Sorting And Searching

* Sorting:

Sorting is the processing of arranging the data in ascending and descending order.

There are two type of Sorting Maner.

1) Ascending Order
2) Descending Order

1 Ascending Order:

Ascending Order is the arrangement of numbers from the smallest to the largest.

2 Descending Order:

Descending Order is the arrangement of numbers from the largest to the smallest.

SMVS
Page No.
Date : / /

* Insertion Sort :

Insertion Sort is a simple sorting algorithms with simple implementation.

Insertion sort is efficient for small data values.

Insertion sort also give a good performance when dealing with a small list.

⇒ Example :

12	31	25	8	32	17

In this short, we take two first element and compare with each other.

First we take 12 and 31 number and 12 < 31 so, 12 is the less then 31.

After that take 31 and 25 number. 25 is smaller than 31, So, 31 is shift to 25 number place.

12	25	31	8	32	17

Again, we compare 31 and 8.
8 is smaller than 31 so,
31 is shift to 8 number place.

12	25	8	31	32	17

Here, 8 is smaller than 25 so,
We can interchange 8 and
25 position.

12	8	25	31	32	17

Againe, 8 is smaller than 12 so,
We can interchange 8 and 12
position.

8	12	25	31	32	17

Next, we compare 31 and 32 but
31 is smaller than 32 and 31
is right position.

then, we compare 32 and 17.
17 is smaller than 32 so, we
interchange both position.

8	12	25	31	17	32

Here, 17 is not porper position.
because 31 is larger than 17.

| 8 | 12 | 25 | 17 | 31 | 32 |

Again, 17 is not proper position because 17 is smaller than 25.

| 8 | 12 | 17 | 25 | 31 | 32 |

Here, all the element are proper position.

Hence, Insertion Short is complete.

=> Algorithm:

1 Start

2 In the list, assume that this element is already sorted.

3 After that pick second element and compare.

4 If element is ~~smaller~~ larger than sorted element than interchange the position,

else, move to the next right element

5 Insert the value.

6 Repeat the following array, until the array is shorted.

7 End.

=> Advantages of Insertion Short.

1 Simple Algorithm

2 Efficient for small data sets.

3 This short working is Simple.

=> Disadvantages of Insertion Short.

1 Efficient only ~~large~~ & small data sets.

2 Less efficient than other shorting method.

=> Complexity

1 Time Complexity

Best case - O(n)
Averge Case - $O(n^2)$
Worst Case - $O(n^2)$

2 Time Complexity - O(1)

* Bubble Short:

Bubble sh sort is the simple algorithm that work by repeatedly swap the element.

Bubble sort is not suitable For large element list.

In Bubble sort there are number of element So, Here we can give (n-1) number element to key.

(n-1) number element is deciesed to use the number of key

=> Example: 45, -40, 190, 99, 11

Here, number of element is 5
∴ number of key = 4

(1) K = 1

45, -40, 190, 99, 11

45 > -40 -> Swap

∴ -40, 45, 190, 99, 11

45 < 190

∴ -40, 45, 190, 99, 11

190 > 99 - swap

∴ -40, 45, 99, 190, 11

190 > 11 - swap

∴ -40, 45, 99, ~~190~~ 11, 190

(2) K = 2

∴ -40, 45, 99, 11, 190

∴ -40, 45, 99, 11, 190

∴ -40, 45, 99, 11, 190

99 > 11 - swap

∴ -40, 45, ~~11~~, 99, 190

∴ -40, 45, 11, 99, 190

(3) K = 3

∴ -40, 45, 11, 99, 190

∴ -40, 45, 11, 99, 190

∴ -40, 11, 45, 90, 190

∴ -40, 11, 45, 90, 190

∴ -40, 11, 45, 90, 190

(4) K = 4

∴ -40, 11, 45, 90, 190

∴ -40, 11, 45, 90, 190

∴ -40, 11, 45, 90, 190

∴ -40, 11, 45, 90, 190

∴ -40, 11, 45, 90, 190

Sorted element list:

-40, 11, 45, 90, 190

=> Algorithm:

1 Start

2 if (K = 1 to n-1)

3 Set K = 1 = i

4 While i ≤ K

5 Then swap,
If data[i] > data[i+1]
swap and i = i+1;
else
no swap

6 Stop.

=> Advantages of Bubble Sort.

1 Easy to understand

2 No, external memory needed.

3 Easy to implement.

=> Disadvantages of Bubble sort.

