GoodLuck | Page No
Date

SORTING

A sorting algorithm is an algorithm that puts elements of list in a certain order. The most used order are numerical order & lexicographical order. Efficient sorting is important to optimizing the use of other algorithm that require ~~seee~~ sorted lists to work correctly & for producing human-readable input.

Sorting techiques are categorized into
1) Internal sorting
2) External sorting

Internal sorting takes place in the main memory of a computer
eg: Bubble sort, Insertion sort, shell sort, Quick sort, Heap sort etc
External sorting takes place in the secondary memory of a computer, since the number of objects to be sorted is too large to fit in main memory.
eg: Merge sort

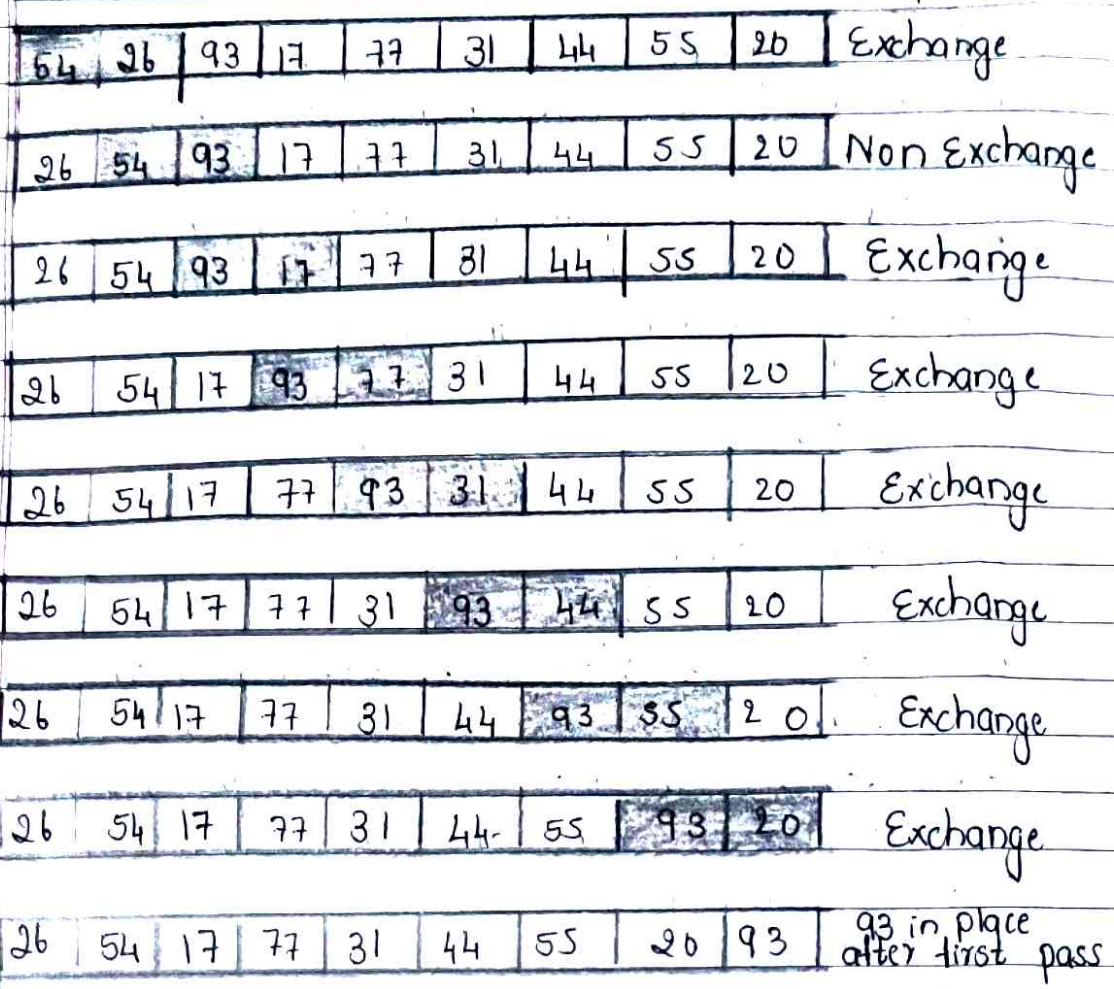
The Bubble sort

The bubble sort makes multiple passes through a list. It compares adjacent item and exchanges those that are out of order. Each pass through the list places the next largest value in it's proper place.

In the following figure the shaded items are being compared to see if they are out of order. If there are n items in the list, then there are n-1 pairs of items that need to be compared on the first pass. It is important to note the largest value in the list is a part of a pair, it will continually be moved along until the pass is complete.

