Problem for Assignment-2: Searching & Sorting

1) Implement Binary Search
2) Implement Merge Sort
3) Implement Quick Sort
4) Implement Insertion Sort
5) Write a program to sort list of strings (similar to that of dictionary)

---------------------ANSWERS------------------------

1) Implement Binary Search

Binary Search Algorithm

Iteration Method

```
do until the pointers low and high meet each other.
    mid = (low + high)/2
    if (x == arr[mid])
        return mid
    else if (x > arr[mid]) // x is on the right side
        low = mid + 1
    else                       // x is on the left side
        high = mid - 1
```

Recursive Method

```
binarySearch(arr, x, low, high)
    if low > high
        return False
    else
        mid = (low + high) / 2
        if x == arr[mid]
            return mid
        else if x > arr[mid]        // x is on the right side
            return binarySearch(arr, x, mid + 1, high)
        else                               // x is on the left side
            return binarySearch(arr, x, low, mid - 1)
```

CODE:

```
# Binary Search in python (Iterative Method)


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")

-----------------------------------------------------------------
# Binary Search in python (Recursive Method)


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")
```

---

2) Implement Merge Sort

MergeSort Algorithm

```
MergeSort(A, p, r):
    if p > r
        return
    q = (p+r)/2
    mergeSort(A, p, q)
    mergeSort(A, q+1, r)
    merge(A, p, q, r)

-----------------------------------------------------------------
```

Merge Sort Code in Python:

```
# MergeSort in Python
```

```
def mergeSort(array):
    if len(array) > 1:

        #  r is the point where the array is divided into two subarrays
        r = len(array)//2
        L = array[:r]
        M = array[r:]

        # Sort the two halves
        mergeSort(L)
        mergeSort(M)

        i = j = k = 0

        # Until we reach either end of either L or M, pick larger among
        # elements L and M and place them in the correct position at A[p..r]
        while i < len(L) and j < len(M):
            if L[i] < M[j]:
                array[k] = L[i]
                i += 1
            else:
                array[k] = M[j]
                j += 1
            k += 1

        # When we run out of elements in either L or M,
        # pick up the remaining elements and put in A[p..r]
        while i < len(L):
            array[k] = L[i]
            i += 1
            k += 1

        while j < len(M):
            array[k] = M[j]
            j += 1
            k += 1


# Print the array
def printList(array):
    for i in range(len(array)):
        print(array[i], end=" ")
    print()


# Driver program
if __name__ == '__main__':
    array = [6, 5, 12, 10, 9, 1]

    mergeSort(array)

    print("Sorted array is: ")
    printList(array)
```

---

3) Implement Quick Sort

Quick Sort Algorithm

```
quickSort(array, leftmostIndex, rightmostIndex)
  if (leftmostIndex < rightmostIndex)
    pivotIndex <- partition(array,leftmostIndex, rightmostIndex)
    quickSort(array, leftmostIndex, pivotIndex - 1)
    quickSort(array, pivotIndex, rightmostIndex)

partition(array, leftmostIndex, rightmostIndex)
  set rightmostIndex as pivotIndex
  storeIndex <- leftmostIndex - 1
  for i <- leftmostIndex + 1 to rightmostIndex
  if element[i] < pivotElement
    swap element[i] and element[storeIndex]
    storeIndex++
  swap pivotElement and element[storeIndex+1]
return storeIndex + 1
```

Quicksort Code in Python

```
# Quick sort in Python

# function to find the partition position
def partition(array, low, high):

  # choose the rightmost element as pivot
  pivot = array[high]

  # pointer for greater element
  i = low - 1

  # traverse through all elements
  # compare each element with pivot
  for j in range(low, high):
    if array[j] <= pivot:
      # if element smaller than pivot is found
      # swap it with the greater element pointed by i
      i = i + 1

      # swapping element at i with element at j
      (array[i], array[j]) = (array[j], array[i])

  # swap the pivot element with the greater element specified by i
  (array[i + 1], array[high]) = (array[high], array[i + 1])

  # return the position from where partition is done
  return i + 1

# function to perform quicksort
def quickSort(array, low, high):
  if low < high:

    # find pivot element such that
    # element smaller than pivot are on the left
    # element greater than pivot are on the right
    pi = partition(array, low, high)

    # recursive call on the left of pivot
    quickSort(array, low, pi - 1)

    # recursive call on the right of pivot
    quickSort(array, pi + 1, high)


data = [8, 7, 2, 1, 0, 9, 6]
print("Unsorted Array")
print(data)

size = len(data)

quickSort(data, 0, size - 1)

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

4) Implement Insertion Sort

Insertion Sort Algorithm

```
insertionSort(array)
  mark first element as sorted
  for each unsorted element X
    'extract' the element X
    for j <- lastSortedIndex down to 0
      if current element j > X
        move sorted element to the right by 1
    break loop and insert X here
end insertionSort
```

Insertion Sort in Python

```
# Insertion sort in Python


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)
```

____

5) Write a program to sort list of strings (similar to that of dictionary)

```
def dictionary_sort(strings):
    def compare_strings(a, b):
        if a.lower() < b.lower():
            return -1
        elif a.lower() > b.lower():
            return 1
        else:
            return 0

    return sorted(strings, key=lambda s: s.lower(), cmp=compare_strings)

# Example usage
strings = ['apple', 'Orange', 'banana', 'Pineapple']
sorted_strings = dictionary_sort(strings)
print(sorted_strings)
```

____

____