UNIT 5

SEARCHING AND SORTING

5.1 Searching an element into List :

- Searching is the process of finding particular element from list. If an element is found in the list then we can say that search is successful otherwise search is unsuccessful.
- There are two methods of searching an element from list.
  (1) Linear Search (2) Binary Search

5.1.1 Linear Search :

- This method of searching is also known as sequential search.
- Linear search work on both sorted as well as unsorted list.
- This method of searching works as follows:
- Suppose we want to search an element X from the list of N elements. First the element X is compared with the 1st element in the List. If the element is matched then search is successful.
- If the element is not matched then the element X is compared with next element and this process is repeated until match is found or all the elements in the list are compared.
- Consider the following example:
- The list consist of 5 elements as shown below :

Index	Element
1	11
2	33
3	55
4	22
5	44

- Suppose we want to find 55 in the list. So 55 is compared with the first element in the list which is 11.

Index	Element
1	11
2	33
3	55
4	22
5	44

← Match not Found

- Since match is not found 55 is compared with next element in the list and this process is repeated until match found or all elements in the list is compared as shown below.

Index	Element
1	11
2	33
3	55
4	22
5	44

← Match not found

Index	Element
1	11
2	33
3	55
4	22
5	44

← Match found

- **Algorithm for Linear search method :**

Step 1 : Length = len (myList)
Step 2 : for I in range(Length)
Step 3 : If myList [I] = X then
return I
Step 4 : return none

- **Program for Linear Search Method :**

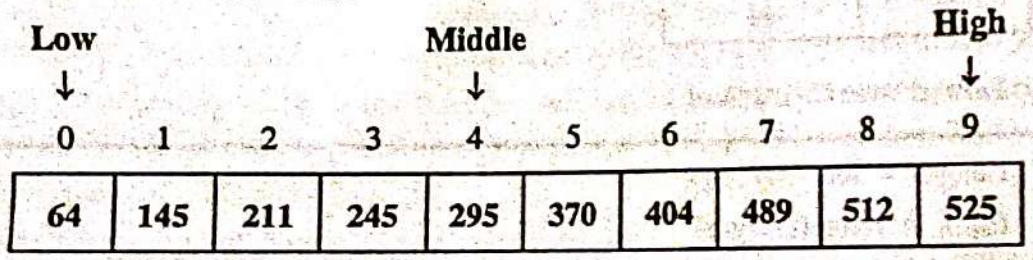
```
def LinierSearch(myList,X):
    Length = len(myList)
    for I in range(Length):
        if(myList[I]==X):
```

```
            return I
        return None
myList=[11,44,55,33,22]
print(myList)
X=int(input("Enter Value To Search:"))
Result = LinierSearch(myList,X)
if(Result==None):
   print("Search Is Unsuccessful")
else:
   print("{} is Found at Position {}".format(X,Result))
```

5.1.2 Binary Search :

- Binary Search method works only for sorted list.
- This method of searching is more efficient as compared to linear search method. Because we does not have to compare the search element with all the elements in the list until the element is found.
- The binary search method of searching an element works as follow :
- Suppose we want to search an element X from the list of N element.
- Determine the Lower and Upper limit of the list by assigning Lower index of the list to LOW and Upper Index of the list to HIGH. Now calculate the MIDDLE position using following equation :

  MIDDLE = (LOW + HIGH)/2
- After finding the MIDDLE index from the list we compare the value of the MIDDLE element with X.
- If the value of the middle element is greater then X then it will exist in the lower interval of the list. So we change the value of HIGH index by MIDDLE – 1.
- If the value of the middle element is smaller then X then it will exist in the upper interval of the list. So we change the value of LOW index by MIDDLE + 1
- This process is repeated until entire list is searched or the element is found.
- Consider following example :



- Suppose we want to find the element X=245 in the above list.
- First,

  LOW = 0, HIGH = 9

  MIDDLE = (LOW + HIGH)/2

  = (0+9)/2

  = 4
- Here the value of MIDDLE element is **295** which is greater then **245** so the element is in the lower interval of the list.

Change HIGH = MIDDLE – 1
= 4-1
= 3

- Now MIDDLE = (LOW + HIGH)/2
= (0 + 3)/2
= 1

- Here the value of the MIDDLE element is **145** which is smaller then **245** so the element is in the upper Interval of the list.

