will be sorted. The Shell sort is named after Donald Shell, the algorithm's creator, who published it in 1959. Shell sort is a series of interleaved insertion sorts based on a sequence of increments.

Shell Sort Pseudocode.



```
for(gap=n/2;gap>0;gap=gap/2)
{
    for(int j=gap;j<n;j=j+1)
    {
        temp=arr[j]
        i=0
        for(i=j;i>=gap && arr[i-gap]>temp;i=i-gap)
        {
            arr[i]=arr[i-gap]
        }
        arr[i]=temp
    }

}
```

Complexity Analysis.

In this sorting algorithm, there are three for loops. In the first for loop the running time will be (logn). But as it depends on n (size of the number of elements of an array) the running time will be (nlogn). In the second for loop the running time will be n as the value of j incremented by each iteration. And in the third one the value of I is decremented by each iteration, so running time will be n. So overall the running time will be **O(nlogn)** which is the best case. Best case depends if my number is almost sorted and finite number of inputs.

Worst case is **O(n^2)** & Average case is **O(n*log(n)^2**). The worst and average case depends on the user input and the size of the array.

Space complexity of this algorithm is **O(1)** as it doesn't require any external memory.

Complexity Analysis.

```
Best case Analysis:
It only occurs when the array is sorted in a
decreasing order i.e. 802 600 300 200 99 55 32. . . .

{
    for(gap=n/2;gap>0;gap=gap/2)
    {
        for(int j=gap;j<n;j=j+1)
        {
            temp=arr[j]
            i=0
            for(i=j;i>=gap && arr[i-gap]>temp;i=i-gap)
            {
                arr[i]=arr[i-gap]
            }
            arr[i]=temp
        }

    }
}
```

nlogn

n

n

2n + nlogn

O(nlogn)

Attributes Of Shell Sort…

As far we already know that Shell sort is the advanced form of "Insertion sort" or "Bubble sort" and even faster. It's an **In-place comparison-based** type sorting algorithm. Shell sort is an **unstable** sorting technique because it can change the relative orders of elements with equal values. This sorting algorithm falls under the **adaptive** sort family as it takes the advantage of the existing order of the input to make it the best and better form. The more presorted the inputs are, the faster the sorting takes place in an array. As a result , Time complexity might vary and most importantly it depends on user inputs. Thus, the performance of existing sorting algorithm can be improved by considering the order of the inputs.

Shell sort is an **offline algorithm** which means all inputs are available to the algorithm and processed simultaneously by the algorithm. In short, during the simulation, no inputs are required. For example – Selection sort and Insertion sort are both offline algorithms.

In Shell sort, we can work with Positive and Negative Integer Numbers. We can also work with Positive and Negative Floating-Point Numbers.

Simulation of Shell Sort.

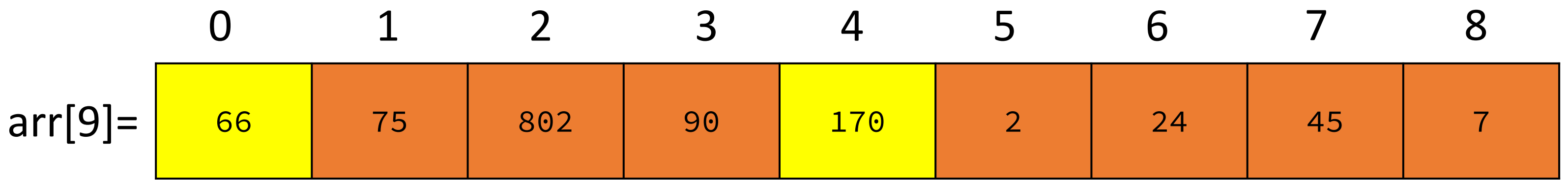
	0	1	2	3	4	5	6	7	8
arr[9]=	170	75	802	90	66	2	24	45	7

Simulation of Shell Sort.

