**Batch: C3 Roll No.: 16010123217**

**Experiment No. : 3**

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

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title: Study, Implementation, and Comparative Analysis of Merge Sort and Quick Sort.** |



**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** 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 A*T[], 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;

}

**Code (Quick Sort) :**

#include <bits/stdc++.h>

using namespace std;

#define int long long

#define endl "\n"

const int MOD = 1e9 + 7;

const int INF = LLONG\_MAX >> 1;

void quicksort(vector<int> &arr, int bigger, int smaller) {

    if (bigger >= smaller) return;

    int pivot = arr[bigger];

    int i = bigger + 1, j = smaller;

    while (i <= j) {

        while (i <= j && arr[i] <= pivot) i++;

        while (i <= j && arr[j] > pivot) j--;

        if (i < j) {

            swap(arr[i], arr[j]);

        }

    }

    swap(arr[bigger], arr[j]);

    quicksort(arr, bigger, j - 1);

    quicksort(arr, j + 1, smaller);

}

signed main() {

    ios::sync\_with\_stdio(false); cin.tie(NULL);

    int n;

    cin >> n;

    vector<int> arr(n);

    for (int i = 0; i < n; i++) cin >> arr[i];

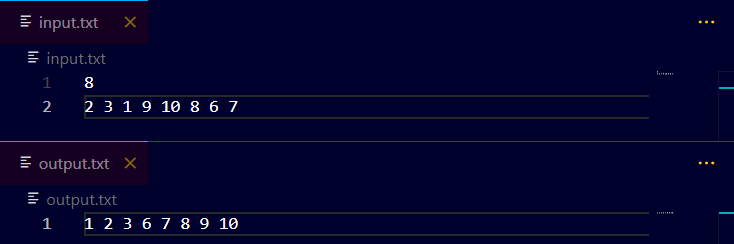
    quicksort(arr, 0, n - 1);

    for (int i = 0; i < n; i++) cout << arr[i] << " ";

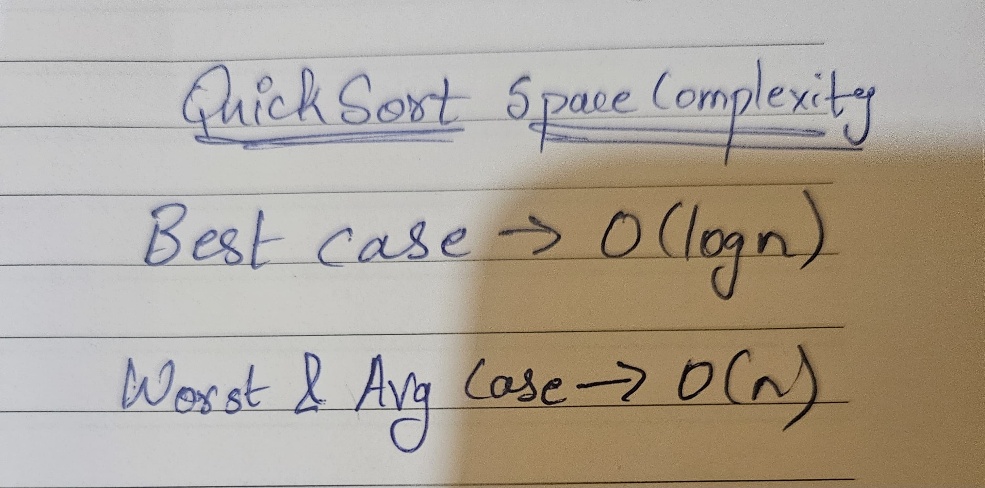
    return 0;

}

**Output:**

****

**The space complexity of Quick Sort:**



**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* ≤ *q* ≤ 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*)

{

*n*1 ← *q* − *p* + 1  
      *n*2 ← *r* − *q*  
      Create arrays L[1 . . *n*1 + 1] and R[1 . . *n*2 + 1]  
      **FOR** *i* ← 1 **TO** *n*1  
            **DO** L[*i*] ← A[*p* + *i* − 1]  
      **FOR** *j* ← 1 **TO** *n*2  
            **DO** R[*j*] ← A[*q* + *j* ]  
      L[*n*1 + 1] ← ∞  
      R[*n*2 + 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 (MergeSort) :**

