Chapter-4
Sorting

Introduction

Sorting is the process of arranging data in some logical order. The order can be either ascending or descending. The main objective of sorting is to enhance searching.

Example: Consider we have to retrieve telephone number of a person name Ram from telephone directory. If the names are stored randomly then it is very time consuming and tedious task to retrieve the desired record, so one solution is to sort the data so that we can directly start our search from the name that starts with letter R. this reduces the number of record to be searched and hence the time to search.

There are two types of sorting

1. Internal sorting

   This method of sorting is applied when the entire collection of data to be sorted is small enough so that sorting can take place within main memory.

   The various internal sort methods are

   i. Insertion sort
   ii. Selection sort
   iii. Bubble sort
   iv. Quick sort
   v. Merge sort
   vi. Radix sort
   vii. Heap sort

2. External sorting

   This method of sorting is applied when the data to be sorted is of large size so that some of the data is present in main Memory and some kept in auxiliary memory.

Elementary Sorting Algorithm

Insertion Sort

Insertion sort is implemented by inserting a particular element at appropriate position. The first iteration starts with comparison of 1st element with 0th element. During second iteration, the 2nd element is compared with 1st and 0th element. So in general, in every iteration an element is compared with all the elements before it. During the comparison, if it is found suitable position to be inserted then space is created by shifting the other elements one position right and inserting the element at the suitable position. This procedure is repeated for all the elements in an array up to (n-1) iteration.

Algorithm

Let A[N] be an array of size N.

```
1. Repeat step 2 through 4 for i varying from 1 to N-1
2. temp = A[i];
   j=i-1
3. while( j>=0 and temp<a[j])
        {
            A[j+1] = a[j];
            j=j-1;
        }
4. A[j+1} =temp
```

C implementation

```
#include<stdio.h>
void insertionSort(int[],int);
void printNumbers(int[],int);
int input[50];
int main()
{
        int n;
        printf("\nEnter the number of elements");
        scanf("%d",&n);
        printf("\nEnter %d elementst to be sorted\n",n);
```

```
        for(int i=0;i<n;i++)
        {
                scanf("%d",&input[i]);
        }
        insertionSort(input,n);

}

void printNumbers(int input[],int n)
{
        for (int i = 0; i <n; i++)
        {
                printf("%d\t",input[i]);
        }
        printf("\n");
}

void insertionSort(int array[],int n)
{
        for (int i = 1; i<n; i++)
        {
                int temp = array[i];
                int j = i-1;
                while ( (j >=0) && ( temp<array[j] ) )
                {
                        array [j+1] = array [j];
                        j--;
                }
                array[j+1] = temp;
                //display each iteration of insertion sort
                printNumbers(array,n);
        }
}
```

Output:

```
Enter the number of elements9

Enter 9 elementst to be sorted
4
2
9
6
23
12
34
0
1
2       4       9       6       23      12      34      0       1
2       4       9       6       23      12      34      0       1
2       4       6       9       23      12      34      0       1
2       4       6       9       23      12      34      0       1
2       4       6       9       12      23      34      0       1
2       4       6       9       12      23      34      0       1
0       2       4       6       9       12      23      34      1
0       1       2       4       6       9       12      23      34
```

Question:

1. Sort the following number using insertion sort.

   25, 17, 31, 13, 2

   Initially,

25	17	31	13	2
0	1	2	3	4

Iteration 1:

Since 17<25, 17 inserted to correct position by shifting other elements one position right

25	17	31	13	2

17	25	31	13	2

Iteration 2:

since 31>25, 31 is inserted at the same position

17	25	31	13	2

Iteration 3:

Since 13<31, 13<25,13<27, so13 inserted to correct position by shifting other elements one position right

17	25	31	13	2

13	17	25	31	2

Iteration 4:

Since 2<31, 2<25, 2<17, 2<13, so 2 inserted to correct position by shifting other element one position right

13	17	25	31	2

2	13	17	25	31

This is the final sorted array.

2. Sort the following data using insertion sort

   70, 80, 30, 10, 20,99

Initially,

70	80	30	10	20	99
0	1	2	3	4	5

Iteration 1:

Since 80>70, so 80 is inserted at the same position.

70	80	30	10	20	99

Iteration 2:

Since 30<80, 30,70, so 30 inserted to correct position by shifting other elements one position right.

70	80	30	10	20	99

30	70	80	10	20	99

Iteration 3:

Since 10<80, 10<70,10<30, so 10 inserted to correct position by shifting other elements one position right

30	70	80	10	20	99

10	30	70	80	20	99

Iteration 4:

Since 20<80, 20<70, 20< 30 but 20>10 so 20 is inserted to correct position by shifting other elements one position right

10	30	70	80	20	99

10	20	30	70	80	99

Iteration 5:
Since 99>80, so 99 is inserted at the same position

10	20	30	70	80	99

This is the final sorted array.

Analysis of insertion sort

To determine the efficiency, we need to find out the number of comparisons made.

i. **Best Case:** If the list is already sorted, we have to make only one comparison in each iteration. So in n-1 iteration, we will have to make n-1 comparisons. Therefore the best case efficiency of insertion sort is O(n).

ii. **Worst Case:** The worst case is when the data in the array is in reversed order. Here one comparison is made in first iteration, two in second iteration and n-1 comparisons in n-1 iteration.

Total number of comparisons = 1 + 2 + 3 + ……………….+ (n-1)

$$= \frac{(n-1) * n}{2}$$

$$= (n^2/2) - (n/2)$$

If n is very large then n^2 >>>>>>n

Therefore Big-Oh notation of insertion sort is $O(n^2)$

iii. **Average Case:** The average case efficiency of insertion sort is $O(N^2)$.

Selection Sort

In selection sort, we repeatedly find the next largest or smallest element in the array and move it to the final position in the sorted array hence it is the combination of both searching and sorting. So in this algorithm, in first iteration, we locate first smallest element from the array and swap it with the element in first position in the array ***i.e. the 0^{th} element is compared with all other elements, if the 0^{th} element is found to be greater than compared element then they are interchanged.*** We then reduce the size of array by one element and proceed to second iteration for processing the smaller sub array. So in second iteration, we locate next smallest element and swap it with the element in the second position in the array. We then reduce the size of array by two elements and proceed to third iteration. We repeat this until the size of sub array is one.

Algorithm

Let array[N] be an array of size N

1. Repeat step 2 for i varying from 0 to N-2
2. Repeat step 3 for j varying from i+1 to N-1
3. If array[i] > array[j] then
   a. temp = array[i]
   b. array[i] = array[j]
   c. array[j] = temp

C-Implementation

```
#include<stdio.h>
void printNumbers(int[], int ) ;
void selectionSort(int[],int);

int input[50];
int main()
{
    int n;
    printf("\nEnter the number of elements");
    scanf("%d",&n);
    printf("\nEnter %d elementst to be sorted\n",n);
    for(int i=0;i<n;i++)
    {
        scanf("%d",&input[i]);
    }
    selectionSort(input,n);
}
```

```
void printNumbers(int input[],int n)
{
	for (int i = 0; i <n; i++)
	{
		printf("%d\t",input[i]);
	}
	printf("\n");
}
void selectionSort(int array[],int n)
{
	for (int i = 0; i <= n - 2; i++)
	{
		for (int j = i + 1; j <=n-1; j++)
		{
			if (array[i] > array[j])
			{
				int temp = array[i];
				array[i] = array[j];
				array[j] = temp;
			}
		}
	printNumbers(array,n);
	}
}
```

Output:

```
Enter the number of elements8

Enter 8 elementst to be sorted
5
1
12
-5
16
2
12
14
-5	5	12	1	16	2	12	14
-5	1	12	5	16	2	12	14
-5	1	2	12	16	5	12	14
-5	1	2	5	16	12	12	14
-5	1	2	5	12	16	12	14
-5	1	2	5	12	12	16	14
-5	1	2	5	12	12	14	16
```

Question:

1. Sort the following using Selection sort

25, 17, 31, 3

Initially,

25	17	31	3
0	1	2	3

Iteration 1:
In first iteration, we locate the first smallest element i.e. 3

25	17	31	3

17	25	31	3

3	25	31	17

3		25	31	17

Iteration 2:
In second iteration, we locate the second smallest element i.e. 27

3		25	31	17

3		17	31	25

3	17		31	25

Iteration 3:
In third iteration, we locate the third smallest element i.e. 25

3	17		31	25

3	17		25	31

3	17	25	31

This is the final sorted array.