1 Efficient only small list.

2 We can not use this sort to longer element list.

3 Very Expensive.

=> Complexity

- Time complexity :

Average case - $O(n^2)$
worst case - $O(n^2)$

- Space Complexity : $O(1)$

* Merge sort :

Merge sort is the sorting method that follows the divide and conquer method.

In merge sort array is divide into different part.

Merge sort is one most popular and efficent method.

=> Example :

70	20	30	40	10	50	60

Here, First array divides into two equal part.

70	20	30		40	10	50	60

Againe, divide into parts

70	20	30	40	10	50	60

Againe, divide into parts.

70	20	30	40	10	50	60

Here, We merge 20 and 70 or
10 and 40 or
50 and 60.

20	70	30	10	40	50	60

Againe merge, ~~70, 20~~ 20, 30 and 70 or
10, 40 and 50, 60

20	30	70	10	40	50	60

Merge

10	20	30	40	50	60	70

Sorted element list

=> Algorithm :

1 Start

2 If in a list only one element then it is sorted, return.

3 else, Divide element list into equal part untill they no more divide.

4 Merge the divide element list into the sorted order.

5 End.

=> Advantages

1 It can be used for internal and external sorting

2 It is efficient method.

3 Easy Algorithm.

=> DisAdvantages.

1 It needs a extra memory.

2 Merge sort, element divide method is very long.

=> Complexity :

- Time Complexity :

  Averge case : O(n logn)
  Worst case : O(n logn)

- Space Complexity : O(n)

* Selection Sort :

Selection Sort is a simplest sorting techniques in a sorting.

Selection Sort, Every element get Pass and number of pass is n-1.

In this sort, One element compare in a order to other element, this element compare untill one element get sort in the list.

-> Example : 65, 30, 22, 80, 47

Number of Pass = n - 1
= 5 - 1
= 4

- Pass-1 - tack First element 65 and and compare with second element 30.
∴ 30 < 65
∴ So small element is 30.

Now take 30 element and compare with next element 22.
∴ 22 < 30
∴ So, 22 small element.

Now take 22 element and compare with 80 and 47
∴ 22 < 80 and 22 < 47

Here, Smallest element is 22.

So, Interchange the position of 65 and 22.

∴ 22, 30, 65, 80, 47

- Pass-2 - Tack second and small element 30. and compare with 65
∴ 30 < 65
So, 30 is small element.

Again, take 30 element and compare with 65, 80 and 47

65, 80 and 47 is larger than 30.

So, 30 is smallest and sorted second element.

∴ 22, 30, 65, 80, 47

- Pass-3 : Now take 65 element and compare with 80 and 47.

∴ 65 < 80

∴ 65 is small element.

∴ 65 > 47

∴ 47, is small element, Then interchange the position of 65 and 47.

∴ 22, 30, 47, 80, 65.

- Pass-4 : Now take 80 element and compare with 65.

∴ 65 ≯ 80

∴ 65 is smaller than 80.

So, Interchange the position of 65 and 80.

- Sorted list:
  22, 30, 47, 65, 80

→ Algorithm:

1 Start
2 Set First element min.
3 Compare First min element to other min element and swap.
4 Increment min to point to next element.
5 Repeat untill list is sorted.
6 End.

→ Advantages:

1 Selection sort is more efficient on small list.
2 In selection sort, no extra ~~element is~~ memory is requires.
3 Selection Sort is an in-place algorithm.

-> Disadvantages:

1 Selection Sort is not efficient on large list.

2 Selection Sort decrease the performance.

-> Complexity:

- Time Complexity-

Averge Case- $O(n^2)$
Worst Case- $O(n^2)$

- Space Complexity- $O(0)$

* Quick Sort:

Quick Sort is the fastest algorithm to the another sort.

Quick Sork working is easy and fastest.

In Quick Sort, we have to set first element as a Pivot element

Left side of the pivot element is less than the pivot element.

and Right side of the pivot element is greater than the pivot element.

-> Example:

90, 77, 60, 99, 55, 88, 66

1 Here, First we take 90 element as a pivot

90, 77, 60, 99, 55, 88, 66
P

2 Then check R->L - element is small than Pivot element

Here, 66 < 90 -> Swap.

3 66, 77, 60, [99], 55, 88, [90]
P

Then check L -> R - larger element then swap

4 66, 77, 60, [90], 55, [88], 99
P

Then check R->L -> Small element then swap.