Change LOW = MIDDLE + 1
= 1+1
= 2

- Now MIDDLE = (LOW + HIGH)/2
= (2+3)/2
= 2

- Here the value of the MIDDLE element is **211** which is smaller then **245** so the element is in the upper Interval of the list.

Change LOW = MIDDLE + 1
= 2 + 1
= 3

- Now MIDDLE = (LOW + HIGH)/2
= (3+3)/2
= 3

- Here the value of the MIDDLE element is **245** which we want to find.
- **Algorithm for Binary Search Method**

Step 1 : LOW ← 0
HIGH ← len(myList)-1

Step 2 : Repeat up to Step4 while LOW ≤ HIGH

Step 3 : MIDDLE ← (LOW+HIGH)/2

Step 4 : If X < A [MIDDLE] then
HIGH ← MIDDLE – 1
Else if X > A [MIDDLE] then
LOW ← MIDDLE + 1
Else
Return MIDDLE

Step 5 : Write "Search is Not Successful"
Return None

❖ Program for Binary Search Method :

```
def BinarySearch(myList,X):
    LOW = 0
    HIGH = len(myList)-1
    while(LOW<=HIGH):
        MIDDLE=(LOW+HIGH)//2
        if(X<myList[MIDDLE]):
            HIGH=MIDDLE-1
        elif(X>myList[MIDDLE]):
            LOW=MIDDLE+1
        else:
            return MIDDLE
    return None
myList=[11,22,33,44,55]
print(myList)
X=int(input("Enter Value To Search:"))
Result = BinarySearch(myList,X)
if(Result==None):
    print("Search Is Unsuccessful")
else:
    print("{} is Found at Position {}".format(X,Result))
```

5.1.3 Compare Linear Search with Binary Search.

Linear Search	Binary Search
(1) Linear Search is used to find an element from sorted as well as unordered list.	(1) Binary Search is used to find an element from sorted list only.
(2) This technique is less efficient.	(2) This technique is more efficient.
(3) The order of Linear search is O (N).	(3) The order of Binary Search is O (Log_2N)

5.2 Sorting Methods :

- ❖ Sorting is the process of arranging elements of a list in particular order either ascending or descending.
- ❖ There are various types of sorting methods available. Sorting methods are basically divided into two sub categories.

(1) Internal Sort :

- ❖ The sorting method that does not required external memory for sorting the elements is known as internal sort. It is useful when we have to sort fewer amounts of elements.
- ❖ Following are the example of internal sort method.

1. Bubble sort 2. Quick sort 3. Selection sort 4. Insertion sort

(2) External Sort :

- The sorting method that required external memory for sorting the elements is known as external sort. It is useful when we have to sort large amount of elements.
  1. Merge sort
  2. Radix sort

1 Bubble Sort :

- Bubble sort is a simple sorting method. Bubble sort woks fine for the smaller number of elements in the list.
- Bubble sort is an example of Internal Sort.
- In order to sort N elements using bubble sort technique we required to perform maximum N–1 PASS.
- During first PASS 1st and 2nd element are compared and if 1st element is greater then 2nd element then they are interchanged. Now 2nd and 3rd element is compared if 2nd element is greater then 3rd element then they are interchanged. This process is repeated for all remaining elements. Thus after the completion of first PASS the element with the largest value is placed at its proper position.
- During second PASS the same process is repeated and the element with next largest value is placed at its proper position.
- During each successive PASS the element with next largest value is placed at its proper position.
- During first pass we required N–1 comparisons. During second pass we required N–2 comparisons and during I^{th} pass we required N–I comparisons. So total number of comparisons is given as :

$$\sum_{I=1}^{N-1} (N-I) = N\ (N-1)/2$$

- Thus the order of comparisons is proportional to N^2 i.e $O(N^2)$
- **Consider the following example :**

5 2 35 9 3 –2 52 0

Element	Pass1	Pass2	Pass3	Pass4	Pass5	Pass6	Pass7
5	2	2	2	2	–2	–2	–2
2	5	5	3	–2	2	0	**0**
35	9	3	–2	3	0	**2**	2
9	3	–2	5	0	**3**	3	3
3	–2	9	0	**5**	5	5	5
–2	35	0	**9**	9	9	9	9
52	0	**35**	35	35	35	35	35
0	**52**	52	52	52	52	52	52

