K. J. Somaiya College of Engineering

(A Constituent College of Somaiya Vidyavihar University)

Department of Computer Engineering

Batch: A1 Roll No.: 16010120015

Experiment No.3

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Implementation of Quick sort/Merge sort algorithm

Objective: To learn the divide and conquer strategy of solving the problems of different types

CO to be achieved:

CO 2 Describe various algorithm design strategies to solve different problems and analyze Complexity.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://en.wikipedia.org/wiki/Quicksort**
4. **https://www.cs.auckland.ac.nz/~jmor159/PLDS210/qsort.html**
5. **http://www.cs.rochester.edu/~gildea/csc282/slides/C07-quicksort.pdf**
6. **http://www.sorting-algorithms.com/quick-sort**
7. **http://www.cse.ust.hk/~dekai/271/notes/L01a/quickSort.pdf**
8. **http://en.wikipedia.org/wiki/Merge_sort**
9. **http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/mergeSort.htm**
10. **http://www.sorting-algorithms.com/merge-sort**
11. **http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Merge_sort.html**

Pre Lab/ Prior Concepts:
Data structures, various sorting techniques

Historical Profile:
Quicksort and merge sort are s a divide-and-conquer sorting algorithm in which division is dynamically carried out. They are one the most efficient sorting algorithms.

New Concepts to be learned:
Number of comparisons, Application of algorithmic design strategy to any problem, Classical problem solving Vs Divide-and-Conquer problem solving.

Algorithm Recursive Quick Sort:

void quicksort(Integer A[ ], Integer left, Integer right)
//sorts A[left.. right] by using partition() to partition A[left.. right], and then //calling itself // twice to sort the two subarrays.
{ **IF** (left < right) then
{ q = partition(A, left, right);
quicksort(A, left, q–1);
quicksort(A, q+1, right);
}
}

Integer *partition(integer* AT**[], Integer *left*, Integer *right*)**
//This function rarranges *A*[*left*..*right*] and finds and returns an integer *q*, such that *A*[*left*], ...,
//*A*[*q*–1] <~ *pivot*, *A*[*q*] = *pivot*, *A*[*q*+1], ..., *A*[*right*] > *pivot*, where *pivot* is the first element of //a[left..right], before partitioning.
{
pivot = A[left]; lo = left+1; hi = right;
WHILE (lo ≤ hi)
{ **WHILE** (A[hi] > pivot) hi = hi – 1;
WHILE (lo ≤ hi and A[lo] <~pivot) lo = lo + 1;
IF (lo ≤ hi) then swap(A[lo], A[hi]);
}
swap(pivot, A[hi]);
RETURN hi;
}

Quick Sort Code:

```
def partition(l, r, arr):
    pivot_index = l
    pivot = arr[pivot_index]
    while l < r:

        while l < len(arr) and arr[l] <= pivot:
            l += 1
        while arr[r] > pivot:
            r -= 1
        if (l < r):
```

```
            arr[l], arr[r] = arr[r], arr[l]
    arr[r], arr[pivot_index] = arr[pivot_index], arr[r]
    return r

def QuickSort(l, r, arr):
    if (l < r):
        p = partition(l, r, arr)
        QuickSort(l, p - 1, arr)
        QuickSort(p + 1, r, arr)


n = int(input())
numbers=[None]*n
for i in range(0,n):
    x= int(input())
    numbers[i]=x

QuickSort(0, n - 1, numbers)


print(f'The sorted array is: {numbers}')
```

Output:

```
6
95
238
682
1295
35
677
The sorted array is: [35, 95, 238, 677, 682, 1295]

Process finished with exit code 0
```

The space complexity of Quick Sort:

O(log n)

Derivation of best case and worst case time complexity (Quick Sort)

Algorithm Merge Sort
MERGE-SORT (*A*, *p*, *r*)