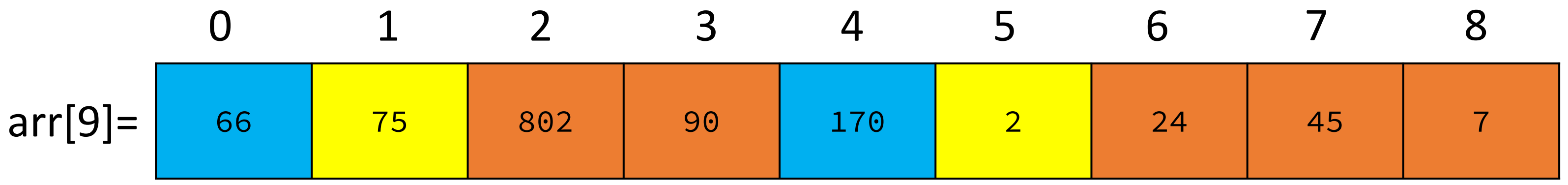
```
Pass 1 :
Gap=n/2
   =9/2
   =4.5
   =4
```

	0	1	2	3	4	5	6	7	8
arr[9]=	170	75	802	90	66	2	24	45	7

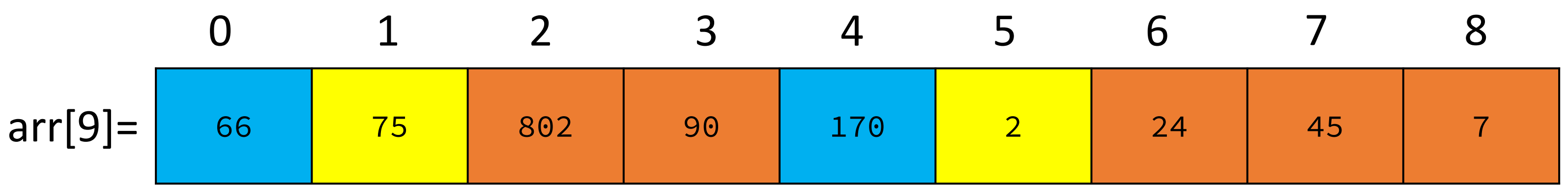
Simulation of Shell Sort.



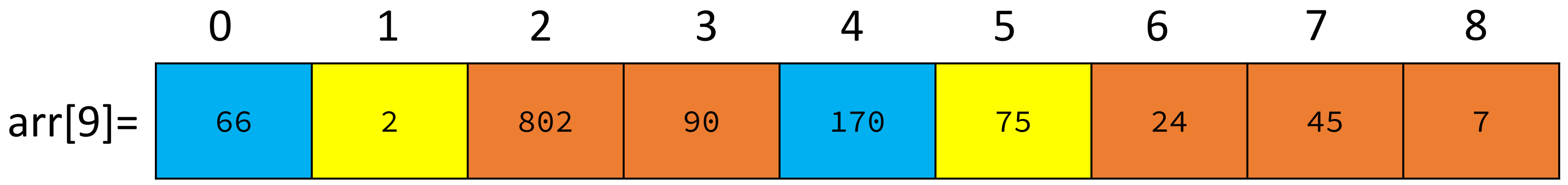
Simulation of Shell Sort.



Simulation of Shell Sort.



Simulation of Shell Sort.



	0	1	2	3	4	5	6	7	8
arr[9]=	66	2	802	90	170	75	24	45	7

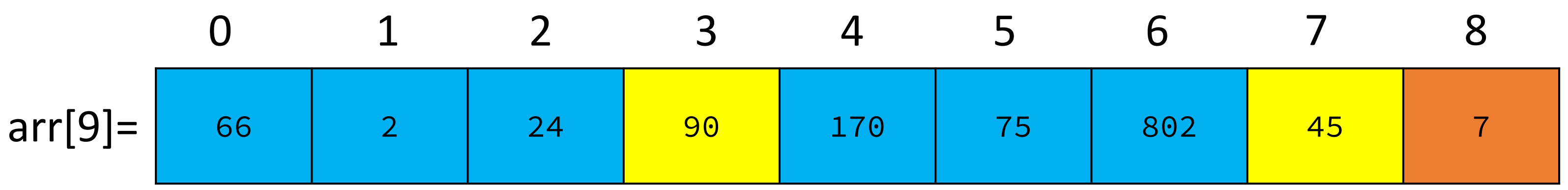
Simulation of Shell Sort.