❖ **Algorithm for Bubble Sort :**

```
Step 1 :  Length ← len(myList)
Step 2 :  Repeat up to Step4 for I in range(Length)
Step 3 :  Repeat Step4 for J in range(Length-I-1)
Step 4 :  if a [J] > a [J + 1] then
              TEMP ← a [J]
              a [J] ← a [J + 1]
              a [J + 1] ← TEMP
Step 5 :  Exit
```

❖ **Program for Bubble Sort Method**

```
def BubbleSort(myList):
    Length=len(myList)
    for I in range(Length):
      for J in range(Length-I-1):
            if(myList[J]>myList[J+1]):
            temp=myList[J]
            myList[J]=myList[J+1]
            myList[J+1]=temp
myList=[10,9,8,7,6,5,4,3,2,1,0]
print("List Before Sorting:")
print(myList)
BubbleSort(myList)
print("List After Sorting:")
print(myList)
```

5.2.2 Selection Sort :

- ❖ Selection Sort method is a simple sorting method. It works fine for smaller number of eleme in the list.
- ❖ Selection sort is an example of Internal Sort.
- ❖ In order to sort N elements using selection sort technique we required to perform N–1 PAS
- ❖ During first PASS entire array is searched from first element to find the smallest element. Wh smallest element is found, it is interchanged with the first element in the array. Thus the eleme with the smallest value is placed at first position in the array.
- ❖ During second PASS array is searched from second record to find the second smallest elemer When this element is found, it is interchanged with the second element in the array. Thus t element with the second smallest value is placed at second position in the array.
- ❖ During each successive PASS the smallest element is placed at its proper position and this proce continues until all the elements in the array are arranged in ascending order.
- ❖ During first pass we required N–1 comparisons. During second pass we required N–2 comparison and during I^{th} pass we required N–I comparisons. So total number of comparisons is given as

$\sum_{I=1}^{N-1} (N-I) = N(N-1)/2$

Thus the order of comparisons is proportional to N^2 i.e $O(N^2)$

Consider the following example :

5 2 35 9 3 –2 52 0

Element	Pass1	Pass2	Pass3	Pass4	Pass5	Pass6	Pass7
5	–2	–2	–2	–2	–2	–2	–2
2	2	0	0	0	0	0	0
35	35	35	2	2	2	2	2
9	9	9	9	3	3	3	3
3	3	3	3	9	5	5	5
–2	5	5	5	5	9	9	9
52	52	52	52	52	52	52	35
0	0	2	35	35	35	35	52

Algorithm for Selection Sort :

```
Step 1 : Length ← len(myList )
Step 2 : Repeat up to Step 6 for I in range(Length)
Step 3 : MIN ← myList [I]
         POS ← I
Step 4 : Repeat step 5 for J in range(I+1,Length)
Step 5 : if myList [J] < MIN then
         MIN ← myList [J]
         POS ← J
Step 6 : TEMP ← myList [I]
         myList [I] ← myList [POS]
         myList [POS] ← TEMP
Step 7 : Exit
```

Program for Selection Sort Method :

```
def SelectionSort(myList):
    K=
    Length=len(myList)
    for I in range(Length):
        MIN=myList[I
        PO
```

```
            for J in range(I+1,Length):
                if(myList[J]<MIN
                    MIN=myLi

            temp=myList[I
            myList[I]=myList[POS]
            myList[POS]=temp
            print("PASS {}:{}".format(K,myList))
            K=
myList=[10,9,8,7,6,5,4,3,2,1,0]
print("List Before Sorting:")
print(myList)
SelectionSort(myList)
print("List After Sorting:")
print(myList)
```