#include <bits/stdc++.h>

using namespace std;

#define int long long

#define endl "\n"

const int MOD = 1e9 + 7;

const int INF = LLONG\_MAX >> 1;

void mergesort(vector<int> &arr, int start, int end ){

    if(start >= end) return;

    int mid = start + (end - start) / 2;

    mergesort(arr, start, mid);

    mergesort(arr, mid+1, end);

    vector<int> temp(end-start+1);

    int i = start, j = mid+1, k = 0;

    while(i <= mid && j <= end){

        if(arr[i] < arr[j]) temp[k++] = arr[i++];

        else temp[k++] = arr[j++];

    }

    while(i <= mid) temp[k++] = arr[i++];

    while(j <= end) temp[k++] = arr[j++];

    for(int i = start; i <= end; i++) arr[i] = temp[i-start];

}

signed main(){

    ios::sync\_with\_stdio(false); cin.tie(NULL);

    int n;   cin >> n;

    vector<int> v(n);

    for (int i = 0; i < n; i++) cin >> v[i];

    mergesort(v, 0, n-1);

    for(int i = 0; i < n; i++) cout << v[i] << " ";

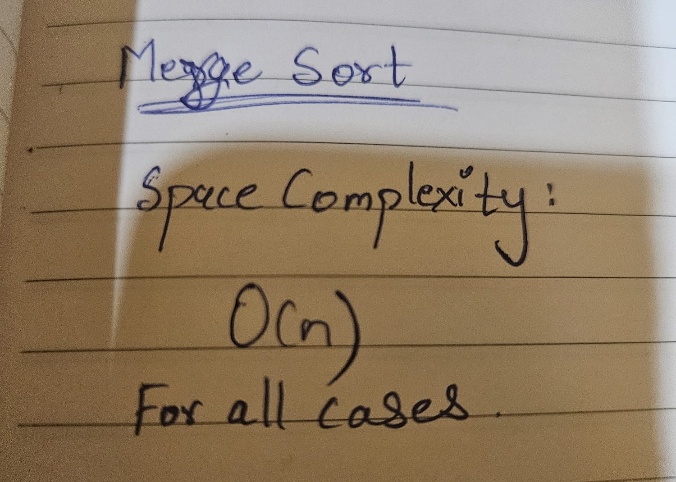
    return 0;