	0	1	2	3	4	5	6	7	8
arr[9]=	66	2	802	90	170	75	24	45	7

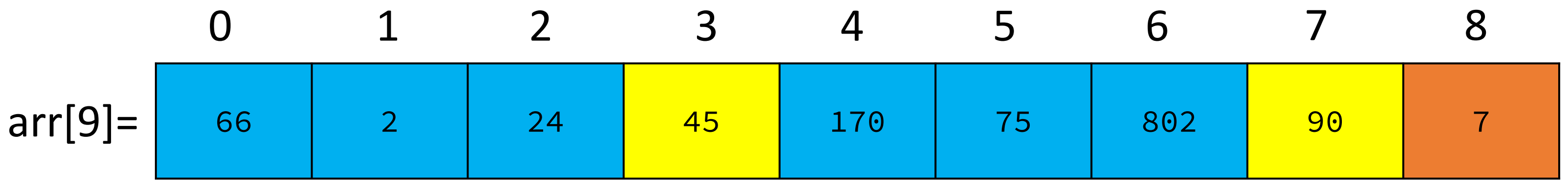
Simulation of Shell Sort.

	0	1	2	3	4	5	6	7	8
arr[9]=	66	2	24	90	170	75	802	45	7

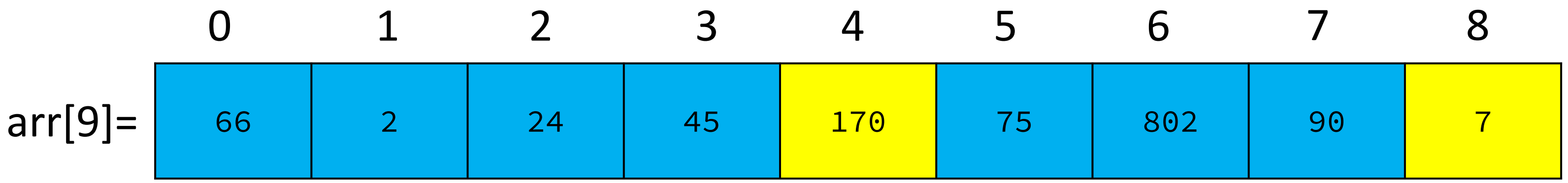
Simulation of Shell Sort.



Simulation of Shell Sort.



Simulation of Shell Sort.

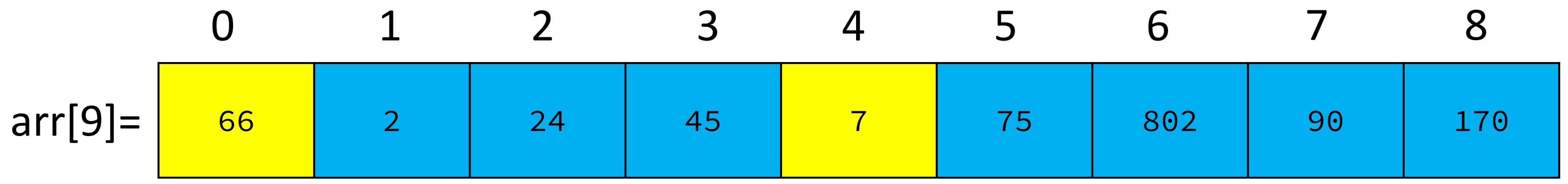


	0	1	2	3	4	5	6	7	8
arr[9]=	66	2	24	45	170	75	802	90	7

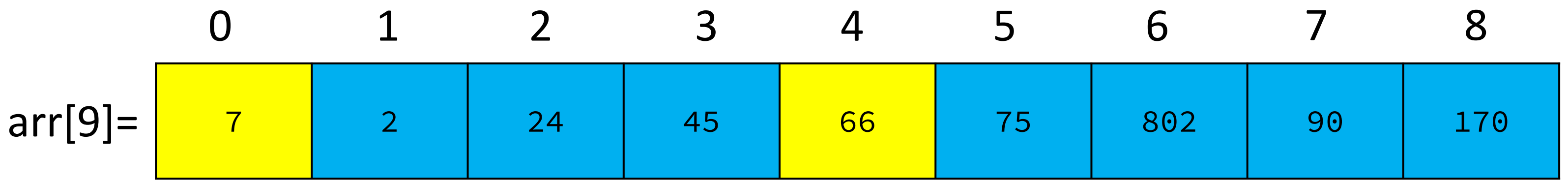
Simulation of Shell Sort.

