SORTING

DIMINISHING INCREMENTAL SORT "Shell Sort"

This sorting technique is also called Shell sort as it was invented by Donald Shell.

- Shell sort is a highly efficient sorting algorithm and is based on insertion sort algorithm.
- Insertion sort, does the minimum number of comparisons ($n^2/4$), but it is inefficient in moving entries only one position at a time .
- In Shell sort, we modify insertion sort so that it first compares keys apart, then it could sort the entries far apart .
- Afterward, the entries closer together would be sorted .
- Finally, the increment between keys being compared is reduced to one .

  - The algorithm makes multiple passes through the list, and each time sorts a number of equally sized sets using the insertion sort. The size of the set to be sorted gets larger with each pass through the list, until the set consists of the entire list.
  - The items contained in each set are not contiguous - rather, if there are i sets then a set is composed of every i-th element. For example, if there are 3 sets then the first set would
  - Contain the elements located at positions 1, 4, 7 and so on. The second set would contain the elements located at positions 2, 5, 8, and so on; while the third set would contain the items located at positions 3, 6, 9, and so on.

Algorithm Shell sort

```
function shell_sort(arr,n){
gap = floor(n/3);
while(gap > 0)
```

```
{
for(i = gap; i < n; i++)
  {
    temp = arr[i];
   j = i;
   while(j >= gap && arr[j - gap] > temp)
 {
 arr[j] = arr[j - gap];
    j -= gap;
}
arr[j] = temp;
}
gap = floor(gap/2);
}
  return arr;
  }
```

Complexity: The worst-case complexity of the above algorithm is $O(n^2)$. However, it is best of all the $O(n^2)$ algorithms. As a matter of fact, the number of comparisons goes on decreasing as and when we proceed.
Problem: The algorithm has a high complexity.

Example 1:

13		5	30	24		8	88	
~~24~~	31	~~8~~	~~88~~	~~13~~	63	~~5~~	~~30~~	4
	24		8	31		30		
13	~~31~~	5	~~30~~	~~24~~	63	~~8~~	88	3
5	8	13	24	30		31		
~~13~~	~~24~~	~~5~~	~~8~~	~~31~~	63	~~30~~	88	2
					31	63		
5	8	13	24	30	~~63~~	~~31~~	88	1
5	8	13	24	30	31	63	88	Sorted

Example 2:

2	12	3	18		
3	18	2	12	24	2
	3	12			
2	12	3	18	24	1
2	3	12	18	24	Sorted

RADIX SORT

Radix sort algorithm sorts the elements of a given set by considering the digits at various places. The process can be understood by taking an example. If the given set is {23, 34, 67, 89, 123, 63, 39, 212, 90}, then the following steps must be followed in order to generate the sorted set. Step 1 Place elements according to units place in holders having indices from 0 to 9. Figure 1 depicts the first step.

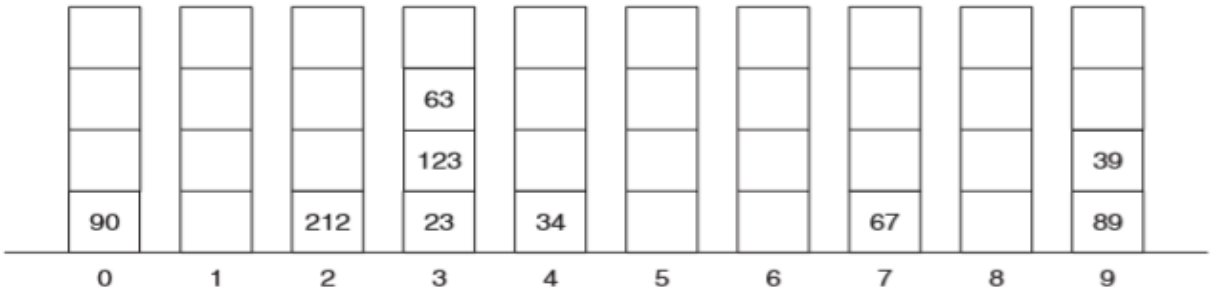

Figure 1: radix sort, elements placed in order of their unit's place

Next, we read the list from left to right. The output of this step would be {90, 212, 23, 123, 63, 34, 67, 89, 39}. In the next step, the above array would be arranged in order of their tens place. The situation is depicted in Fig. 2.