}

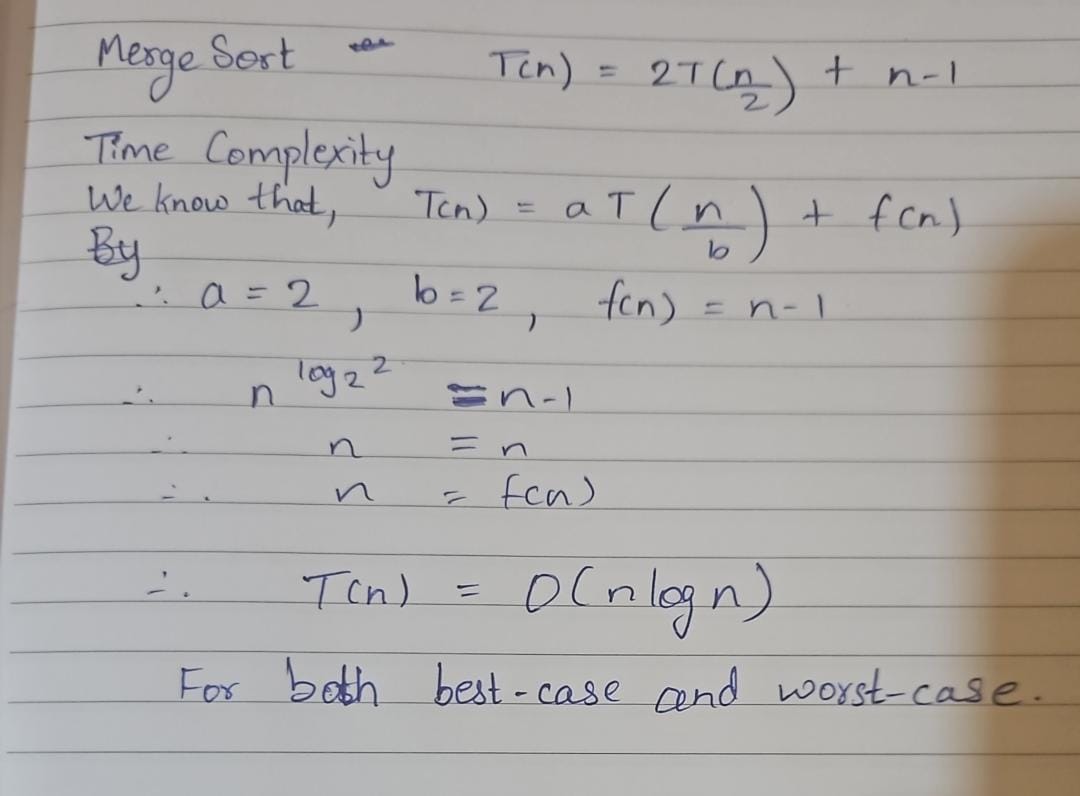
**Output:**

****

**The space complexity of Merge sort:**