2. Sort the following data using Selection Sort
5, 1, 12, -5, 16,2,12,14

Initially:

5	1	12	-5	16	2	12	14
0	1	2	3	4	5	6	7

Iteration 1:
In first iteration, we locate the first smallest element i.e. -5

5	1	12	-5	16	2	12	14

1	5	12	-5	16	2	12	14

-5	5	12	1	16	2	12	14

-5		5	12	1	16	2	12	14

Iteration 2:

In second iteration, we locate the second smallest element i.e. 1

-5		5	12	1	16	2	12	14
-5		1	12	5	16	2	12	14
-5	1		12	5	16	2	12	14

Iteration 3:

In third iteration, we locate the third smallest element i.e. 2

-5	1		12	5	16	2	12	14
-5	1		5	12	16	2	12	14
-5	1		2	12	16	5	12	14
-5	1	2		12	16	5	12	14

Iteration 4:

In fourth iteration, we locate the fourth smallest element i.e. 5

-5	1	2		12	16	5	12	14
-5	1	2		5	16	12	12	14
-5	1	2	5		16	12	12	14

Iteration 5:

In fifth iteration, we locate the fifth smallest element i.e. 12

-5	1	2	5		16	12	12	14
-5	1	2	5		12	16	12	14
-5	1	2	5	12		16	12	14

Iteration 6:

In sixths iteration, we locate the sixth smallest element i.e. 12

-5	1	2	5	12		16	12	14
-5	1	2	5	12		12	16	14
-5	1	2	5	12	12		16	14

Iteration 7:

In seventh iteration, we locate the seventh smallest element i.e. 16

-5	1	2	5	12	12		16	14
-5	1	2	5	12	12		14	16

This

-5	1	2	5	12	12	14	16

This is the final sorted array

Analysis of Selection Sort
There are n-1 comparisons in 1st iteration, n-2 in 2nd iteration, so total number of comparisons =(n-1) +(n-2) +…..+3 + 2 + 1
Therefore the efficiency of selection sort is of order $O(N^2)$

Bubble Sort

In bubble sort, each element is compared with its adjacent element. If the first element is larger than the second element then the position of the element are interchanged otherwise it is not changed. Then the next element is compared with its adjacent element and the same process is repeated for all the elements in the array. So at the end of this, last position contain the biggest element. The next iteration is done up to N-2 element and give second largest element and so on.

Algorithm

Let a[N] be an array of size N.

1. Repeat step 2 for i varying from 0 to N-2.
2. Repeat step 3 for j varying from 0 to N-i-2
3. If(a[j] > a[j+1]) then
   a. temp = a[j]
   b. a[j] = a[j+1]
   c. a[j+1] = temp

C-Implementation

```
#include<stdio.h>
void printNumbers(int[], int ) ;
void bubbleSort(int[],int);
int input[50];
int main()
{
        int n;
        printf("\nEnter the number of elements");
        scanf("%d",&n);
        printf("\nEnter %d elementst to be sorted\n",n);
        for(int i=0;i<n;i++)
        {
                scanf("%d",&input[i]);
        }
        bubbleSort(input,n);
}

void printNumbers(int input[],int n)
{
        for (int i = 0; i <n; i++)
        {
                printf("%d\t",input[i]);
        }
        printf("\n");
}
```

```
void bubbleSort(int array[],int n)
{
        for (int i = 0; i <= n - 2; i++)
        {
                for (int j = 0; j <=n-i-2; j++)
                {
                        if (array[j] > array[j+1])
                        {
                                int temp = array[j];
                                array[j] = array[j+1];
                                array[j+1] = temp;
                        }
                }
                printNumbers(array,n);
        }
}
```

Output:

```
Enter the number of elements5

Enter 5 elementst to be sorted
25
17
31
13
2
17      25      13      2       31
17      13      2       25      31
13      2       17      25      31
2       13      17      25      31
```

Question:

1. Sort the following numbers using bubble sort

25, 17, 31, 13, 2

Initially:

25	17	31	13	2
0	1	2	3	4

Iteration 1:

17<25, so interchchange.

25	17	31	13	2

17	25	31	13	2

13<31, so interchange

17	25	31	13	2

17	25	13	31	2

2<31, so interchange

17	25	13	31	2

17	25	13	2	31

17	25	13	2	31

Iteration 2:
13<25 so interchange

17	25	13	2	31
17	13	25	2	31
17	13	25	2	31

2<25, so interchange

17	13	2	25	31
17	13	2	25	31

Iteration 3:
13<17, so interchange

17	13	2	25	31
13	17	2	25	31

2<17, so interchange

13	17	2	25	31
13	2	17	25	31

Iteration 4:
2<13, so interchange

13	2			
2	13	17	25	31
2	13	17	25	31

This is the final sorted array.

2. Sort the following number using bubble sort.

5, 1, 12, -5, 16,2,12,14

Initially:

5	1	12	-5	16	2	12	14
0	1	2	3	4	5	6	7

Iteration 1:
1<5, so interchange

5	1	12	-5	16	2	12	14
1	5	12	-5	16	2	12	14

-5<12, so interchange

1	5	12	-5	16	2	12	14
1	5	-5	12	16	2	12	14

2<16, so interchange

1	5	-5	12	16	2	12	14
1	5	-5	12	2	16	12	14

12<16, so interchange

1	5	-5	12	2	16	12	14
1	5	-5	12	2	12	16	14

14<16, so interchange

1 | 5 | -5 | 12 | 2 | 12 | 16 | 14

1 | 5 | -5 | 12 | 2 | 12 | 14 | 16

1 | 5 | -5 | 12 | 2 | 12 | 14 || 16

Iteration 2:
-5<5 so interchange

1 | 5 | -5 | 12 | 2 | 12 | 14 || 16

1 | -5 | 5 | 12 | 2 | 12 | 14 || 16

2<12 so interchange

1 | -5 | 5 | 12 | 2 | 12 | 14 || 16

1 | -5 | 5 | 2 | 12 | 12 | 14 || 16

1 | -5 | 5 | 2 | 12 | 12 || 14 | 16

Iteration 3:
-5<1, so interchange

1 | -5 | 5 | 2 | 12 | 12 || 14 | 16

-5 | 1 | 5 | 2 | 12 | 12 || 14 | 16

2<5, so interchange

-5 | 1 | 5 | 2 | 12 | 12 || 14 | 16

-5 | 1 | 2 | 5 | 12 | 12 || 14 | 16

-5 | 1 | 2 | 5 | 12 || 12 | 14 | 16

Iteration 4: -5 | 1 | 2 | 5 || 12 | 12 | 14 | 16

Iteration 5: -5 | 1 | 2 || 5 | 12 | 12 | 14 | 16

Iteration 6: -5 | 1 || 2 | 5 | 12 | 12 | 14 | 16

Iteration 7: -5 || 1 | 2 | 5 | 12 | 12 | 14 | 16

-5 | 1 | 2 | 5 | 12 | 12 | 14 | 16

This is the final sorted array.

Analysis of Bubble Sort

There are N-1 comparisons in 1st iteration, N-2 comparisons in 2nd iteration and so. Therefore total number of comparisons = (n-1) + (n-2) +….+ 3 + 2 + 1. Hence the efficiency of bubble sort algorithm is of order $O(N^2)$ i.e. time taken to execute increases quadratically with increase in input data.

Efficient Sorting Algorithms

Quick Sort / Partition exchange sort

Quick sort is based on divide and conquer approach. That means divide the big problem into two small problems and then those two small problems into two small problems and so on. The quick sort algorithm recursively divides the list into two subsets. It is inefficient for small array.

Algorithm

1. Let A[N] be an array of size N. Select an element from the list as key.

2. Now divide the list into two sub lists, such that left list is less than or equal to key value and right list is greater than key value. For doing this, we take two pointer variables, one called **up** and another called **down.**
3. The two pointer up and down are moved toward each other in the following ways
    i. Repeatedly increase the down pointer by one position until A[down] > key or down = N-1
    ii. Repeatedly decrease the up pointer by one position until A[up] <=key.
    iii. If up > down, interchange A[down] with A[up]
4. Repeat step 3 until the up and down pointer crossover each other i.e. (down >=up)
5. After the crossover takes place, exchange the up value with key value.
6. Then there will be two lists of elements i.e. left sub list and right sub list
7. Now we repeat the same steps for both sides till each list contains only one element.

