Chapter 9

A.

1. B
2. B
3. B
4. D
5. C
6. D
7. A
8. B
9. B
10. C

B.

1. True
2. True
3. True
4. False
5. False
6. True
7. True
8. True
9. False
10. False

C.

1. For, Binary search the elements in a list must be in sorted order. Let us consider the list is in ascending order. The binary search compares the element to be searched i.e. the key element with the element in the middle of the list. The binary search algorithm is based on the following steps.

i) If the key is less than the list's middle element, then the programmer has to search only in the first half of the list.

ii) If the key is greater than the list's middle element, then the programmer has to search only in the second half of the list.

iii) If the element to be found i.e. the key element is equal to lists middle element then search ends.

iv) If the element to be found is not present within the list then the it returns None or -1 which it indicates the element to be searched is not present in the list.

2.

Consider the unsorted list of 8 elements

[55, 58, 90, 33, 42, 89, 59, 71]

<u>Operation</u>

Select **33** the smallest element and swap it with 55 the first element within the list.	55, 58, 90, 33, 42, 89, 59, 71
Select **42** smallest element and swap it with **58** in the remaining list.	**33**, 58, 90, 55, 42, 89, 59, 71
Select **55** smallest element and swap it with **90** in the remaining list.	**33**, **42**, 90, 55, 58, 89, 59, 71
Select **58** smallest element and swap it with **90** in the remaining list.	**33**, **42**, **55**, 90, 58, 89, 59, 71
Select **59** smallest element and swap it with **90** in the remaining list.	**33**, **42**, **55**, **58**, 90, 89, 59, 71
Select **71** element and swap it with **89** in the remaining list.	**33**, **42**, **55**, **58**, **59**, 89, 90, 71
Select **89** element and swap it with **90** in the remaining list.	**33**, **42**, **55**, **58**, **59**, **71**, 90, 89
Final Sorted List	**33, 42, 55, 58, 59, 71, 89, 90**

3. In Linear Search elements are examined sequentially starting from the first element. It compares the element to be searched i.e. the **(Key element)** sequentially with each element in the list. If the element to be searched is found within the list, the linear search returns index of the matching element. If element is not found, the search returns -1. Suppose we are having unsorted list then the analysis with respect to searching the element in unsorted list is as follows.

Unordered List – Analysis of Sequential Search

Case	Best Case	Worst Case	Average Case
Element is present in the list	1	N	$\frac{N}{2}$
Element is not present in the List	N	N	N

4.

Searching is a technique of finding an element from a given list of elements.

Sorting means rearranging the elements of a list, so that they are kept in some relevant order. The order can be either ascending or descending.

Different type of Searching Technique are as follows

a) Binary Search
b) Linear Search

and Different type of Sorting Technique are as follows
a) Bubble sort
b) Selection Sort
c) Quick Sort
d) Insertion Sort
e) Merge Sort

Programming Assignments

1.

```
def Bubble_Sort(MyList):
    for i in range(len(MyList)-1,0,-1):
        for j in range(i):
            if MyList[j]>MyList[j+1]:
                temp,MyList[j]=MyList[j],MyList[j+1]
                MyList[j+1] = temp
MyList = [30, 50, 45,1,6,3,20, 90, 78]
print('Elements of List Before Sorting: ',MyList)
Bubble_Sort(MyList)
print('Elements of List After Sorting: ',end='')
print(MyList)
```

Output:

```
Elements of List Before Sorting:  [30, 50, 45, 1, 6, 3, 20, 90, 78]
Elements of List After Sorting: [1, 3, 6, 20, 30, 45, 50, 78, 90]
```

