Lab 8

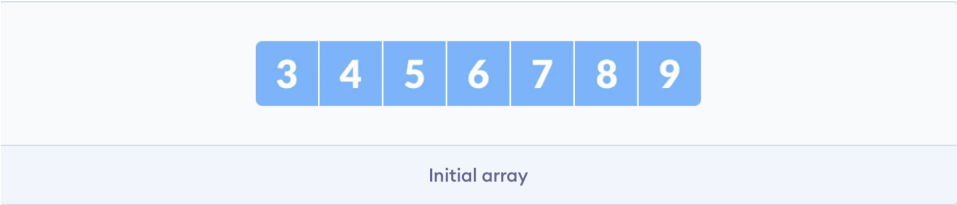
Aim: This lab will focus on Binary search and sorting algorithms: Bubble sort and Insertion sort

- **Binary search**

Finding an element's location in a sorted array can be done using the searching algorithm known as binary search. In this approach, the element is always searched in the middle of a portion of an array. Binary search can be implemented only on a sorted list of items. If the elements are not sorted already, we need to sort them first.
Binary Search Algorithm can be implemented in two ways which you all will discuss:
1. Iterative Method
2. Recursive Method
The recursive method follows the divide and conquer approach.
Important: To use the divide and conquer algorithm, recursion is used.
How Divide and Conquer Algorithms Work?
Here are the steps involved:
Divide: Divide the given problem into sub-problems using recursion.
Conquer: Solve the smaller sub-problems recursively. If the subproblem is small enough, then solve it directly.
Combine: Combine the solutions of the sub-problems that are part of the recursive process to solve the actual problem.

- The general steps for both methods are discussed below.

1. The array in which searching is to be performed is:



Let x = 4 be the element to be searched.

2. Set two pointers low and high at the lowest and the highest positions respectively.

3. Find the middle element mid of the array ie. arr[(low + high)/2] = 6.

Mid element

4. If x == mid, then return mid.Else, compare the element to be searched with m.
5. If x > mid, compare x with the middle element of the elements on the right side of mid. This is done by setting low to low = mid + 1.
6. Else, compare x with the middle element of the elements on the left side of mid. This is done by setting high to high = mid - 1.

Finding mid element

7. Repeat steps 3 to 6 until low meets high.

Mid element

8. x = 4 is found.

Create a new file named as "Practice_sorting.py" in laboratory 8 folder
In the file -

Perform code demonstration for Binary search:

1. Iterative Approach

```python
# Binary Search in python


def binarySearch(array, x, low, high):

    # Repeat until the pointers low and high meet each other
    while low <= high:

        mid = low + (high - low)//2

        if array[mid] == x:
            return mid

        elif array[mid] < x:
            low = mid + 1

        else:
            high = mid - 1

    return -1


array = [3, 4, 5, 6, 7, 8, 9]
x = 4

result = binarySearch(array, x, 0, len(array)-1)

if result != -1:
    print("Element is present at index " + str(result))
else:
    print("Not found")
```

Displayed to the screen:

```
Element is present at index 1
```

2. Recursive Approach

```
def binarySearch(array, x, low, high):

    if high >= low:

        mid = low + (high - low)//2

        # If found at mid, then return it
        if array[mid] == x:
            return mid

        # Search the left half
        elif array[mid] > x:
            return binarySearch(array, x, low, mid-1)

        # Search the right half
        else:
            return binarySearch(array, x, mid + 1, high)
    else:
        return -1
array = [3, 4, 5, 6, 7, 8, 9]
x = 4
result = binarySearch(array, x, 0, len(array)-1)
if result != -1:
    print("Element is present at index " + str(result))
else:
    print("Not found")
```

Displayed to the screen:

```
Element is present at index 1
```

- **Sorting Algorithms**

A sorting algorithm is an algorithm that puts elements of a list into an order. The most frequently used orders are numerical order and lexicographical order, and either

ascending or descending. Efficient sorting is important for optimizing the efficiency of other algorithms (such as search and merge algorithms) that require input data to be in sorted lists.
We are going to talk about 2 different sorting algorithms today:

- **Bubble sort**

Bubble sort is a sorting algorithm that compares two adjacent elements and swaps them until they are in the intended order.
Just like the movement of air bubbles in the water that rise up to the surface, each element of the array moves to the end in each iteration. Therefore, it is called a bubble sort.