Question:

1. Sort the following numbers using quick sort.

24,2,45,20,56,75,2,56,99,53,12

Initially:

24	2	45	20	56	75	2	56	99	53	12
0	1	2	3	4	5	6	7	8	9	10

Let key value = 24

24	2	45	20	56	75	2	56	99	53	12
		↑ Down								↑ Up

Since Down < Up, interchanging Up and down values

24	2	12	20	56	75	2	56	99	53	45
				↑ Down		↑ Up				

Since Down < Up, interchanging Up and down values

24	2	12	20	2	75	56	56	99	53	45
				↑ Up	↑ Down					

Since Down>=Up i.e. crossover of Down and Up pointer so interchanging Up and key values. This leads to two lists, left list contains all the elements less than or equal 24 and right list contains all the elements greater than 24.

2	2	12	20	24	75	56	56	99	53	45

2	2	12	20

24

75	56	56	99	53	45

List 1List 2

Now sorting List1

Initially:

2	2	12	20

0 1 2 3

Let key value = **2**

2	2	12	20
	↑	↑	

Up Down

Since Down>=Up i.e. crossover of Down and Up pointer so interchanging Up and key values. This leads to two lists, left list contains all the elements less than or equal to 2 and right list contains all the elements greater than 2.

2	2	12	20

2

2

12	20

List 3List 4

Now sorting List 4
Initially:

12	20
2	3

Let key = 12

12	20
Up	Down

Since Down>=Up i.e. crossover of Down and Up pointer so interchanging Up and key values. This leads to two lists, left list contains all the elements less than or equal 12 and right list contains all the elements greater than12

12		20

List 5

Now sorting List 2
Initially:

75	56	56	99	53	45
5	6	7	8	9	10

Let key = 75

75	56	56	99	53	45
			Down		UP

Since Down < Up, interchanging Up and down values

75	56	56	45	53	99
				Up	Down

Since Down>=Up i.e. crossover of Down and Up pointer so interchanging Up and key values. This leads to two lists, left list contains all the elements less than or equal 75 and right list contains all the elements greater than75

53	56	56	45	75	99

53	56	56	45		75		99

List 6 **List 7**

Now sorting List6
Initially:

53	56	56	45
5	**6**	**7**	**8**

Let key = 53

53	56	56	45
	Down		Up

Since Down<Up, interchanging Up an

53	45	56	56

Up Down

Since Down>=Up i.e. crossover of Down and Up pointer so interchanging Up and key values. This leads to two lists, left list contains all the elements less than or equal 53 and right list contains all the elements greater than53

45	53	56	56

45	53	56	56

List8 **List9**

Now sorting List 9

Initially:

56	56
7	8

Let key = 56

56	56

UP Down

Since Down>=Up i.e. crossover of Down and Up pointer so interchanging Up and key values. This leads to two lists, left list contains all the elements less than or equal 56 and right list contains all the elements greater than56

56	56

List10

So our final sorted data is

2	2	12	20	24	45	53	56	56	75	99

2. Sort the following numbers using Quick Sort

25, 37, 17, 81, 43, 32, 19, 9, 23

Initially:

25	**37**	**17**	**81**	**43**	**32**	**19**	**9**	**23**
0	1	2	3	4	5	6	7	8

Let key = 25

25	**37**	**17**	**81**	**43**	**32**	**19**	**9**	**23**

Down Up

Since Down < Up, interchanging Up and Down values.

25	**23**	**17**	**81**	**43**	**32**	**19**	**9**	**37**

Down Up

Since Down < Up, interchanging Up and Down values

25	**23**	**17**	**9**	**43**	**32**	**19**	**81**	**37**

Down Up

Since Down < Up, interchanging Up and Down values

25	**23**	**17**	**9**	**19**	**32**	**43**	**81**	**37**

Up Down

Since Down>=Up i.e. crossover of Down and Up pointer so interchanging Up and key values. This leads to two lists, left list contains all the elements less than or equal 25 and right list contains all the elements greater than25.

19	**23**	**17**	**9**	**25**	**32**	**43**	**81**	**37**

19	23	17	9	25	32	43	81	37

List1 **List2**

Now sorting List1

Initially:

19	23	17	9
0	1	2	3

Let key = 19

19	23	17	9

Down Up

Since Down < Up, interchanging Up and Down values

19	9	17	23

Up Down

Since Down>=Up i.e. crossover of Down and Up pointer so interchanging Up and key values. This leads to two lists, left list contains all the elements less than or equal 19 and right list contains all the elements greater than 19.

17	9	19	23

17	9	19	23

List3 **List4**

Now sorting List3

Initially:

17	9
0	1

Let key = 17

17	9

Up Down

Since Down>=Up i.e. crossover of Down and Up pointer so interchanging Up and key values. This leads to two lists, left list contains all the elements less than or equal 17 and right list contains all the elements greater than 17.

9	17

9	17

List5

Now sorting List2

Initially:

32	43	81	37
5	6	7	8

Let key = 32

32	43	81	37

↑ ↑

Up Down

Since Down>=Up i.e. crossover of Down and Up pointer so interchanging Up and key values. This leads to two lists, left list contains all the elements less than or equal 32 and right list contains all the elements greater than 32.

32		43	81	37

List6

Now sorting List6

Initially:

43	81	37
6	7	8

Let key = 43

43	81	37

↑ ↑

Down Up

Since Down < Up, interchanging Up and Down values

43	37	81

↑ ↑

Up Down

Since Down>=Up i.e. crossover of Down and Up pointer so interchanging Up and key values. This leads to two lists, left list contains all the elements less than or equal 43 and right list contains all the elements greater than 43.

37	43	81

37	43	81

List 7 **List8**

So our sorted data is

9	**17**	**19**	**23**	**25**	**32**	**37**	**43**	**81**

Analysis of Quick sort

Worst Case: The worst case occur when the list is already sorted because one of the two subset will be always empty and the other will contain all the elements. So 1st element require n-1 comparisons to recognize that it remains in first position, 2nd element will require n-2 comparisons to recognize that it remain in 2nd position. Similarly last element will require 1 comparison. Therefore total number of comparisons made =(n-1) + (n-2) + (n-3) +………+ 2 + 1

= ((n-1) *n)/2

Hence the worst case efficiency of bubble sort algorithm is of order $O(N^2)$ i.e. time taken to execute increases quadratically with increase in input data.

Best Case: The best case occur when we select the median (middle) of all the value as key. So partitioning of initial list places one element at its correct position and produce two sub lists of nearly equal size. Similarly partitioning of two sub list places two element at their correct position and produces four sub lists of nearly equal size i.e. 3 elements are placed in correct position. This process continues until all elements are placed in their correct position. Generalizing the process, we can say k^{th} make 2^k-1 element at their correct position i.e. if K^{th} partition is require to sort all the elements then all the n elements are placed in their correct position.

So $2^k - 1 = n$

$2^k = n + 1$

$k = \log_2(n+1)$ // if $a^b = c$ then $b = \log_a(c)$

$\approx \log(n)$

Hence there is maximum of log(n) reductions to sort n element and for each reduction, there are maximum of n comparisons. Hence best case efficiency of quick sort is of order O(nlogn).

Average Case: The average case efficiency of quick sort is O(nlogn).

Merge Sort

Merging means combining two or more sorted lists into a third sorted list. It is also based on divide and conquers approach strategy of solving problem. In this algorithm, the list to be sorted is divided into two sub lists of two nearly equal sizes then two sub list are sorted separately by using merge sort. The two sorted sub lists are then merged into a single sorted list.

