**WEEK 3**

**AIM:** To perform sorting techniques on an array

**Description**: In this exercise, I made a menu-driven program where I gave the user 5 choices. Two choices are different ways of initializing the array and the other three are different sorting techniques(quick, heap, and merge).

**Variables Used**:

**size**: Size of the array

**arr**: name of the array

**choice**: choice-value entered by the user

**l, mid, r, i, j, idx, value:** helper variables in merge sort

**largest, left, right, i:** helper variables in heap sort

**idx, i, j, l, h, pivot:** helper variables in quick sort

Three sorting techniques I performed on the array are explained below in detail

1. **QuickSort(Randomized):**

partition(arr[], l, h)

pivot = arr[h]

i = l

for j = l to h – 1 do

if arr[j] <= pivot then

swap arr[i] with arr[j]

i = i + 1

swap arr[i] with arr[h] //placing pivot at its correct pos

return i

partition\_random(arr[], l, h)

idx = Random Number from l to h // selecting random index

Swap arr[idx] and arr[h] // swapping this with lastelement

return partition(arr, l, h)

quicksort(arr[], l, h)

if l >= h then

return

else

pivot = partition\_random(arr, l, h)

quicksort(arr, l , pivot-1)

quicksort(arr, pivot+1, h)

Input: [84 686 446 337 725 107 666 101 258 207 ]

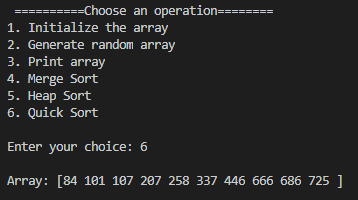
Output: [84 101 107 207 258 337 446 666 686 725 ]

**Worst-case complexity: O(n^2)**

**Average case complexity: O(n log n)**

**Stability: Not stable**

**Storage: In-place**



1. **Heap Sort**

max\_heapify(arr[], size, i)

largest = i

left = 2\*i+1 // left child index

right = 2\*i+2 // right child index

If left < size and arr[left]>arr[largest] then

largest = left

If right < size and arr[right]>arr[largest] then

Largest = right

If i != largest then

swap arr[i] with arr[largest]

call max\_heapify(arr, size, largest)

heap\_sort(arr[], size)

For i = size/2 - 1 to 0 do

Call max\_heapify(arr, size, i)

For i = size-1 to 0 do

Swap arr[i] with arr[0]//placing the root elem at last

Call max\_heapify(arr, i, 0)

Input: [753 896 690 832 322 236 86 147 216 305 ]

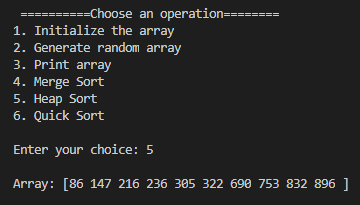
Output: [86 147 216 236 305 322 690 753 832 896 ]

**Worst-case complexity: O(n log n)**

**Average case complexity: O(n log n)**

**Stability: Not stable**

**Sorting method: Internal**



**3) Merge Sort**

merge\_sort(arr[], l, r)

if l >= r

return

mid = (l+r)/2

mergeSort(arr, l, mid)// calling merge sort on left half

mergeSort(arr, mid+1, r)// calling merge sort on right half

merge(arr, l, mid, r)// merging the left and right subarray

merge(arr[], l, mid, r)

size1 = mid - l +1

size2 = r - mid

Create left[size1] and right[size2]

For i = 0 to size1 do

left[i] = arr[l+i]

For j = 0 to size2 do

right[j] = arr[mid+1+i]

i = 0, j = 0, k = l

While i<size1 and j<size2 do

If left[i] <= right[j] then

arr[k] = left[i]

k = k+1

i = i+1

Else

arr[k] = right[j]

k = k +1

j = j +1

While i < size do // element (if any) of left to arr

arr[k] = left[i]

k = k+1

i = i+1

While j < size2 do // element (if any) of right to arr

arr[k] = right[j]

k = k +1

j = j +1

Input: [783 813 762 428 560 390 206 952 967 905 ]

Output: [206 390 428 560 762 783 813 905 952 967 ]

**Worst-case complexity: O(n log n)**

**Average case complexity: O(n log n)**

**Stability: Stable**

**Storage: not In-place**