5.2.3 Quick Sort :

- Quick Sort method is an efficient sorting method for larger List. It works fine for the list hav large number of elements. It uses Divide and conquers strategy in which a list is divided i two smaller lists.
- First initialize LOW with index of the first element and HIGH with index of last element.
- Now find the pivot element from the list of elements using the equation:
  PIVOT ← (LOW + HIGH) / 2
- Now we scan elements from left to right and compare each element with PIVOT element. If scanned element is less then the PIVOT element we scan next element and increment the va of LOW. Repeat same procedure until we found the element which is greater then the PIV element.
- Now we scan elements from right to left and compare each element with PIVOT element. If scanned element is greater then the PIVOT element we scan next element and decrement the va of HIGH. Repeat same procedure until we found the element which is less then the PIVOT elem
- Now we compare the value of LOW and HIGH. If LOW is less then HIGH then we intercha the element which are at the index LOW and HIGH.
- Increment the value of LOW and decrement the value of HIGH. Repeat above procedure w value of LOW less then or equal to value of HIGH.
- After the completion of first PASS the entire list of elements is divided in to two lists. First contains elements which are less then the PIVOT element and second list contains elements wh are greater then the PIVOT element.
- The above procedure is recursively repeated for the sub lists until all the elements in the are sorted.
- The order of comparison for this method is o (nlogn)
- Consider the following example :
  5 2 35 9 3 –2 52 0

Element	5	2	35	9P	3	–2	52	0
PASS–1	{ 5	2	0P	–2	3}	9	{52	35}
PASS–2	{ –2 }	0	{2	5P	3}	9	52	35
PASS–3	–2	0	{2	3}	5	9	52P	35
PASS–4	–2	0	2	3	5	9	35	52

❖ **Algorithm For Quick Sort :**

Step 1 : LOW←FIRST
HIGH←LAST
Step 2 : PIVOT←(LOW+HIGH)//2
Step 3 : Repeat thru step-6 while LOW<HIGH
Step 4 : Repeat while myList[LOW]< myList [PIVOT]
LOW←LOW+1
Step 5 : Repeat while myList [HIGH]> myList [PIVOT]
HIGH←HIGH-1
Step 6 : if (LOW<=HIGH) then
TEMP=myList[LOW]
myList[LOW]=myList[HIGH]
myList[HIGH]=TEMP
LOW←LOW+1
HIGH←HIGH-1
Step 7 : IF(FIRST<HIGH)
call QUICK_SORT(A,FIRST,HIGH)
IF (LOW<LAST)
call QUICK_SORT(A,LOW,LAST)
Step 8 : Exit

❖ **Program for Quick Sort Method :**

```
def  QuickSort(myList,FIRST,LAST):
    LOW=FIRST
    HIGH=LAST
    PIVOT=(LOW+HIGH)//2
    while(LOW<HIGH):
        while(myList[LOW]<myList[PIVOT]):
            LOW=LOW+1
        while(myList[HIGH]>myList[PIVOT]):
            HIGH=HIGH-1
```

```
            if(LOW<=HIGH):
                TEMP=myList[LOW]
                myList[LOW]=myList[HIGH]
                myList[HIGH]=TEMP
                LOW=LOW+1
                HIGH=HIGH-1
        print(myList)
        if(FIRST<HIGH):
            QuickSort(myList,FIRST,HIGH)
        if(LOW<LAST):
            QuickSort(myList,LOW,LAST)
    myList=[43,3,20,89,4,77]
    print("List Before Sorting:")
    print(myList)
    QuickSort(myList,0,len(myList)-1)
    print("List After Sorting:")
    print(myList)
```

5.2.4 Insertion Sort :

- Insertion Sort method is a simple sorting method. It works fine for smaller number of elements in the list.
- In order to sort N elements using insertion sort method we required to perform N PASS.
- During PASS–1, 1st element of the list is scanned which is trivially sorted.
- During PASS–2, 2nd element of the list is scanned and it is compared with 1st element in the list. If 2nd element is smaller then 1st element then they are interchanged.
- During PASS–3, 3rd element of the list is scanned and it is compared with 2nd and 1st element respectively.
- The same process is repeated until all the elements from the list are scanned and arranged in ascending order.
- Thus the order of comparisons is proportional to N^2 i.e $O(N^2)$
- Consider the following example :