Algorithm:

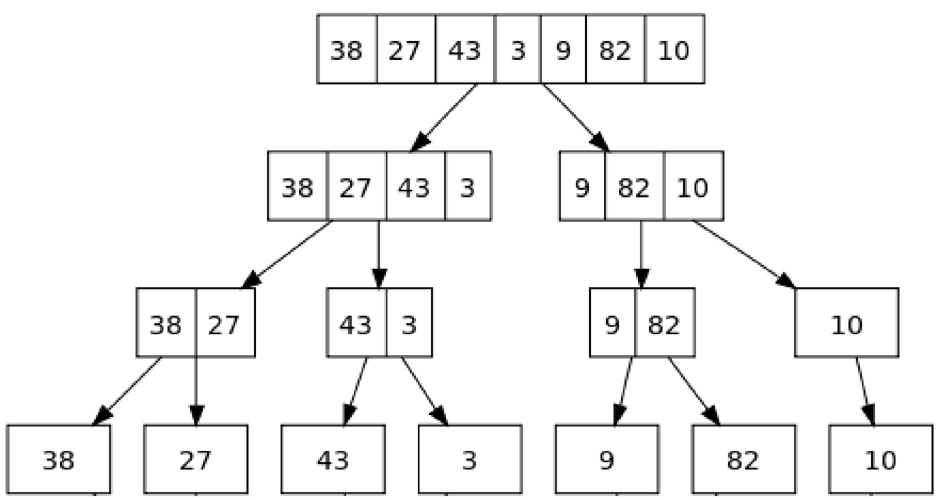
1. If then number of item to sort is 0 or 1, return.
2. Divide the list into two sub lists until each list contain only 1 element.
3. Recursively merge sort the first and second list separately.
4. Merge the two sorted lists into a single sorted list.

Question:

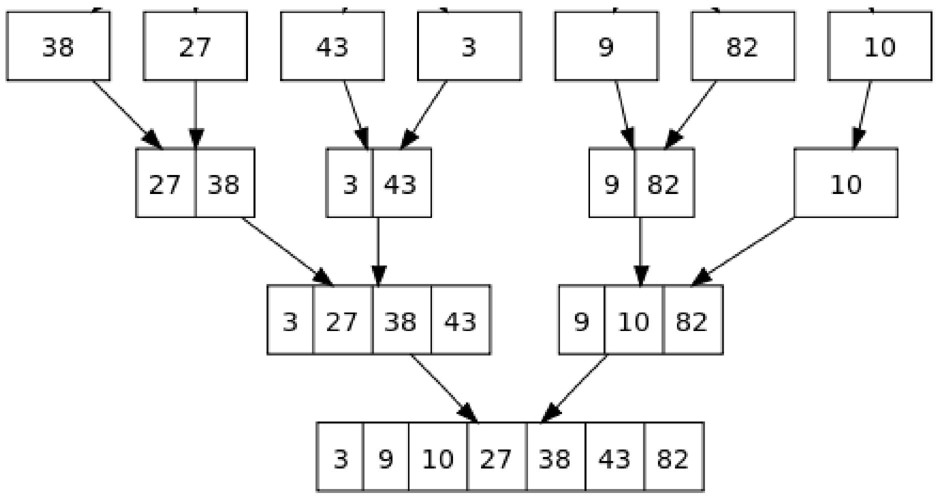
1. Sort the following numbers using merge sort

38, 23, 43, 3, 9, 82, 10

Partitioning the list into two nearly equal sub lists until the list size is 1 as below



Now merging the sorted sub list, we get

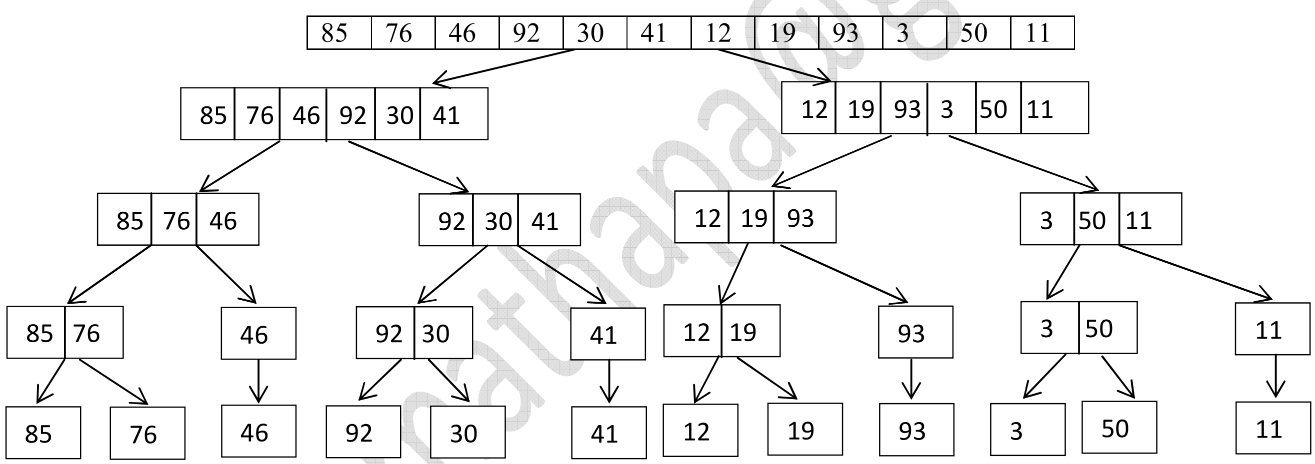


This is the final sorted array.

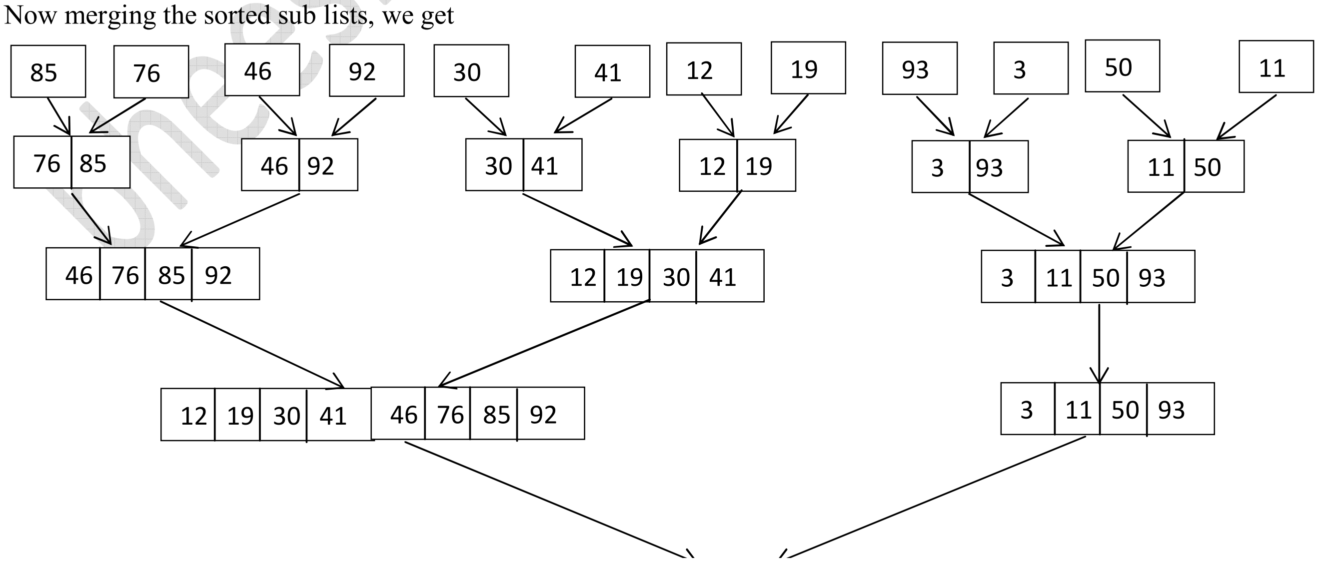
2. Sort the following numbers using merge sort.

85, 76, 46, 92, 30, 41, 12, 19, 93, 3, 50, 11

Partitioning the list into two nearly equal sub lists until the list size is 1 as below'



Now merging the sorted sub lists, we get



This is the final sorted array.

Analysis of Merge Sort

Let us consider the list of size N. so the sort the list, we need to recursively divide the list into two nearly equal sub list until each sub list consists of only one element. To divide the list into sub list of size one, it require logN passes. Again in each pass, a maximum of N comparisons are performed. Therefore the efficiency of merge sort is equal toO(NlogN)

Radix Sort / Bucket Sort

Radix sort is a method that can be used to sort numbers as well as list of strings alphabetically. For sorting numbers, the base or radix is 10, so we need 10 buckets but for sorting strings, the base or radix is 26, so we need 26 buckets. It is fastest sorting method for sorting random fixed length strings but doesn't work for variable length strings.

Algorithm for sorting Numbers

Let A[N] be an array of size N

1. In the first pass, the unit digits are sorted into the buckets.
2. In the second pass, the tens digits are sorted into the bucket
3. In the third pass, the hundredth digits are sorted into the bucket.
4. Continue the pass up to the length of the digit with maximum length.
5. Finally collect all the numbers from the bucket in order.