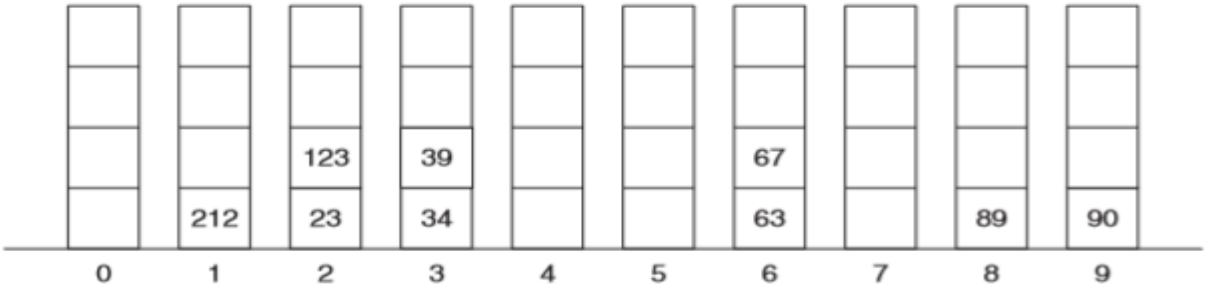

Figure 2: radix sort, elements placed in order of their tens' place

Next, we read the list from left to right. The output of this step would be {212, 23, 123, 34, 39, 63, 67, 89, 90}. In the next step, the above array would be arranged in order of their hundreds place. The situation is depicted in Fig. 3.

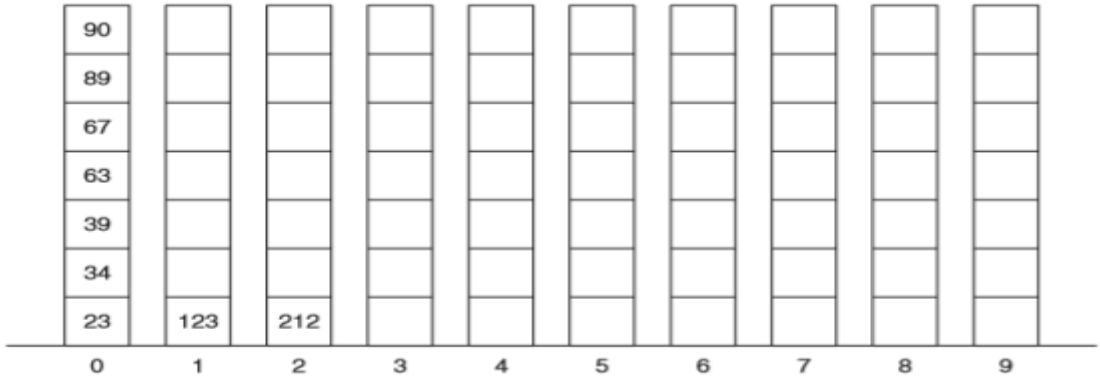

Figure 3: radix sort, elements placed in order of their hundreds place

Next, we read the list from left to right. The output of this step would be {23, 34, 39, 63, 67, 89, 90, 123, 212}.

Algorithm radix sort(a[], n) returns a sorted array

```
k=number of digits in the maximum element of the given array
for(i=1;i<=k;i++)
  {
  Sort the list in accordance with digit di.
  }
```

Complexity: There are k iterations, the complexity of the above algorithm is, therefore, O(N). Problem: The algorithm needs 9 arrays. The number of elements in the array should be at least the maximum number of j digit numbers in the list, for all j.