- ***Working with Bubble Sort***

Suppose we are trying to sort the elements in ascending order.

1. First Iteration (Compare and Swap)

~Starting from the first index, compare the first and the second elements. ~If the first element is greater than the second element, they are swapped. ~Now, compare the second and the third elements. Swap them if they are not in order. ~The above process goes on until the last element

2. Remaining Iteration

The same process goes on for the remaining iterations.After each iteration, the largest element among the unsorted elements is placed at the end.

3. In each iteration, the comparison takes place up to the last unsorted element.
4. The array is sorted when all the unsorted elements are placed at their correct positions.

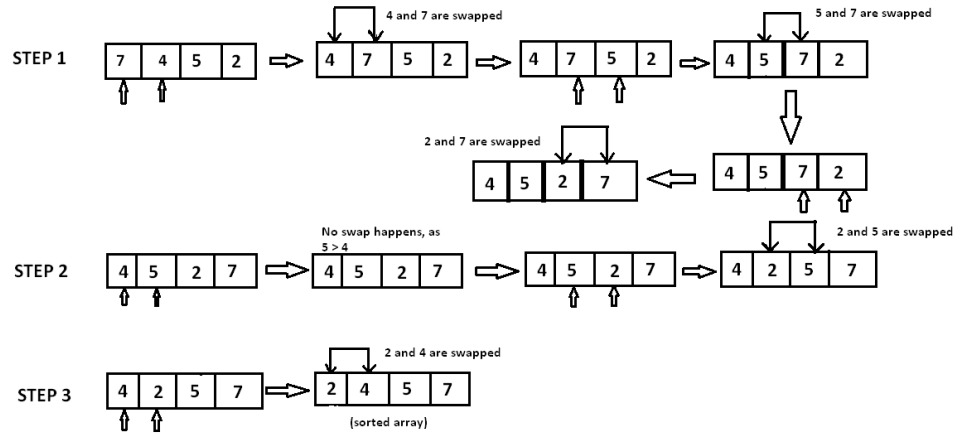
Code Demonstration

```
# Bubble sort in Python

def bubbleSort(array):

  # loop to access each array element
  for i in range(len(array)):

    # loop to compare array elements
    for j in range(0, len(array) - i - 1):

      # compare two adjacent elements
      # change > to < to sort in descending order
      if array[j] > array[j + 1]:

        # swapping elements if elements
        # are not in the intended order
        temp = array[j]
        array[j] = array[j+1]
        array[j+1] = temp


data = [-2, 45, 0, 11, -9]

bubbleSort(data)

print('Sorted Array in Ascending Order:')
print(data)
```

Output should be:

```
Sorted Array in Ascending Order:
[-9, -2, 0, 11, 45]
```

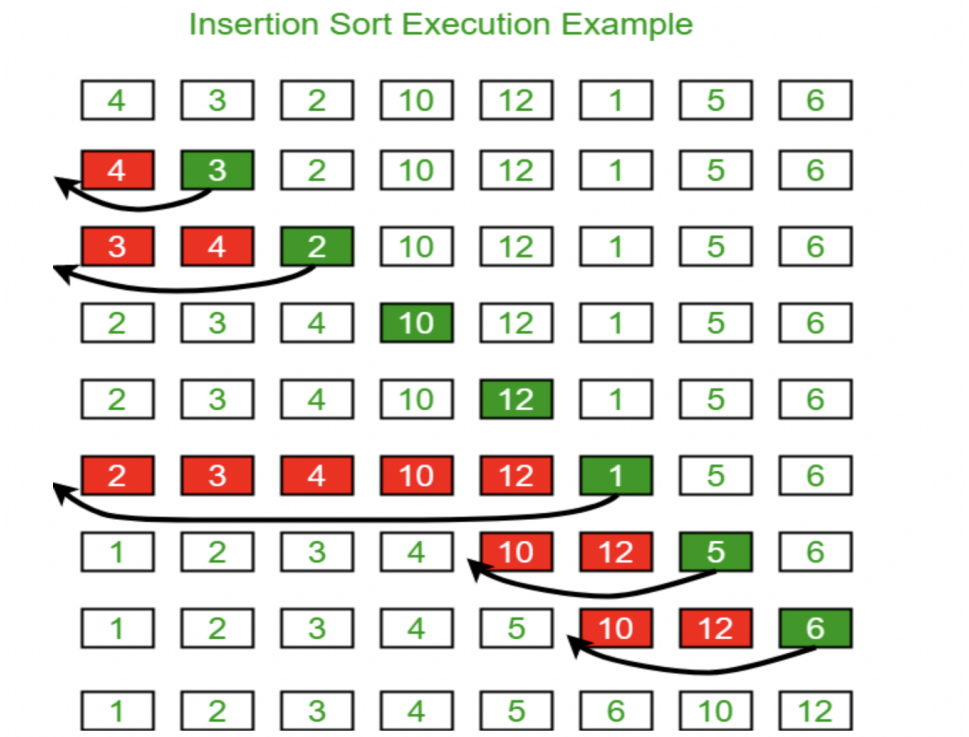
Let's try to understand the an another example: A [ ] = { 7, 4, 5, 2}