	0	1	2	3	4	5	6	7	8
arr[9]=	66	2	24	45	7	75	802	90	170

Simulation of Shell Sort.



Simulation of Shell Sort.

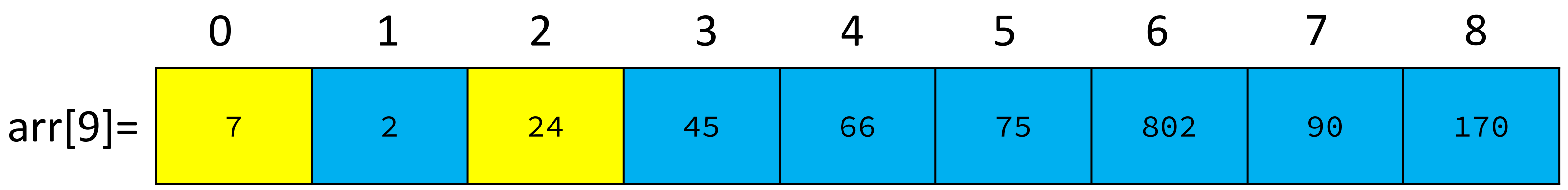


Simulation of Shell Sort.

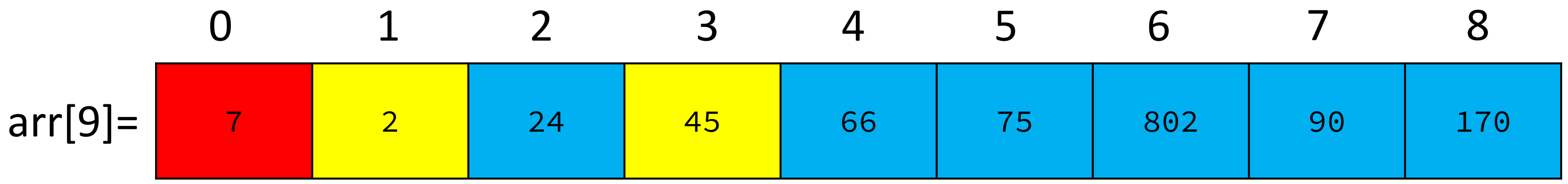
	0	1	2	3	4	5	6	7	8
arr[9]=	7	2	24	45	66	75	802	90	170

Simulation of Shell Sort.

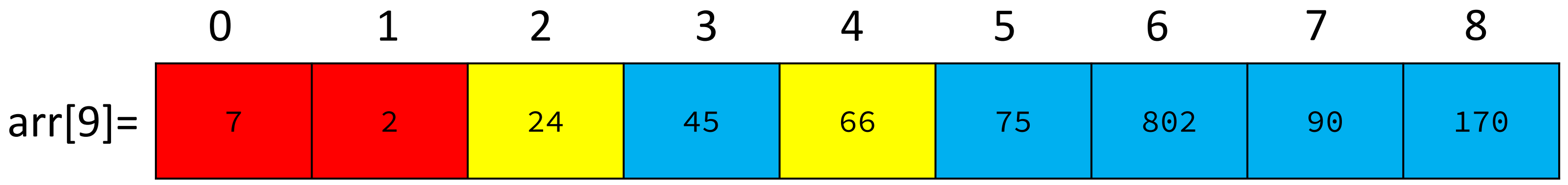
```
Gap=gap/2
   =4/2
   =2
```



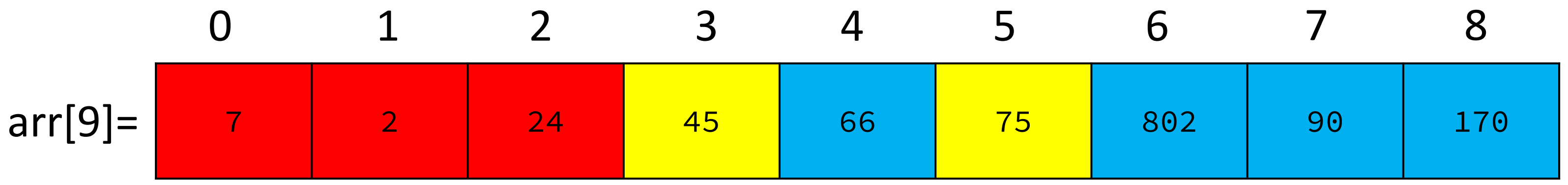
Simulation of Shell Sort.