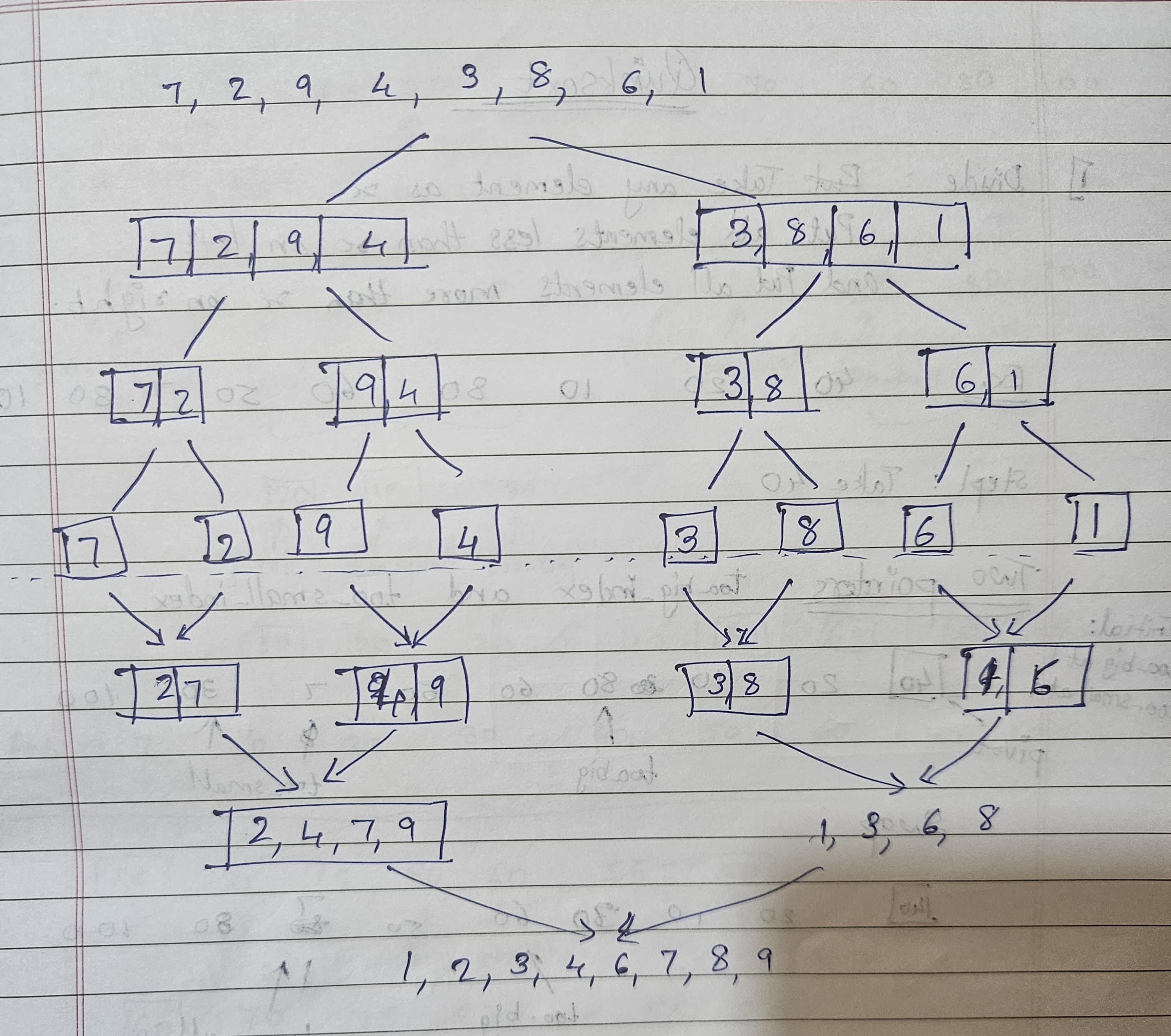


**Derivation of best case and worst-case time complexity (Merge Sort)**