Algorithm for sorting Strings

Let A[N] be an array of size N

1. In the first pass, the list of strings is sorted according to the last letter of each string.
2. In the second pass, the list of strings is sorted according to the second last letter of each string.
3. In the third pass, the list of string is sorted according to the third last letter of each string.
4. Continue the pass up to the length of the string with maximum length.
5. Finally collect all the strings from the bucket in order.

Question:

1. Sort the following numbers using radix sort.

499, 283, 491, 353, 726, 315, 124, 278, 329

Solution:

For sorting numbers, the radix is 10 so we need 10 buckets numbered 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

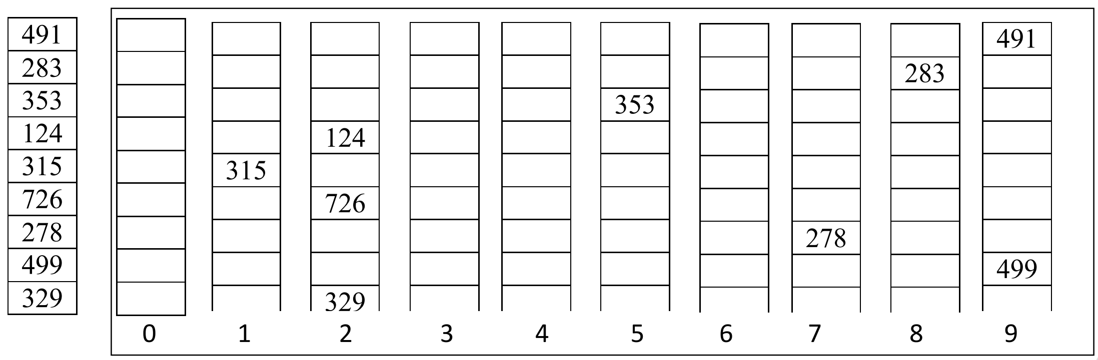
Pass 1:

Input | Buckets

Input	0	1	2	3	4	5	6	7	8	9
499										499
283				283						
491		491								
353				353						
726							726			
315						315				
124					124					
278									278	
329										329

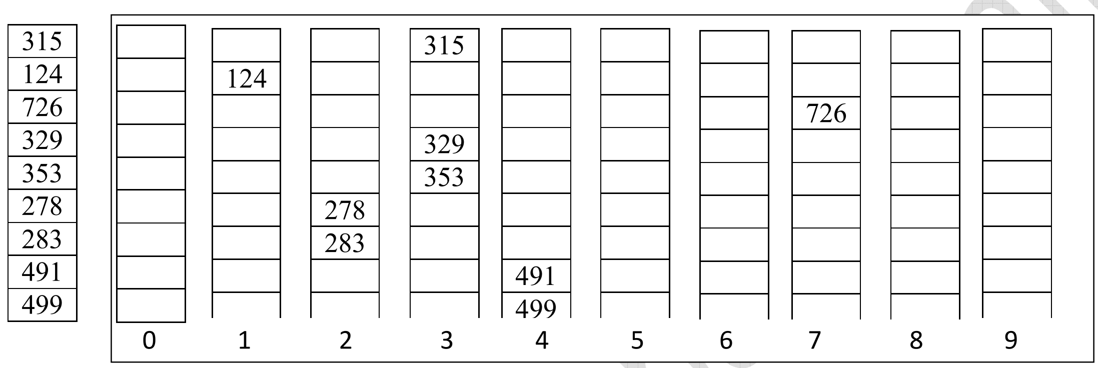
Pass 2:

Input Buckets



Pass 3:

Input Buckets



Collecting all the numbers from the bucket in order we get the following sorted numbers

124, 278, 283, 315, 329, 353, 491, 499, 726

2. Sort the following numbers using radix sort

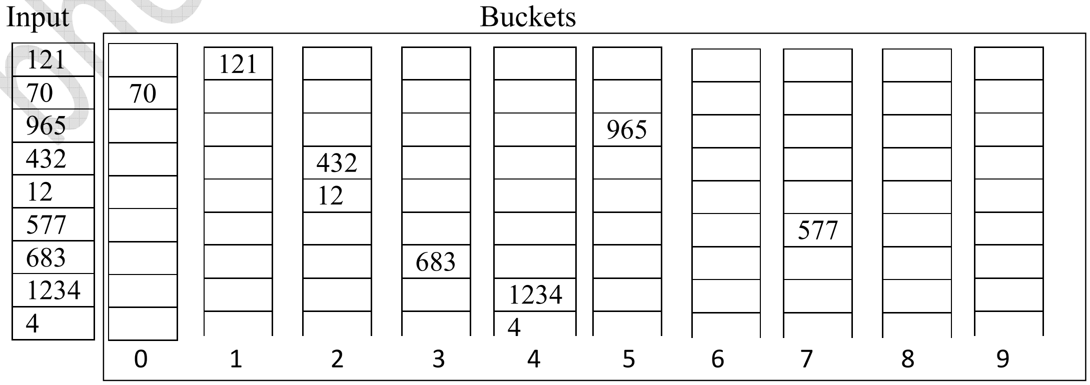
121, 70, 965, 432, 12, 577, 683,1234,4

Solution:

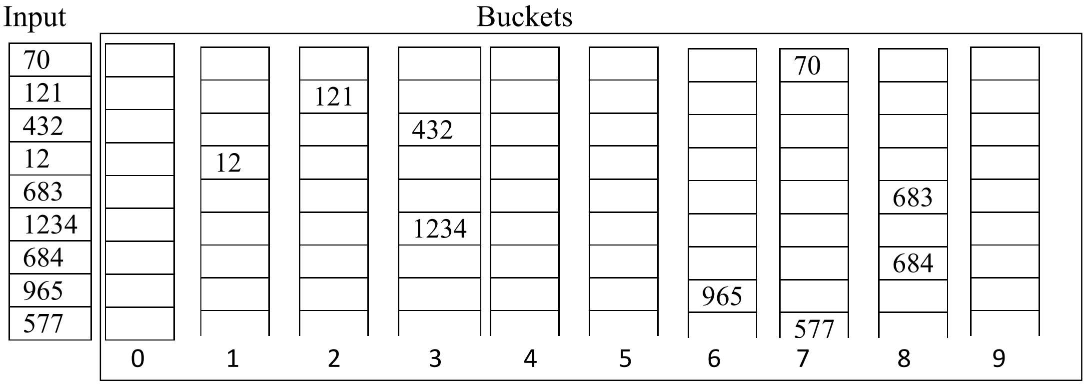
For sorting numbers, the radix is 10 so we need 10 buckets numbered 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Pass 1:

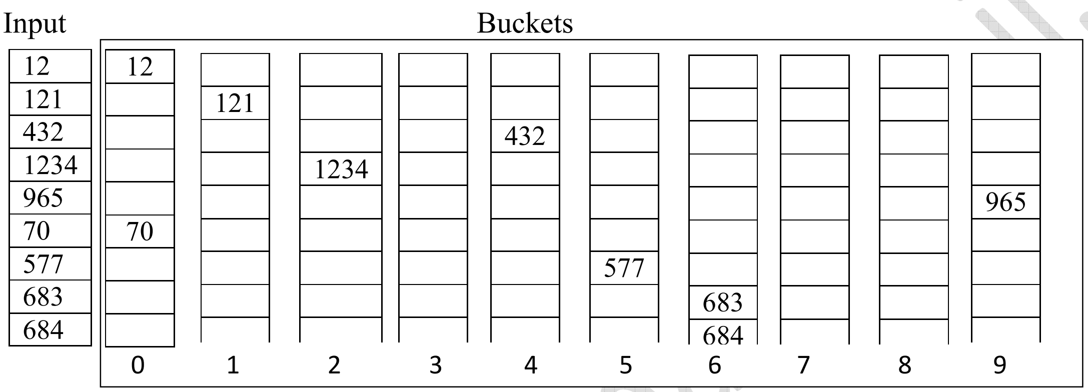
Input Buckets



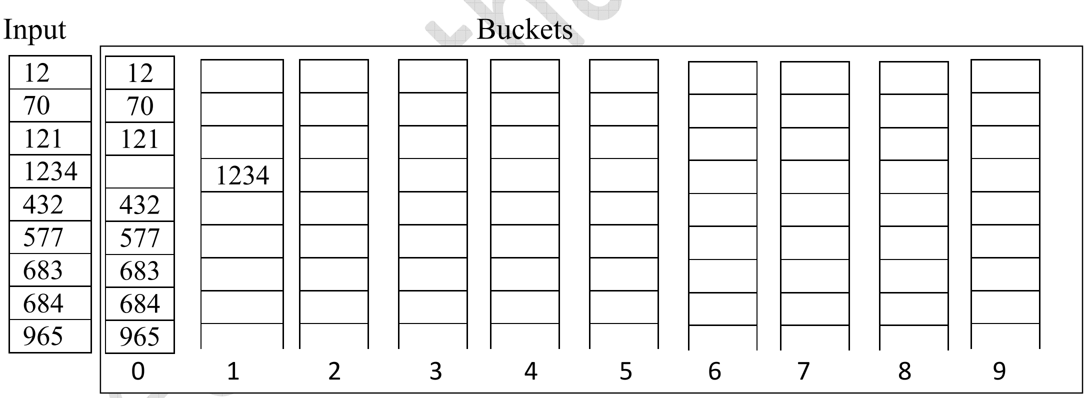
Pass 2:



Pass 3:



Pass 4:



Collecting all the numbers from the bucket in order we get the following sorted numbers

12, 70, 121, 432, 577, 683, 684, 965, 1234

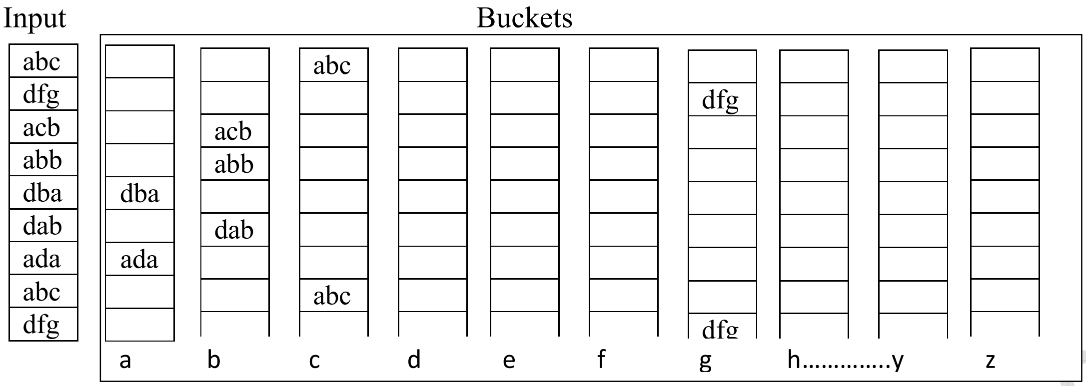
3. Sort the following word in ascending order.

abc, dfg, acb,abb, dba, dab, ada, abc, dfg

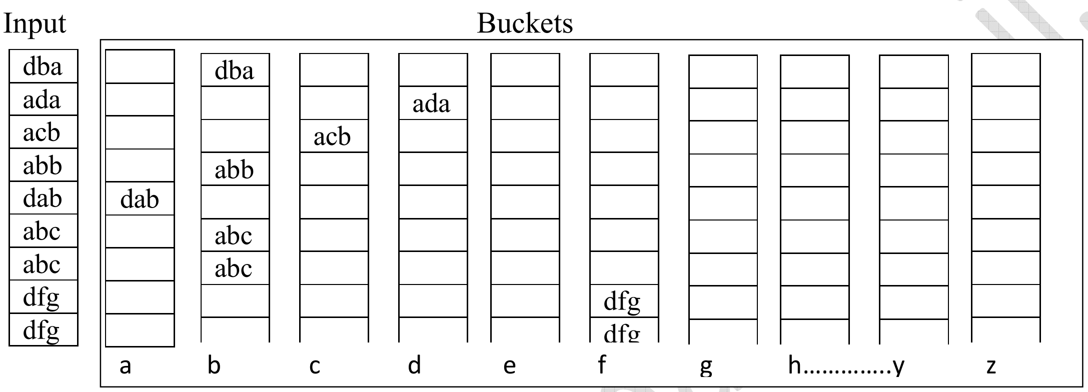
Solution:

For sorting strings, the radix is 26 so we need 26 buckets named a, b, c…..x, y, z

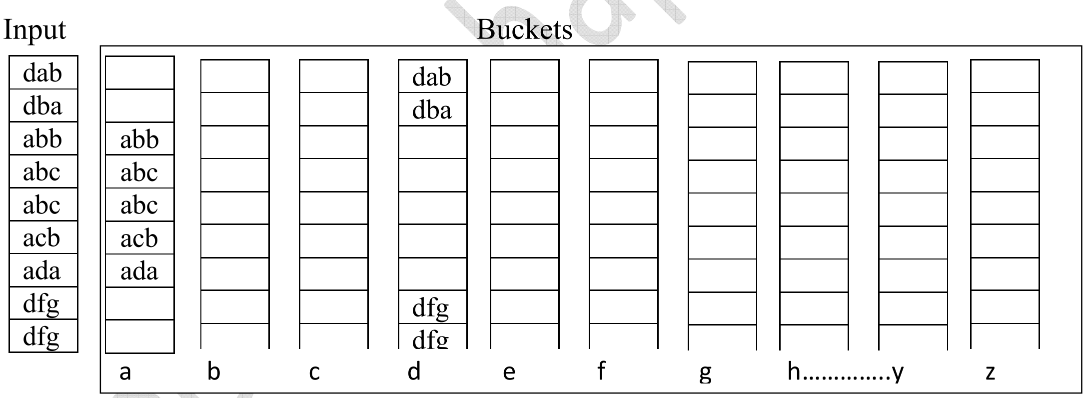
Pass 1:



Pass 2:



Pass 3:



Collecting all the strings from the bucket in order we get the following sorted strings

abb, abc, abc, acb, ada, dab, dba, dfg, dfg

<u>Analysis of Radix Sort</u>

Best Case: O(NlogN)

Average Case: $O(NLogN)^2$

Worst Case: $O(N^{3/2})$

Heap / Heap as priority queue

A heap is defined as an almost complete binary tree of node n such that the value of each node is less than or equal to the value of the father or greater than or equal to the value of father.

Types of Heap

1. Descending Heap(Max Heap)
   The type of heap in which the root of the binary tree has the largest element of the heap i.e. the value of each node is less than or equal to the value of its father.
2. Ascending Heap(Min Heap)
   The type of heap in which in which the root of the binary tree has the smallest element of the heap i.e. the value of each node is greater than or equal to the value of its father.

A heap is very useful for implementing priority queue. In priority queue, items can be inserted in any order but either smallest or largest element is deleted first i.e. in ascending priority queue, smallest item is deleted first. While in descending priority queue, largest item is deleted first. So by creating descending heap, the root consists of largest element and hence used for implementing Descending priority queue and similarly by creating ascending heap, the root consists of smallest element and hence used for implementing ascending priority queue.

Heap Sorting

A heap can be used to sort a set of elements. We can use heap as a descending priority queue or ascending priority queue to sort the given set of elements in ascending order or descending order.

Algorithm (Using Max Heap)

Let A[N] be an array of N elements which is used to represent a Max heap.