GoodLuck | Page No.
Date

First pass



At the start of the second pass, the largest value is now in place. There are n-1n-1 items left to sort, meaning that there will be n-2n-2 pairs since each pass places the next largest value in place, the total number of passes necessary will be n-1 n-1 After completing the n-1 n-1 passes, the smallest item must be in the correct position with no further processing required The exchange operation sometimes called a "swap"

Program for Bubble sort.

```
def bubblesort(alist):
    for passnum in range(len(alist)-1, 0, -1):
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp
    alist = [54,26,93,17,77,31,44,55,20]
    bubblesort(alist)
    print(alist)
```

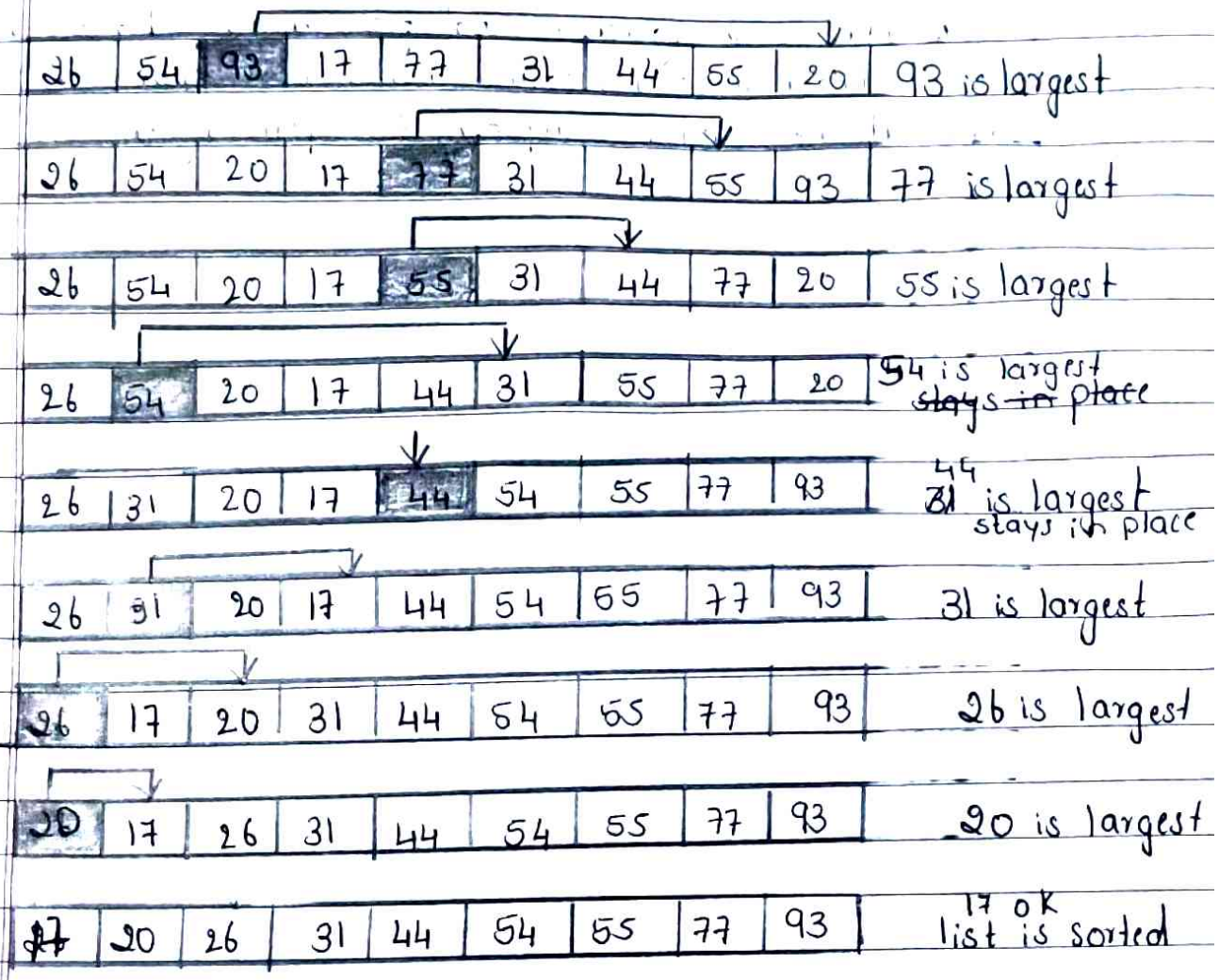
output

```
[17, 20, 26, 31, 44, 54, 55, 77, 93]
```