// To sort the entire sequence A[1 .. n], make the initial call to the procedure MERGE-SORT (*A*, //1, *n*). Array *A* and indices *p*, *q*, *r* such that $p \leq q \leq r$ and sub array *A*[*p* .. *q*] is sorted and sub array //*A*[*q* + 1 .. *r*] is sorted. By restrictions on *p*, *q*, *r*, neither sub array is empty.
//OUTPUT: The two sub arrays are merged into a single sorted sub array in *A*[*p* .. *r*].

```
IF p < r                                  // Check for base case
  THEN q = FLOOR[(p + r)/2]                 // Divide step
       MERGE (A, p, q)                     // Conquer step.
       MERGE (A, q + 1, r)                 // Conquer step.
       MERGE (A, p, q, r)                  // Conquer step.
```

```
MERGE (A, p, q, r )
{
    n1 ← q − p + 1
    n2 ← r − q
    Create arrays L[1 . . n1 + 1] and R[1 . . n2 + 1]
    FOR i ← 1 TO n1
        DO L[i] ← A[p + i − 1]
     FOR j ← 1 TO n2
        DO R[j] ← A[q + j ]
    L[n1 + 1] ← ∞
    R[n2 + 1] ← ∞
   i ← 1
   j ← 1
   FOR k ← p TO r
       DO IF L[i ] ≤ R[ j]
           THEN A[k] ← L[i]
                i ← i + 1
           ELSE A[k] ← R[j]
               j ← j + 1
```

}

Code:

```
def mergeSort(a):
   if len(a) > 1:
      mid = len(a) // 2
      l = a[:mid]
      r = a[mid:]
      mergeSort(l)
      mergeSort(r)
      i = j = k = 0
      while i < len(l) and j < len(r):
         if l[i] < r[j]:
            a[k] = l[i]
            i += 1
         else:
            a[k] = r[j]
            j += 1
         k += 1
      while i < len(l):
         a[k] = l[i]
         i += 1
         k += 1
      while j < len(r):
         a[k] = r[j]
         j += 1
         k += 1

def printList(arr):
   for i in range(len(arr)):
      print(arr[i], end=" ")
   print()


n = int(input())
numbers= [None]*n
for i in range(n):
 x= int(input())
 numbers[i]=x

mergeSort(numbers)
```

```
print("Sorted array is: \n")
printList(numbers)
```

Output:

```
6
2354
8964
359
23549
1458
98
Sorted array is:
98 359 1458 2354 8964 23549

Process finished with exit code 0
```

The space complexity of Merge sort:

O(n)

Derivation of best case and worst case time complexity (Merge Sort)

Best Case : The best case is when the partition process always picks the middle element as the first.

$\therefore T(n) = 2T(n/2) + \Theta(n)$
which is equal to $\Theta(n \log n)$

Worst Case : The worst case is when the partition process always picks the greatest smallest element as first.

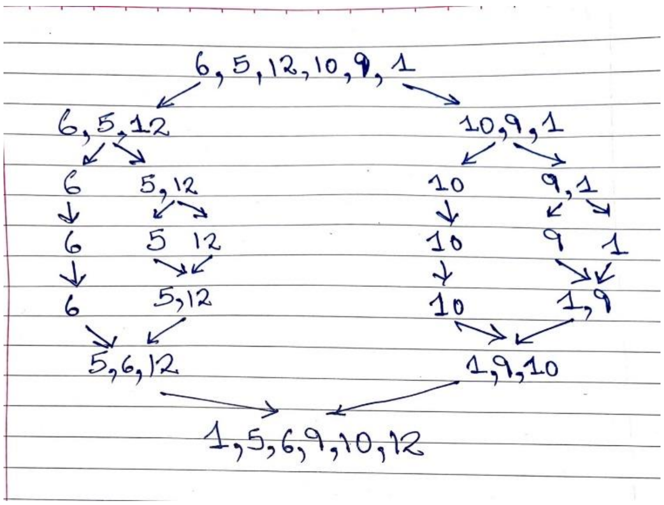
$T(n) = T(0) + T(n-1) + \Theta(n)$
$= T(n-1) + \Theta(n)$
which is equal to $\Theta(n^2)$

* MERGE SORT

$T(n) = 2T(n/2) + O(n)$
$= O(n \log n)$

Since list of size N is divided into max log n parts and merging takes O(N) time. The time complexity is O (nlogn) for all cases since the algorithm always divides the array into two halves and takes linear time to Merge two halves.

Example for quicksort/Merge tree for merge sort:



Conclusion:

By performing this experiment we understood the concept and working of the two sorting methods namely merge sort and quick sort and calculated their space and time complexities.