∴ 88 < 90

5 66, 77, 60, 88, 55, 90, 99
P

Here, Left side of 90 all element is smaller than Pivot element and Right side of the 90 all element is greater than Pivot element.

So, We have to change Pivot element and take First element as a pivot element.

6 66, 77, 60, 88, 55, 90, 99
P

check R -> L - element is small
∴ 55 < 66 - swap.

7 55, 77, 60, 88, 66, 90, 99
P

check L -> R -> element is Greater.
77 > 66 - swap.

8 55, 66, 60, 88, 77, 90, 99
P

check R -> L - small element
∴ 66 > 60 - swap

9 55, 60, 66, 88, 77, 90, 99
P

Here, again 66 is a proper position. then change Pivot element and take 55 as a Pivot element.

10 55, 60, 66, 88, 77, 90, 99
P

11 Again, Change Pivot element and take 60 as a Pivot element.

55, 60, 66, 88, 77, 90, 99
P

12 Again change Pivot element.

∴ 55, 60, 66, 88, 77, 90, 99
P

13 Again change Pivot element.

∴ 55, 60, 66, 88, 77, 90, 99
P

Check R -> L - small element
∴ 77 < 88 - Swap.

14 55, 60, 66, 77, 88, 90, 99
P

Here, all the elements are Proper Position.

Sorted list - 55, 60, 66, 77, 88, 90, 99.

→ Algorithm :

1 Start

2 Take First element as a Pivot element

3 Take Pivot element and take two variable to pivot left and right to the list.

4 Left points to the low index and Right points to the high index.

5 While value at left is less than pivot more right.

6 While value at Right is greter than pivot more left.

7 If step 5 and 6 not match then swap left and Right.

8 If left and right element is a proper position to the pivot then take new pivot.

9 End.

→ Advantages:

1 Quick Sort working is fast.

2 Quick Sort is very efficient than the other sort.

→ Disadvantages:

1 Quick sort algorithm is not stable.

2 Quick sort worst case complexity is $O(n^2)$.

→ Complexity:

1 Time Complexity:

Avarge Case - $O(n \log n)$

Worst Case - $O(n^2)$

2 Space Complexity: $O(n)$

Page No.
Date : / /

* Heap Sort:

Heap Sort is a based on sorting technique using ~~Bin~~ Binary Heap.

Heap Sort is a similar to the selection sort.

Heap Sort is not required additional memory space to work.

There to condition to perform Heap Sort.

1) Value of node $\leq$ Parent node value

2) Each level of the binary tree must be filled.

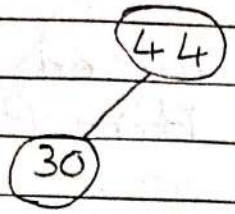
Ex. 44, 30, 50, 22, 60, 55, 77, 55

(1) Take 44. First element of the list as a root.

(44)

(2) Take 30, to 44 element. 44 is geter than the 30 so,

make 30 as as left child



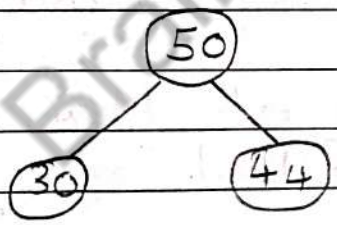
(3) Take next element 50, and Compare whi with 44.

50 > 44 -> So, 50 is be a parent node.

after that compare. 44 and 30

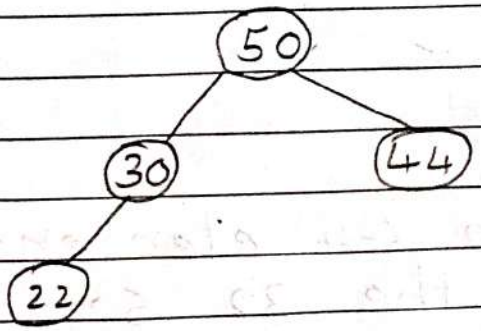
44 > 30.

So, 30 is be a left child and 44 be a right child.



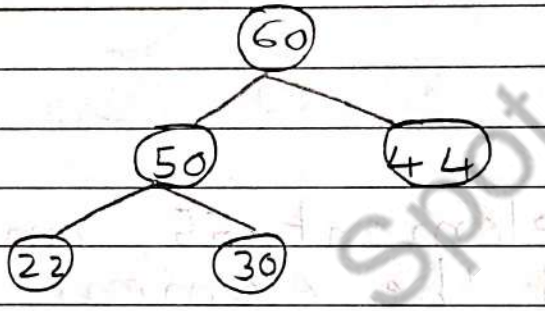
(4) Take next element 22, and compare with 30. 30 > 22.