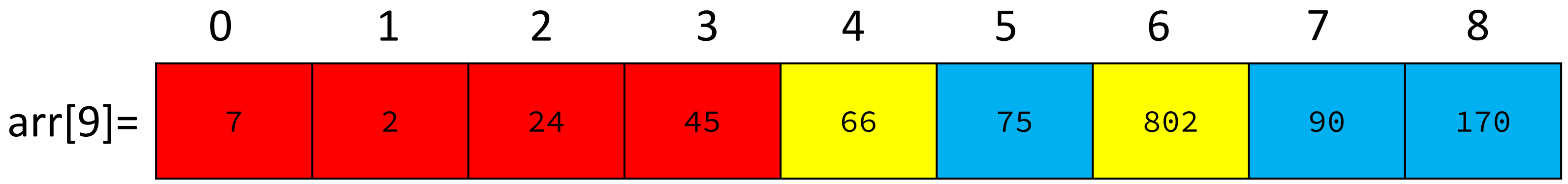
Simulation of Shell Sort.



Simulation of Shell Sort.

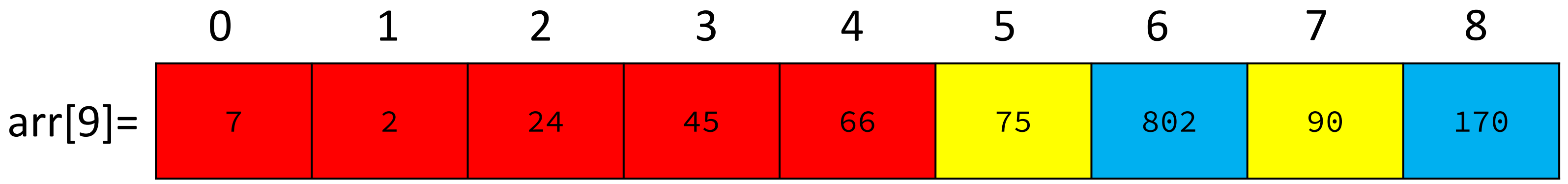


Simulation of Shell Sort.



	0	1	2	3	4	5	6	7	8
arr[9]=	7	2	24	45	66	75	802	90	170

Simulation of Shell Sort.



Simulation of Shell Sort.

	0	1	2	3	4	5	6	7	8
arr[9]=	7	2	24	45	66	75	802	90	170

Simulation of Shell Sort.

	0	1	2	3	4	5	6	7	8
arr[9]=	7	2	24	45	66	75	170	90	802

Simulation of Shell Sort.

	0	1	2	3	4	5	6	7	8
arr[9]=	7	2	24	45	66	75	170	90	802

Simulation of Shell Sort.

	0	1	2	3	4	5	6	7	8
arr[9]=	7	2	24	45	66	75	170	90	802

Simulation of Shell Sort.