* The Selection sort

The selection sort improves on the bubble sort by making only one exchange for every pass through the list In order to do this, a Selection sort looks for the largest value as it makes a pass and, after completing the pass, places it in the proper location As With a bubble sort after the first pass, the largest item is in the correct place After the second pass the next largest is in place. This proced continues and requires n"1 n"1 passes to sort n items, since the final item must be in place after the (n"1) (n"1) last pass

Following figure shows the entire sorting process on each pass the largest remaining item is selected & then placed in it's proper location. The first pass is 93 second pass places 77 & 3rd pass 55 and so on



Program on Selection sort

```
def SelectionSort(alist):
    for fillslot in range(len(alist)-1, 0, -1):
        positionofMax = 0
        for location in range(1, fillslot+1):
            if alist[location] > alist[positionofMax]:
                positionofMax = location
        temp = alist[fillslot]
        alist[fillslot] = alist[positionofMax]
        alist[positionofMax] = temp
alist = [54, 26, 93, 17, 77, 31, 44, 55, 20]
SelectionSort(alist)
print(alist)
```

output

[17, 20, 26, 31, 44, 54, 55, 77, 93]

GoodLuck | Page No
Date

* Insertion Sort

Works by taking elements from the list one by one and inserting them in their current position into a new sorted list Insertion sort consists of N-1 passes, Where N is the no of elements to be sorted. The i^{th} pass of insertion sort Will insert the i^{th} element A[i] into it's rightful place Among A[1], A[2] - - A[i-1] After doing this insertion the records occuppying A[1] - - A[i] are in sorted order

Program:-

```
Void insertion_sort(int a[], int n)
{
  int i, j, temp;
  for(i=0; i<n; i++)
  {
    temp = a[i];
    for(j=i; j>0 && a[j-1]>temp; j--)
    {
      a[j] = a[j-1];
    }
    a[j] = temp;
  }
}
```

Passes of Insertion sort

Orginial	20	10	60	40	30	15	Positions Moved
After i=1	10	20	60	40	30	15	1
After i=2	10	20	60	40	30	15	0
After i=3	10	20	40	60	30	15	1
After i=4	10	20	30	40	60	15	2
After i=5	10	15	20	30	40	60	4
Sorted Array	10	15	20	30	40	60	

* Shell sort

Is also called as shellsort or shell's method is an in-place comparison-based sorting algorithm that Generalizes insertion sort by allowing the exchange of items that are far apart, starting With a large gap and progressively reducing it

GoodLuck | Page No.
Date

Eg Program:-

```
Void shellsort(int A[], int N)
{
  int i, j, k, temp;
  for(k=N/2; k>0; k=k/2)
     for(i=k; i<N; i++)
     {
        temp=A[i];
        for(j=i; j>=k && A[j-k]>temp; j=j-k)
     {
       A[j]=A[j-k];
     }
       A[j]=temp;
   }
}
```

Eg:-

consider an unsorted array

0	1	2	3	4	5	6
90	34	87	64	58	54	78

d = n/2 = 7/2 = 3.5 = 3

	90	34	87	64	58	54	78
[0,3]	64	34	87	90	58	54	78
[1,4]	64	34	87	90	58	54	78
[2,5]	64	34	54	90	58	87	78
[3,6]	64	34	54	78	58	87	90

d = 3/2 = 1.5 = 1

[0,1]	34	64	54	78	58	87	90
[1,2]	34	54	64	78	58	87	90
[2,3]	34	54	64	78	58	87	90

[3,4] 34 54 64 58 78 87 90
[4,5] 34 54 64 58 78 87 90
[5,6] 34 54 64 58 78 87 90

d = 1/2 = 0.5 = 0

34 54 64 58 78 87 90

34 54 58 64 78 87 90

34 ↓

Final sorted Array.

* Radix Sort:-
Is a small method used when alphabetizing a large list of names Intuitively one might want to sort numbers on their most significant digit

Algorithm:-
Radix-sort(list, n)

Shif = 1
for loop = 1 to Keysize do

for entry = 1 to n do
bucketnumber = (list[entry].key / shif) mod 10
append (bucket[bucketnumber], list[entry])
list = Combinebuckets()
shift = shift * 10

GoodLuck | Page No.
Date

* Eg:-

Radix sort operates on seven 3-digits no

Input	1st pass	2nd pass	3rd pass
329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

The first column is input The remaining columns show the list after successive sorts on increasingly significant digits position The code for Radix sort Assumes that each element in an Array A of nelements has d digits, Where digit 1 is the lowest-order digit and d is the highest-order digit