In step 1, *7* is compared with *4*. Since *7>4*, *7* is moved ahead of *4*. Since all the other elements are of a lesser value than *7*, *7* is moved to the end of the array.
Now the array is *A[]={4,5,2,7}*.
In step 2, *4* is compared with *5*. Since *5>4* and both *4* and *5* are in ascending order, these elements are not swapped. However, when *5* is compared with *2*, *5>2* and these elements are in descending order. Therefore, *5* and *2* are swapped.
Now the array is *A[]={4,2,5,7}*.
In step 3, the element *4* is compared with *2*. Since *4>2* and the elements are in descending order, *4* and *2* are swapped.
The sorted array is *A[]={2,4,5,7}*.

- **Insertion Sort:**

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part. Algorithm
To sort an array of size n in ascending order:
1: Iterate from arr[1] to arr[n] over the array.
2: Compare the current element (key) to its predecessor.
3: If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.



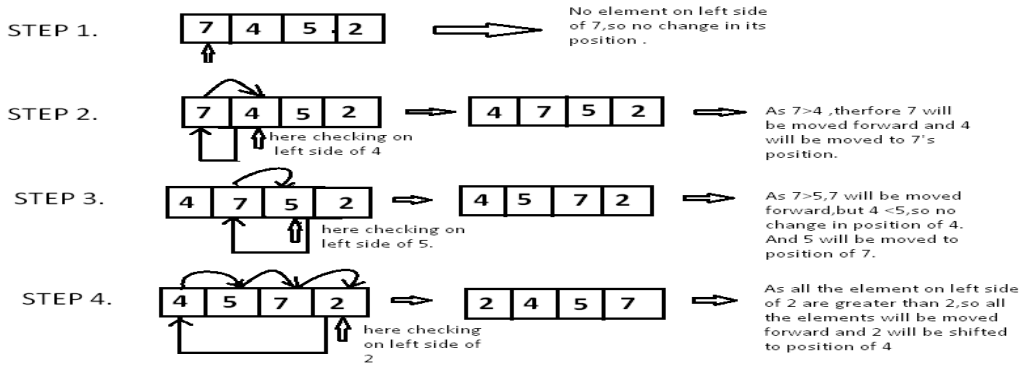
Code demonstration

```
# Insertion sort


def insertionSort(array):

    for step in range(1, len(array)):
        key = array[step]
        j = step - 1

        # Compare key with each element on the left of it until an element smaller than it is found
        # For descending order, change key<array[j] to key>array[j].
        while j >= 0 and key < array[j]:
            array[j + 1] = array[j]
            j = j - 1

        # Place key at after the element just smaller than it.
        array[j + 1] = key


data = [9, 5, 1, 4, 3]
insertionSort(data)
print('Sorted Array in Ascending Order:')
print(data)
```

Output should be:

```
Sorted Array in Ascending Order:
[1, 3, 4, 5, 9]
```

Another Example:
Take array *A[]=[7,4,5,2]*.



Since *7* is the first element and has no other element to be compared with, it remains at its position. Now when on moving towards *4*, *7* is the largest element in the sorted list and greater than *4*. So, move *4* to its correct position i.e., before *7*. Similarly with *5*, as *7* (largest element in the sorted list) is greater than *5*, we will move *5* to its correct position. Finally for *2*, all the elements on the left side of *2* (sorted list) are moved one position forward as all are greater than *2* and then *2* is placed in the first position. Finally, the given array will result in a sorted array.