So, 22 be a left child.

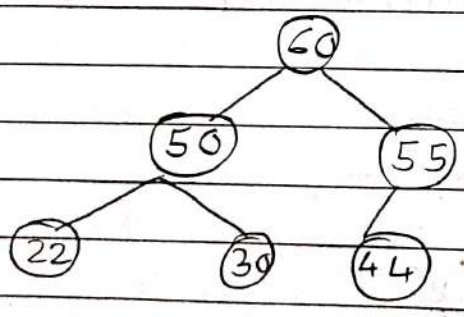


(5) Take next element 60, and compare with all the element. 60 is geater than 50. So, 60 be a parent node.

After that compare 50 with 30. 30 is smaller than 50. So, 50 be a left child of the 60. and 30 be a right chid of the 50.



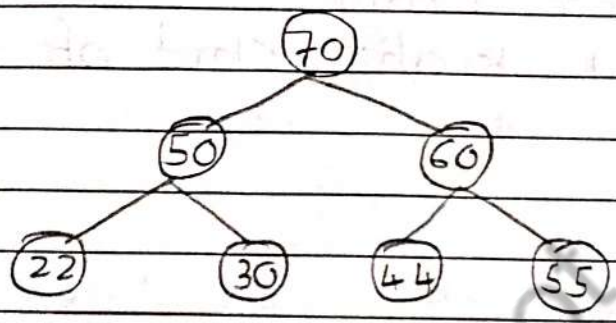
(6) Insert 55, and compare with 44. 44 is smaller than 55. So 44 be a left child of the 55.



(7) Insert 70, and compare with all the element. 70 is a larger element.

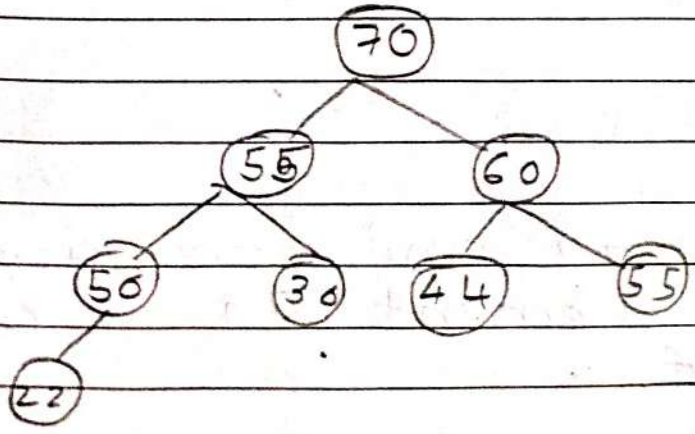
So, 70 is be a parent node.

After that take 60 and compare with 50 and 55. 50 is smaller element. So, 55 be a right child of the 60.



(8) Insert element 55, and compare with all the element, 50 is smaller than 50.

After that Compare 50 with 22 and 30. 22 is smaller element. So, 22 become a left child of the 50.



This Tree is a sort by using Heap Sort.

-> Advantages:

1 This sorting algorithm is very efficient.

2 This sorting is no require any additional memory.

3 It is simpler to understand other sorting algorithm.

-> Disadvantages:

1 This sort working is unstable.

2 This sorting is not efficient for larger data list.

3 This sorting is expensive.

-> Complexity:

Time Complexity:

Average Case - $O(n \log n)$
Worst Case - $O(n \log n)$

SMVS
Page No.
Date : / /

* Linear Search:

Linear Search is also called Sequential search.

Linear Search is a very easy method to searching.

Linear Search, Starting at the begine of the dataset, each is compare with each other, until a match is found.

Ex.

11, 22, 44, 30, 33, 19

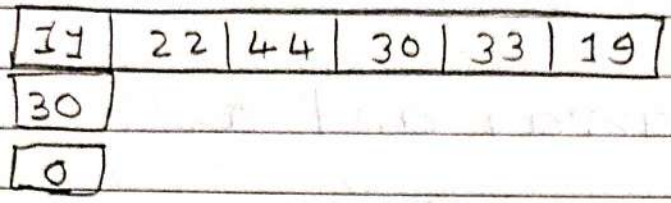
Search the element : 30.