1. Construct Max heap to obtain the largest element at the root i.e. A[0].

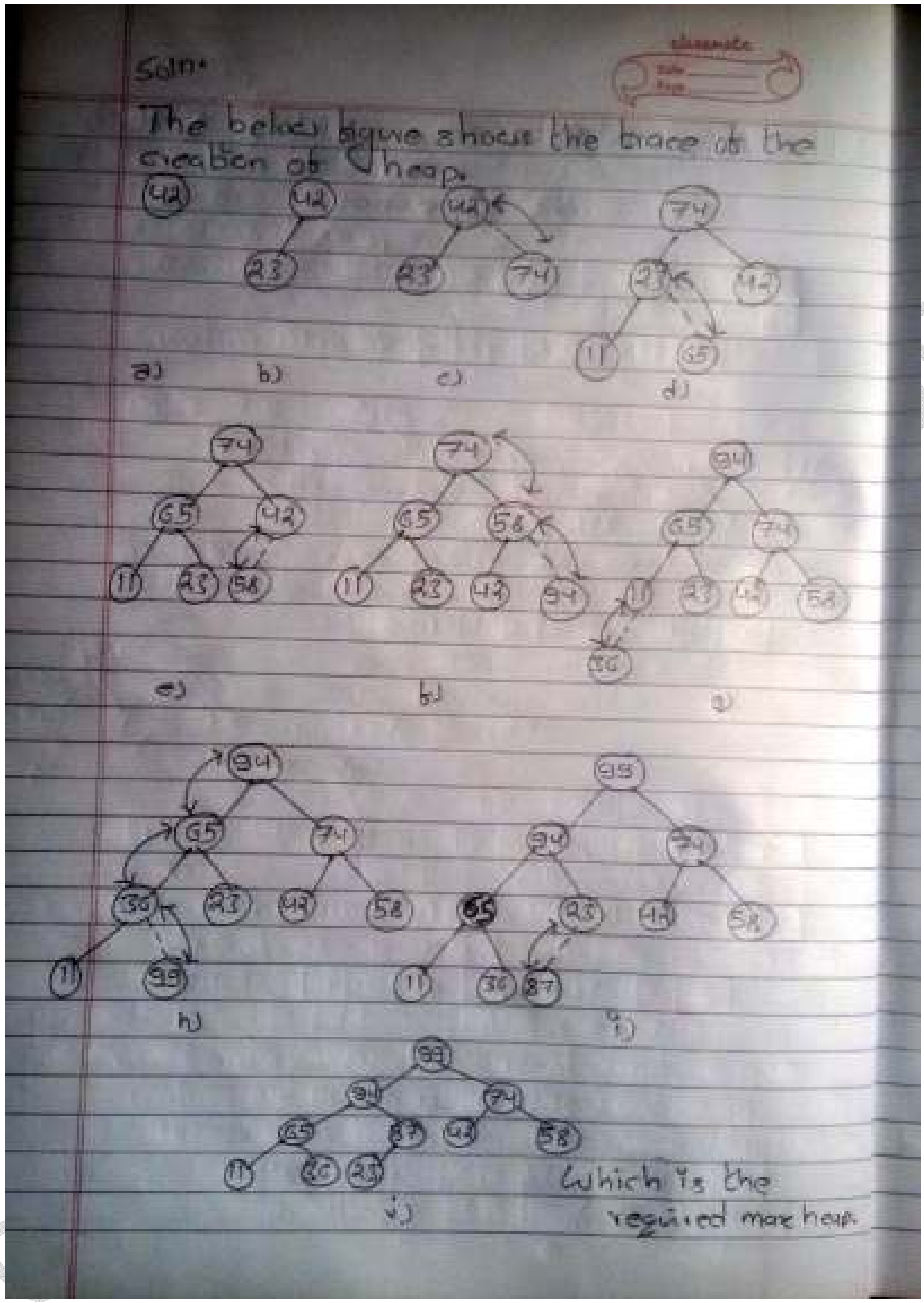
   Algorithm

   i. Put the element at missing leaf i.e. first to left and then to right.
   ii. Switch with its parent i.e. bubble up if its parent is smaller.
   iii. Repeat step i and ii until all the elements are inserted into the heap.
2. Put the largest element into its correct final position by exchanging it with A[N-1] i.e. last element in A.
3. Now discard A[N-1] element and repeat step 1 and 2 until all the elements are sorted in ascending order.

Question:

1. Construct heap (Max/Min) for the following set of data

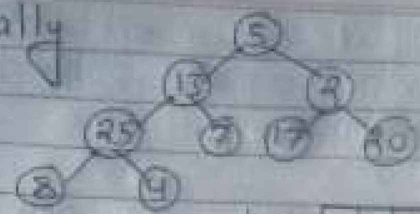
42, 23, 74, 11, 65, 58, 94, 36, 99, 87



2. Illustrate the operation of heap sort on the array

A={5, 13, 2, 25, 7, 17, 20, 8, 4, }

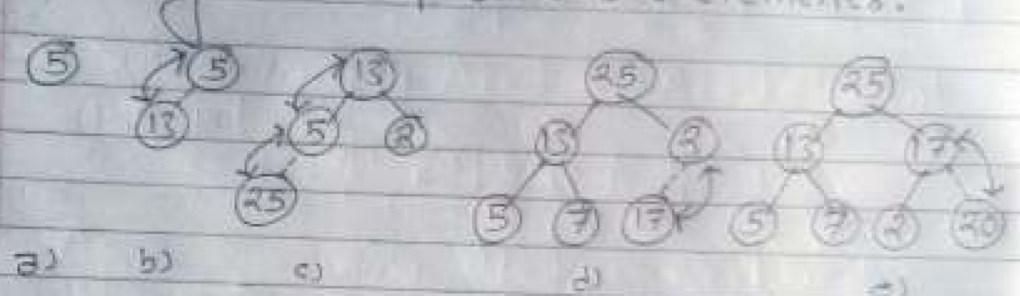
Soln: Originally

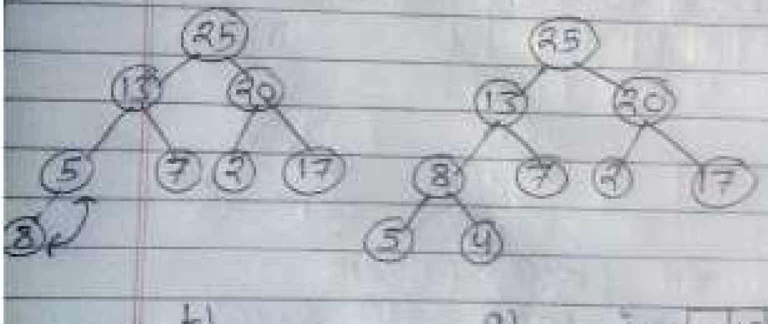


i.e

5	13	2	25	7	17	20	8	4	[redacted]
0	1	2	3	4	5	6	7	8	

Creating max heap for entire elements.





i.e

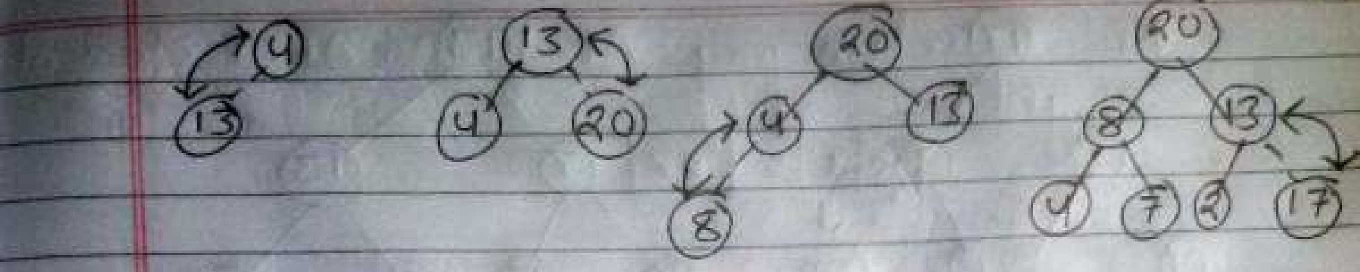
25	13	20	8	7	2	17	5	4
0	1	2	3	4	5	6	7	8

Now exchanging A[0] element with A[8] element we get

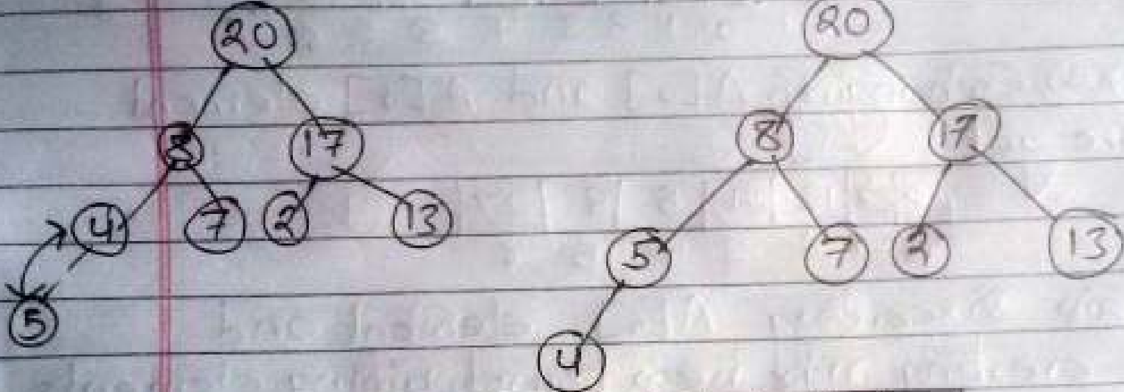
4	13	20	8	7	2	17	5	25
0	1	2	3	4	5	6	7	8

Discarding A[8] element and again creating max heap for the remaining elements as below.