2.

```
def quickSort( MyList ):
    """ Sorts an array or list using the recursive quick sort algorithm. """
    print('Elements of List are as follows')
    print(MyList)
    n = len( MyList )
    Rec_Quick_Sort( MyList, 0, n-1 )

def Rec_Quick_Sort( MyList, first, last ):
    """ The recursive implementation. """
    if first < last:
        pos = Partition( MyList, first, last )
        """ Split the List into two subLists Left and Right. """
        Rec_Quick_Sort( MyList, first, pos - 1 )
        Rec_Quick_Sort( MyList, pos + 1, last )

def Partition(MyList, first, last ):
    """ Partitions the sublists or subarrays using the first key as the pivot. """

    pivot = MyList[first]    #Select the Pivot element

    # Find the pivot position and move the elements around the pivot.
    i = first + 1
    j = last
    while i < j :

        # Find the first key larger than the pivot.
        while i <= j and MyList[i] <= pivot :
            i = i + 1

 #Find the key from the list that is smaller than or equal to the pivot.
        while j >= i and  pivot <= MyList[j] :
            j = j - 1

        # Swap the two keys if we have not completed this partition.
        if i < j :
            temp= MyList[i]
            MyList[i] =  MyList[j]
            MyList[j] = temp

     # Put the pivot in the proper position.
      #Swap pivot with MyList[j]
    temp=MyList[first]
    MyList[first] = MyList[j]
    MyList[j] = temp

    # Return the index position to partition into left and right
    return j

MyList=[50, 30, 10, 90, 80, 20, 40, 60];
quickSort(MyList)
print('Elements of List after Sorting Using Quick Sort')
print(MyList)
```

Output:

```
Elements of List are as follows
[50, 30, 10, 90, 80, 20, 40, 60]
Elements of List after Sorting Using Quick Sort
[10, 20, 40, 30, 50, 60, 80, 90]
```

3.

```
def selection_sort(L1, i, j, flag):
    size = len(L1)
    if (i < size - 1):
        if (flag):
            j = i + 1;
        if (j < size):
            if (L1[i] > L1[j]):
                 L1[i], L1[j] = L1[j], L1[i]
            selection_sort(L1, i, j + 1, 0);
        selection_sort(L1, i + 1, 0, 1);


L1 = [55, 58, 90, 33, 42, 89, 59, 71]
selection_sort(L1, 0, 0, 1)
print(L1)
```

Output

```
[33, 42, 55, 58, 59, 71, 89, 90]
```

4.

```
def Binary_Search(MyList,key):
     low=0
     high=len(MyList)-1
     while low<=high:
         mid=(low+high)//2  #Find the middle index
         if MyList[mid]==key: If key matches the mid index element
             return mid      #If so  return index
         elif key>MyList[mid]: # else if key is greater
             low=mid+1
         else:
             high=mid-1
     return -1   #If no match return -1
MyList=[10,20,30,34,56,78,89,90]
print(MyList)
key=(eval(input("Enter the number to Search:")))
x=Binary_Search(MyList,key)
if(x==-1):
   print(key,"is not present in the list")
else:
   print(" The Element ",key,"is found at position ",x+1)

Output:
#Test Case1
[10, 20, 30, 34, 56, 78, 89, 90]
Enter the number to Search:20
The Element  20 is found at position  2
#Test Case 2
[10, 20, 30, 34, 56, 78, 89, 90]
Enter the number to Search:43
43 is not present in the list
```

5.

```
def mergeSort(MyList):
   if len(MyList)>1:
        mid = len(MyList)//2
        leftList = MyList[:mid]
        rightList = MyList[mid:]
      '''Merge sort to the left part of the list from 0 to mid-1.'''
        mergeSort(leftList)
'''Merge sort to the right part of the list from mid to len(List)'''
        mergeSort(rightList)
        i=0
        j=0
        k=0
        ''' Merge Two Sorted List i.e. LeftList and RightList'''
        while i < len(leftList) and j < len(rightList):
            if leftList[i] < rightList[j]:
                MyList[k]=leftList[i]
                i=i+1
            else:
                MyList[k]=rightList[j]
                j=j+1
            k=k+1

        while i < len(leftList):
            MyList[k]=leftList[i]
            i=i+1
            k=k+1

        while j < len(rightList):
            MyList[k]=rightList[j]
            j=j+1
            k=k+1

MyList = [54,26,93,17,77,31,44,55,20]
print('List Before Sorting',MyList)
mergeSort(MyList)
print('List After Sorting',MyList,end='')
```

Output:

```
List Before Sorting [24, 11, 9, 2, 17, 16, 14, 3]
List After Sorting [2, 3, 9, 11, 14, 16, 17, 24]
```