-> For Finding the 30, element, we have to compare 30 element at 0th location of array.

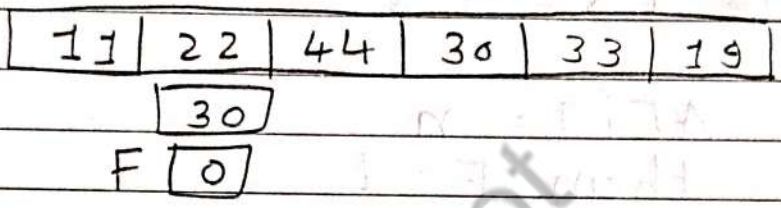
11	22	44	30	33	19

We have to compare 30, to each element of array. when element is found then it become one.

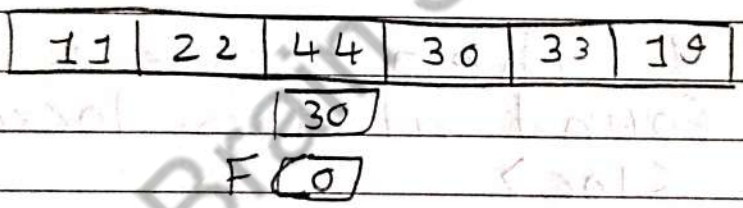
(1) First we compare 30, to 11 element.



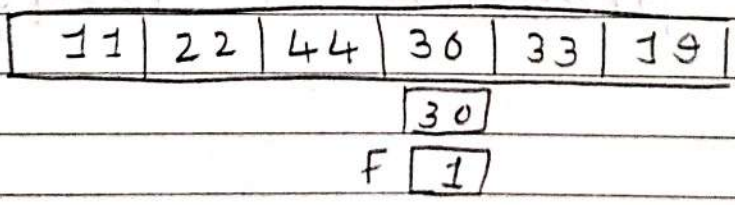
(2) Then, Compare 30 to 22,



(3) Then, Compare 30 to 44.



(4) Then, Compare 30 to 30,



Here, location 3rd is equal to 30 element. and we stop the compare other element.

-> Algorithm:

1 Start

2 Read array and n.

3 set F = 0, i = 0, Loc = 0

4 while i <= n-1

5 if A[i] = n
than F = 1
set loc = i+1
break.

6 If F = 1 then write x element is found at this location <loc>

else, write element is not present in array.

7 End.

-> Advantages:

1 It is simiple to implement.

2 Linear search not required sorted list.

Page No.
Date : / /

→ Disadvantages:

1 It is less efficient.

2 Linear Search is not usefull to longer list.

* Binary Search:

For Binary Search, element should be in a sorted form.

For Binary Search, First we have to Find the middle element in the array.

after that compare the middle element with searching element.

1) If middle element = Searching element

Then search is complete.

2) If middle element > Searching element

Then search First half of the array.

3) If Middle element < Searching element.

Then, Search second half of array.

For Middle element = $\frac{\text{First location} + \text{Last location}}{2}$

Ex. 1, 5, 8, 10, 15, 29, 50, 52, 57, 80

Searching - 29 element.

1	5	8	10	15	29	50	52	57	80
0	1	2	3	4	5	6	7	8	9

(1) Middle element = $0 + 9 / 2 = 4$

A[4] = 15.

~~A[4] = 15~~

A[mid] < Searching element, than the element in second half.

∴ Low = mid + 1 = 4 + 1 = 5

(2) Middle element $= \frac{5+9}{2} = 7$

$A[7] = 52$

$\therefore A[mid] >$ Searching element than the element in first half.

$\therefore$ High = Mid - 1
= 7 - 1
= 6

(3) Middle element $= \frac{6+5}{2} = 5$

$A[5] = 29$

$\therefore A[mid] =$ Searching element

$\therefore$ Searching element $= 29$

Hence, Searching element is found.

Page No.
Date : / /

→ Algorithm:

1 Start

2 If (low > high)
    return.

3 If mid = low + high / 2

4 If (x = = A[mid])
    return A[mid]

5 If (x < A[mid])
  then search A[low] to A[mid-1]
  else,

6 else,
    Search A[Mid + 1] to A[High]

7 Stop.

Page No.
Date : / /

→ Advantages:

1. It is efficient method.
2. It is better than a linear search.

→ Disadvantages:

1. For Binary search, Every element should be sorted.