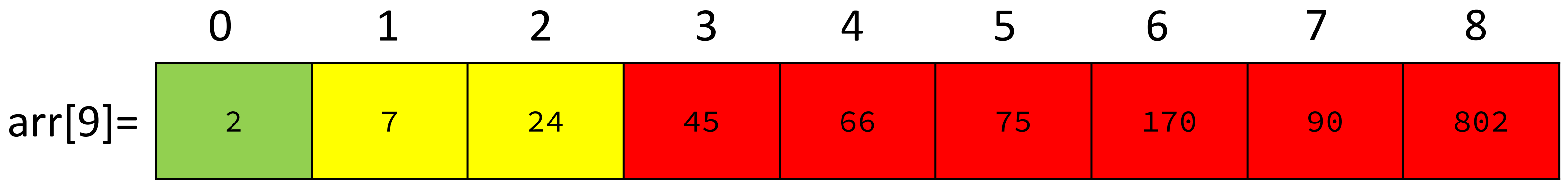
```
Gap=gap/2
   =2/2
   =1
```

	0	1	2	3	4	5	6	7	8
arr[9]=	7	2	24	45	66	75	170	90	802

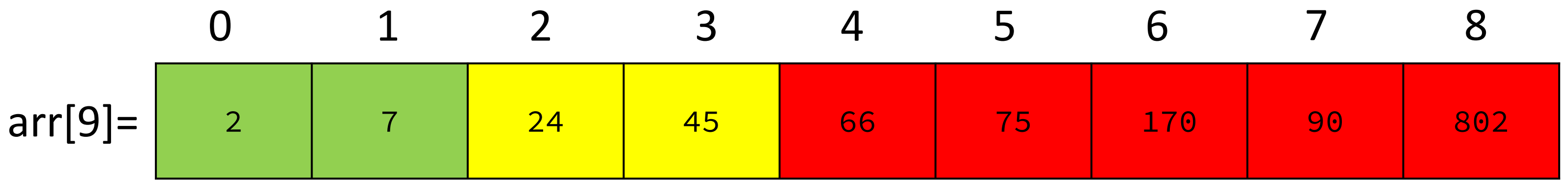
Simulation of Shell Sort.

	0	1	2	3	4	5	6	7	8
arr[9]=	2	7	24	45	66	75	170	90	802

Simulation of Shell Sort.



Simulation of Shell Sort.



Simulation of Shell Sort.

	0	1	2	3	4	5	6	7	8
arr[9]=	2	7	24	45	66	75	170	90	802

Simulation of Shell Sort.

	0	1	2	3	4	5	6	7	8
arr[9]=	2	7	24	45	66	75	170	90	802

Simulation of Shell Sort.

	0	1	2	3	4	5	6	7	8
arr[9]=	2	7	24	45	66	75	170	90	802

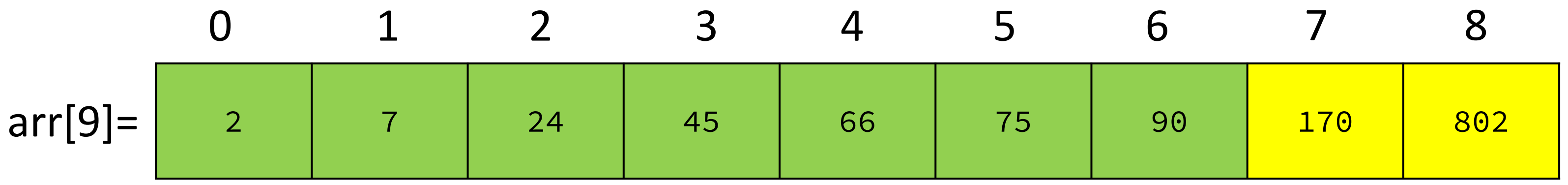
Simulation of Shell Sort.

	0	1	2	3	4	5	6	7	8
arr[9]=	2	7	24	45	66	75	90	170	802

Simulation of Shell Sort.

	0	1	2	3	4	5	6	7	8
arr[9]=	2	7	24	45	66	75	90	170	802

Simulation of Shell Sort.



Simulation of Shell Sort.

	0	1	2	3	4	5	6	7	8
arr[9]=	2	7	24	45	66	75	90	170	802

Pros & Cons Of Shell Sort …

Pros

- ❑ **Much faster algorithm for finite number of elements in an array.**
- ❑ **Faster than Bubble sort & Insertion sort algorithm.**
- ❑ **No auxiliary array required.**
- ❑ **Shell sort avoids large shifts as in case of Insertion sort.**