  5 2 35 9 3 –2 52 0

Element	5	2	35	9	3	–2	52	0
PASS–1	5	2	35	9	3	–2	52	0
PASS–2	2	5	35	9	3	–2	52	0
PASS–3	2	5	35	9	3	–2	52	0
PASS–4	2	5	9	35	3	–2	52	0
PASS–5	2	3	5	9	35	–2	52	0
PASS–6	–2	2	3	5	9	35	52	0
PASS–7	–2	2	3	5	9	35	52	0
PASS–8	–2	0	2	3	5	9	35	52

❖ **Algorithm for Insertion Sort :**

Step 1 : Length ← len(myList)
Step 2 : Repeat up to step 5 for I in range(Length)
Step 3 : KEY ← myList [I]
Step 4 : Repeat step 5 for J in range(I,0,-1)
Step 5 : if KEY < myList [J - 1] then
TEMP ← myList [J]
myList [J] ← myList [J - 1]
myList [J - 1] ← TEMP
Step 6 : Exit

❖ **Program for Insertion Sort Method :**

```
def InsertionSort(myList):
    K=1
    Length=len(myList)
    for I in range(Length):
        KEY=myList[I]
        for J in range(I,0,-1):
            if(KEY<myList[J-1]):
                temp=myList[J]
                myList[J]=myList[J-1]
                myList[J-1]=temp
    print("PASS {}:{}".format(K,myList))
    K=K+1
myList=[10,9,8,7,6,5,4,3,2,1,0]
print("List Before Sorting:")
print(myList)
```

```
InsertionSort(myList)
print("List After Sorting:")
print(myList)
```

5.2.5 Merge Sort :

- Simple merge sort is used to merge two sorted list in a single sorted list.
- In a simple merge sort we compare the elements of both lists and the element which has the smallest value is placed in a new list.
- This process is repeated until all the elements from both the lists are placed in a new list.
- Consider the following example :

  List1 : 28 31 37 39

  List2 : 24 35 43 53 85 89
- First we compare the elements of both the table and the element which has the smallest value is placed in a new table.
- So the trace is as follow :

Comparision	New List
	Empty
List1 [0] with List2 [0], 28 with 24, Here 24 is smaller so it is placed in a new list.	24
List1 [0] with List2 [1], 28 with 35, Here 28 is smaller so it is placed in a new list.	24 28
List1 [1] with List2 [1], 31 with 35, Here 31 is smaller so it is placed in a new list.	24 28 31
List1 [2] with List2 [1], 37 with 35, Here 35 is smaller so it is placed in a new list.	24 28 31 35
List1 [2] with List2 [2], 37 with 43, Here 37 is smaller so it is placed in a new list.	24 28 31 35 37
List1 [3] with List2 [2], 39 with 43, Here 39 is smaller so it is placed in a new list.	24 28 31 35 37 39
Now List1 is empty so place remaining elements of List2 in New List.	24 28 31 35 37 39 43 53 85 89

- Thus the order of comparisons is propartional to NLogN.e O(NLogN)
- **Algorithm for Simple Merge Sort**
- Let N1 = Size of List1 and N2 = Size of List2

```
Step 1 :  I ← 0
          J ← 0
          N1←len(myList1)
          N2←len(myList2)
Step 2 :  Repeat Step3 while I <= N1 - 1 and J <= N2 - 1
Step 3 :  if List1 [I] < List2 [J] then
              List3.append(List1 [I])
              I ← I + 1
          Else
              List3.append(List2 [J])
              J ← J + 1
Step 4 :  Repeat Step5 while I <= N1-1
Step 5 :  List3.append(List1 [I])
          I ← I + 1
Step 6 :  Repeat Step7 while J <= N2-1
Step 7 :  List3.append(List2[J])
          J ← J + 1
```

❖ **Program for Merge Sort Method :**

```
def MergeSort(myList1,myList2,myList3):
    I=0
    J=0
    N1=len(myList1)
    N2=len(myList2)
    while(I<=N1-1  and  J<=N2-1):
        if(myList1[I]<myList2[J]):
            myList3.append(myList1[I])
            I=I+1
        else:
            myList3.append(myList2[J])
            J=J+1
    while(I<=N1-1):
        myList3.append(myList1[I])
        I=I+1
    while(J<=N2-1):
        myList3.append(myList2[J])
        J=J+1
myList1=[28,31,37,39]
myList2=[24,35,43,53,85,89]
myList3=[]
```