Aryan Nanda **221070003**

Output:

EXPERIMENT NO 8

AIM: Implementation of Insertion Sort and Merge Sort. Performing time complexity comparison with multiple executions with different sizes of Input.

THEORY: Sorting is a fundamental operation in computer science, and various sorting algorithms exist, each with its own strengths and weaknesses. Insertion Sort is a simple algorithm that builds the final sorted array one item at a time, while Merge Sort is a divide-and-conquer algorithm that recursively divides the array into subarrays until they are trivially sorted and then merges them back together.
In this experiment, we will implement both Insertion Sort and Merge Sort algorithms and evaluate their time complexity. Time complexity analysis provides insights into how the algorithms scale with different input sizes, helping us make informed decisions about their practical usage in various scenarios. Lets see both the sorting algorithms in detail.

INSERTION SORT:

Approach:

1. Select an element in each iteration from the unsorted array(using a loop).
2. Place it in its corresponding position in the sorted part and shift the remaining elements accordingly (using an inner loop and swapping).
3. The “inner while loop” basically shifts the elements using swapping.

ALGORITHM:

```
insertionSort(int arr[ ], int n)
{
        for(int i=0; i<n; i++)
        {
                int j = i;
                while(j>0 && arr[ j-1 ]>arr[ j ])
                {
                        swap(arr[ j-1 ] , arr[ j ]);
                        j - -;
                }
        }
}
```

Output:

Example :
Input: N=5, array[] = {5,4,3,2,1}
Output: 1,2,3,4,5
Explanation: After sorting the array is: 1,2,3,4,5

Time complexity: O(N2), (where N = size of the array), for the worst, and average cases.

Reason: If we carefully observe, we can notice that the outer loop, say i, is running from 0 to n-1 i.e. n times, and for each i, the inner loop j runs from i to 1 i.e. i times. For, i = 1, the inner loop runs 1 time, for i = 2, the inner loop runs 2 times, and so on. So, the total steps will be approximately the following: 1 + 2 + 3 +……+ (n-2) + (n-1). The summation is approximately the sum of the first n natural numbers i.e. (n*(n+1))/2. The precise time complexity will be O(n2/2 + n/2). Previously, we have learned that we can ignore the lower values as well as the constant coefficients. So, the time complexity is O(n2). Here the value of n is N i.e. the size of the array.

Best Case Time Complexity: The best case occurs if the given array is already sorted. And if the given array is already sorted, the outer loop will only run and the inner loop will run for 0 times. So, our overall time complexity in the best case will boil down to O(N), where N = size of the array.

MERGE-SORT:

Approach:

1. We will be creating 2 functions mergeSort() and merge().
2. mergeSort(arr[], low, high):
    a. In order to implement merge sort we need to first divide the given array into two halves. Now, if we carefully observe, we need not divide the array and create a separate array, but we will divide the array's range into halves every time until the range size becomes 1.
    b. So, in mergeSort(), we will divide the array around the middle index(*rather than creating a separate array*) by making the recursive call :
        1. mergeSort(arr,low,mid) [*Left half of the array*]
        2. mergeSort(arr,mid+1,high) [*Right half of the array*]

Output:

where low = leftmost index of the array, high = rightmost index of the array, and mid = middle index of the array.

c. Now, in order to complete the recursive function, we need to write the base case as well. We know from step 2, that our recursion ends when the array has only 1 element left. So, the leftmost index, low, and the rightmost index high become the same as they are pointing to a single element.
Base Case: if(low >= high) return;

3. merge(arr[], low, mid, high):
    a. In the merge function, we will use a temp array to store the elements of the two sorted arrays after merging. Here, the range of the left array is low to mid and the range for the right half is mid+1 to high.
    b. Now we will take two pointers left and right, where left starts from low and right starts from mid+1.
    c. Using a while loop(while(left <= mid && right <= high)), we will select two elements, one from each half, and will consider the smallest one among the two. Then, we will insert the smallest element in the temp array.
    d. After that, the left-out elements in both halves will be copied as it is into the temp array.
    e. Now, we will just transfer the elements of the temp array to the range low to high in the original array.

ALGORITHM:

```
mergeSort( arr[ ], low, high)
{
        if( low>=high)
                return;
        mid = (low+high)/2;
        mergeSort(arr[ ], low, mid);
        mergeSort(arr[ ], mid+1, high);
        merge(arr[ ], low, high, mid);
}

merge(arr[ ], low, high, mid)
{
```

Output:

```
        left = low;
        right = mid+1;
        while(left <= mid && right <= high)
        {
                if(arr[left] <=arr[right]
                {
                        print( arr[left] );
                        left++;
                }
                else
                {
                        print( arr[right] );
                        right++;
                }
        }
        while( right <= high)
        {
                print(arr[right] )
                right++;
        }
        while( left <= mid)
        {
                print(arr[left] )
                left++;
        }
}
```

Example :

Input: N=7,arr[]={3,2,8,5,1,4,23}
Output**:** 1,2,3,4,5,8,23

Time complexity: O(nlgn)

Reason: At each step, we divide the whole array, for that lgn and we assume n steps are taken to get sorted array, so overall time complexity will be nlgn

Space complexity: O(n)

Output:

Reason: We are using a temporary array to store elements in sorted order.