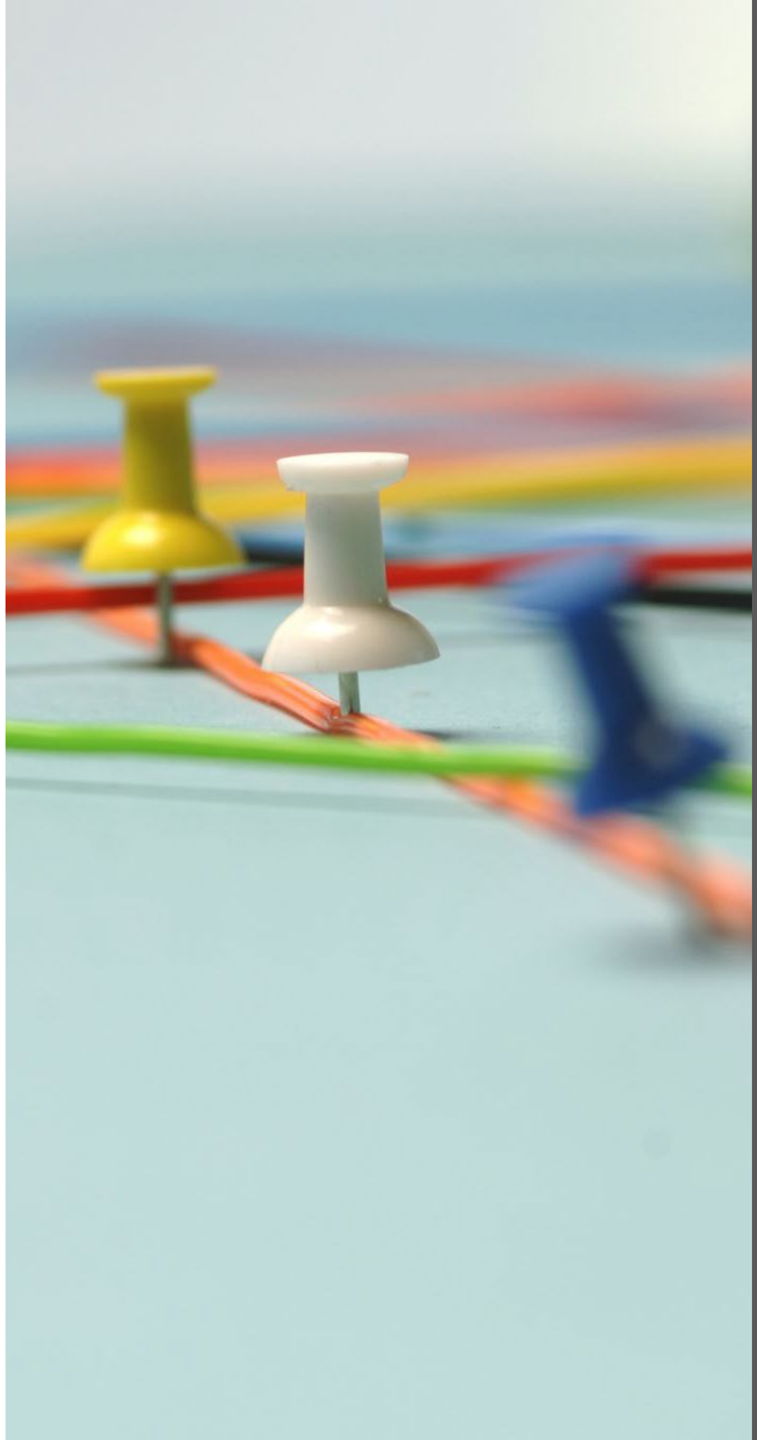
Cons

- ❑ **Complex in structure & a bit difficult to understand.**
- ❑ **Slower than Merge sort, Quick sort & Heap sort algorithm.**

Practical Use..

Applications of Shell sort :-

- *Shell sort performs more operations and has higher cache ration than quicksort.*
- *The C standard library uses shell sort, in lieu of quicksort function when dealing with embedded systems.*
- *Shell sort is used in the Linux kernel because it does not use the call stack.*
- *Compressors such like – bzip2 is used to avoid problems that could come when sorting algorithm exceeds a language's recursion depth.*



```cpp
#include<iostream>
using namespace std;
void ShellSort(int arr[], int n)
{
    for(int gap=n/2; gap>0; gap /= 2)
    {
        for(int j = gap; j<n; j+=1)
        {
            int temp = arr[j];
            int i=0;

            for(i=j; (i>=gap)&&(arr[i-gap]>temp); i-=gap)
            {
                arr[i] = arr[i-gap];
            }
            arr[i] = temp;
        }
    }
}

int main()
{
    int n;
    cout<<"Enter the size of the array: "<<endl;
    cin>>n;
    int arr[n];

    cout<<"Enter elements ";
    for(int i=0; i<n; i++)
    {
        cin>>arr[i];
    }

    ShellSort(arr, n);
    for(int i=0; i<n; i++)
    {
        cout<<arr[i]<<" ";
    }
   return 0;
}
```