4	13	20	8	7	2	17	5	[redacted]	25
0	1	2	3	4	5	6	7		8



a) b) c) d)



i.e

20	8	17	5	7	2	13	4
0	1	2	3	4	5	6	7

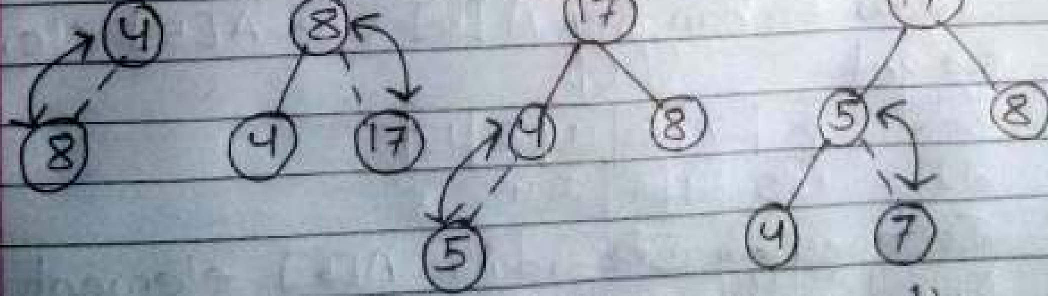
Now Exchanging A[0] with A[7] we get

4	8	17	5	7	2	13	20
0	1	2	3	4	5	6	7

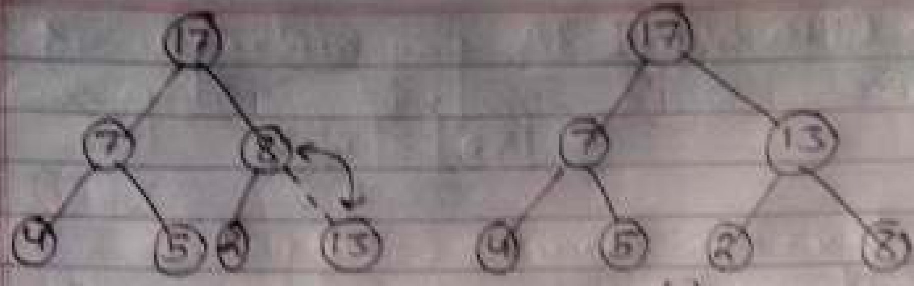
Again Discarding A[7] element and creating max heap for remaining elements.

4	8	17	5	7	2	13

20	25



a) b) c) d)



a) b)

i.e

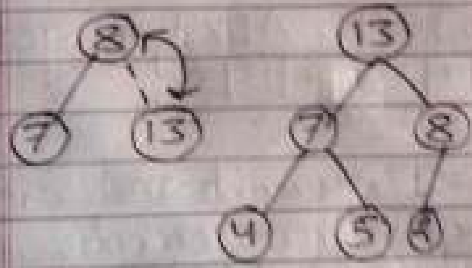
17	7	13	4	5	2	8
0	1	2	3	4	5	6

Now Exchanging A[0] and A[6] element we get

8	7	13	4	5	2	17
0	1	2	3	4	5	6

Again Discarding A[6] element and creating max heap for remaining elements

8	7	13	4	5	2	17	20	25
0	1	2	3	4	5	6	7	8



a) b)

i.e

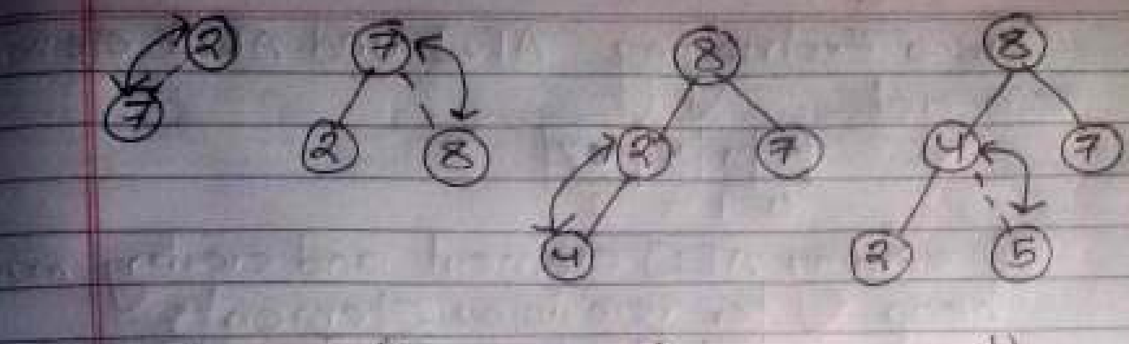
13	7	8	4	5	2
0	1	2	3	4	5

Now Exchanging A[0] and A[5] element we get

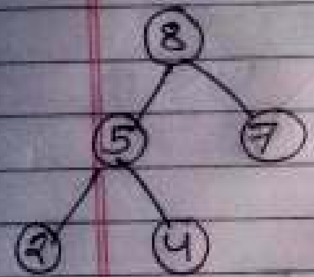
2	7	8	4	5	13
0	1	2	3	4	5

~~Exchanging~~ Discarding A[5] element and creating max heap for remaining elements

2	7	8	4	5	[redacted]	13	17	20	25
0	1	2	3	4		5	6	7	8



a) b) c) d)



i.e

8	5	7	2	4
0	1	2	3	4

e)

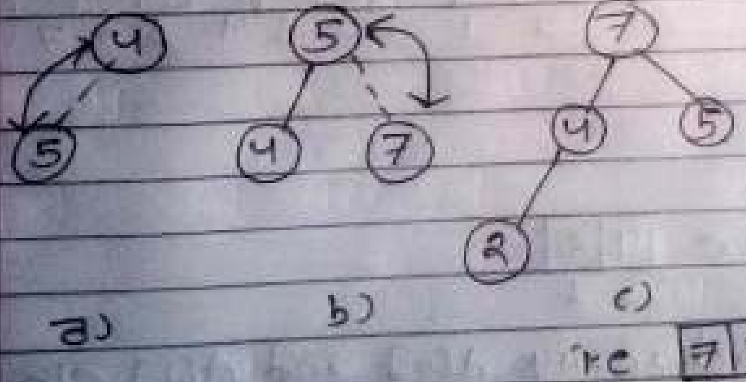
Now Exchanging A[0] and A[4] element we get

4	5	7	2	8
0	1	2	3	4

Discarding A[4] element and creating more heap for remaining elements.

4	5	7	2
0	1	2	3

8	13	17	24	25
4	5	6	7	8



a) b) c)

i.e

7	4	5	2
0	1	2	3

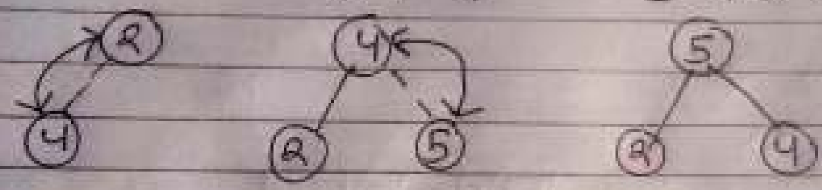
Now Exchanging A[0] and A[3] element we get

2	4	5	7
0	1	2	3

Discarding A[3] element and creating max heap for remaining elements

2	4	5
0	1	2

7	8	13	17	20	25
3	4	5	6	7	8



a) b) c)

i.e

5	2	4
0	1	2

Now Exchanging A[0] and A[2] element we get

4	2	5
0	1	2

Discarding A[2] element and creating max heap for remaining elements

4	2
0	1

5	7	8	13	17	20	25
2	3	4	5	6	7	8

a) i.e

4	2
0	1

Now Exchanging A[0] and A[1] element we get

2	4
0	1

Discarding A[1] element and creating max heap for remaining elements

2
0

4	5	7	8	13	17	20	25
1	2	3	4	5	6	7	8

(2)

i.e

2
0

Therefore the final sorted data is

2	4	5	7	8	13	17	20	25
0	1	2	3	4	5	6	7	8

#

Analysis of Heap Sort

Time complexist= O(NlogN)

Shell Sort / Diminishing increment Sort

See class notes