Auxiliary Space Complexity: O(n)

Code:-

```
from random import *

from time import *

import matplotlib.pyplot as plt

def create_input(n):

    with open("input.txt","w") as data:

        li =[0]*n

        for i in range(n):

            li[i]=str(randint(1,100))+"\n"

        data.writelines(li)



    with open("input.txt","r") as data:

        l = data.readlines()

        s = list(map(str.strip, l))

        res = [eval(i) for i in s]

    return res

def insertion_sort(lis):

    for i in range(1,len(lis)):

        key = lis[i]

        j=i-1
```

Output:

```
        while j>=0 and lis[j]>key:

            lis[j+1] = lis[j]

            j=j-1

        lis[j+1]=key

    endt =time()

    return lis



def merge(arr, l, m, r):

    n1 = m - l + 1

    n2 = r - m



    L = [0] * (n1)

    R = [0] * (n2)



    for i in range(0, n1):

        L[i] = arr[l + i]



    for j in range(0, n2):

        R[j] = arr[m + 1 + j]



    i = 0

    j = 0
```

Output:

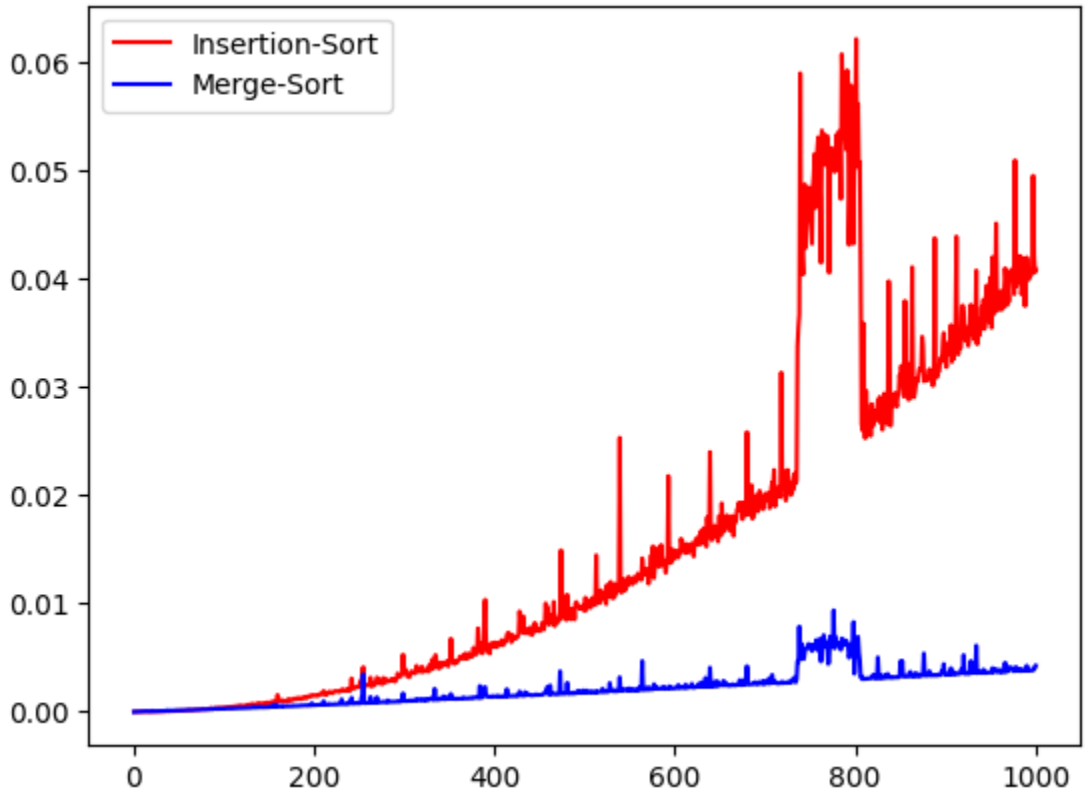
```
    k = l



    while i < n1 and j < n2:

        if L[i] <= R[j]:

            arr[k] = L[i]

            i += 1

        else:

            arr[k] = R[j]

            j += 1

        k += 1



    while i < n1:

        arr[k] = L[i]

        i += 1

        k += 1



    while j < n2:

        arr[k] = R[j]

        j += 1

        k += 1



def mergeSort(arr, l, r):
```

Output:

```
    if l < r:

        m = l+(r-l)//2

        mergeSort(arr, l, m)

        mergeSort(arr, m+1, r)

        merge(arr, l, m, r)

    return arr

time1,time2 = [0]*1000,[0]*1000

x_points = list(range(1, 1001))

for i in range(1,1000):

    inp = create_input(i)

    bgt=time()

    insertion_sort(inp)

    endt=time()

    time1[i]=endt-bgt

    bgt=time()

    mergeSort(inp,0,i-1)

    endt=time()

    time2[i]=endt-bgt

plt.plot(x_points,time1,"r",label="Insertion-Sort")

plt.plot(x_points,time2,"b",label="Merge-Sort")

plt.legend()

plt.show()
```

Aryan Nanda **221070003**

Output:



Conclusion:

We have implemented the merge sort and insertion sort algorithm and found its time complexity for a desired number of elements in an array. We have compared the time complexity and found its graph using the matplotlib module.