EXPERIMENT-2

Aim: To implement Linear search and Binary search and analyse its time complexity.

Pseudo Code:

1) Linear Search:

```
procedure linearsearch (list, value)
    for each item in the list
        if item == value then
            return the item's location
        end if
    end for
end procedure
```

2) Binary Search:

```
procedure binarySearch (list, value)
    set beg = 0, end = size of array - 1, mid = (beg + end)/2
    while beg ≤ end and list[mid] != value do:
        if value < list[mid] then
            set end = mid - 1
        end if
        else
            set beg = mid + 1
        end else
        set mid = (beg + end)/2
    end while
    if value == list[mid] then
        return = the items location
    end if
end procedure
```

EXPERIMENT-3

Aim: To implement following algorithm using array as data structure and analyse its time complexity.

a) **Quick sort**
b) **Merge sort**

Pseudo Code:

a) Quick Sort

```
QuickSort (arr, low, high);
    if low < high
        pivotIndex = Partition (arr, low, high)
        QuickSort (arr, low, pivotIndex - 1)
        QuickSort (arr, pivotIndex + 1, high)
        // Recursive call the left/right subarray

Partition (arr, low, high);
    pivot = arr[high]
    i = low - 1

    for j from low to high - 1
        if arr[j] <= pivot
            i = i + 1
            Swap (arr[i], arr[j])

    Swap (arr[i+1], arr[high]) // place the pivot element in its correct position

    return i+1

end procedure
```

EXPERIMENT-1

Aim: To implement following algorithm using array as data structure and analyse its time complexity.

a) **Bubble sort**
b) **Insertion sort**
c) **Selection sort**

Psendo Code:

a) Bubble Sort :

```
Procedure bubble sort ( list : array of items)
    loop = list.count;
    for i = 0 to loop-1 do;
      for j = 0 to loop-1 do;
        if list[j] > list[j+1] then
          swap (list[j], list[j+1])
        end if
      end for
    end for
end procedure
```

b) Insertion sort :

```
procedure insertionsort (list : array of items)
    loop = list.count;
    for i = 0 to loop-1 do;
      for j = i+1 to j>0 do:
        If list[j-1] > list[j] then
          swap (list[j-1], list[j])
    end if
```

```
        else
            bswak
        end else
    end for
  end for
end procedure.
```

c) Selection Sort :

```
procedure selection sort (dist : array of items)
    loop = dist.count;
    for ( i = 0 to i < loop-1 do :
        for  j = i+1 to j < loop do;
            if  dist[j] > dist[j] then:
                swap (dist[i], dist[j])
            end if
        end for
    end for
end procedure.
```

b) Merge Sort

```
MergeSort (arr)
    if length(arr) <= 1
        return arr
    mid = length(arr)/2
    left = arr[0 : mid]      // subarray from start to mid
    right = arr[mid : ]      //    "    from mid to end

    left = MergeSort(left)
    right = MergeSort(right)
    // Recursively sort the left/right subarray
    return merge(left, right) // merge the sorted subarrays

merge(left, right):
    result = []
    while left is not empty & right is not empty
        if left[0] <= right[0]
            append left[0] to result
            remove left[0] from left
        else
            append right[0] to result
            remove right[0] from right

    append remaining elements of left to result
    append remaining elements of right to result

    return result

end procedure
```