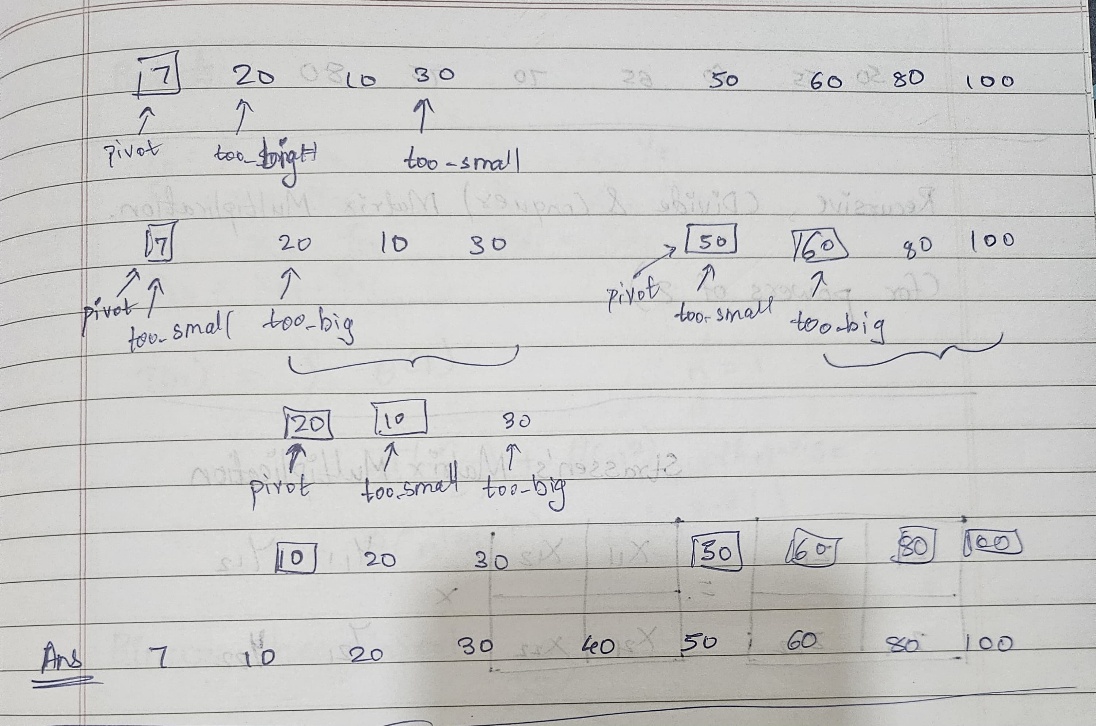
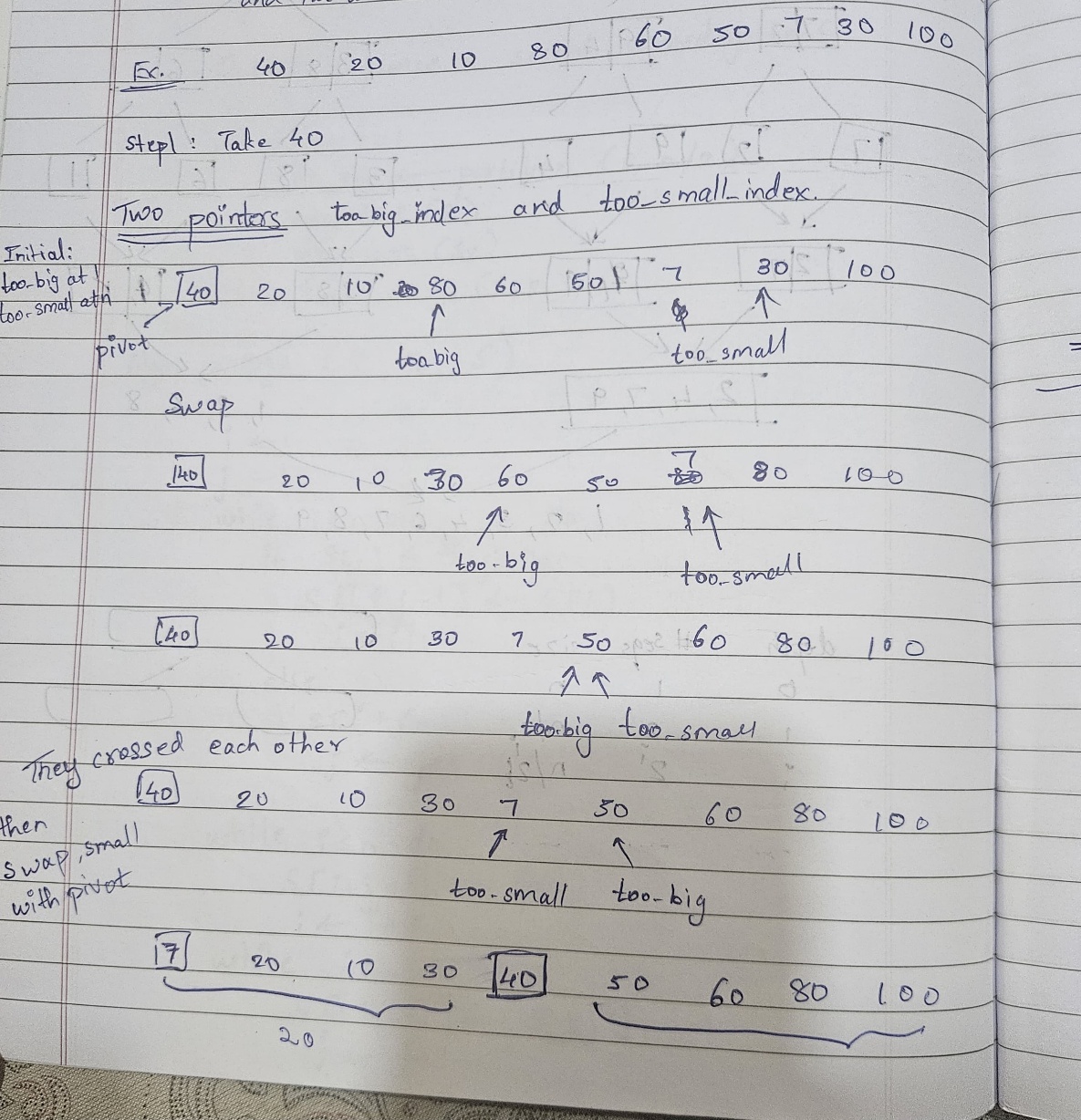


**Example for quicksort/Merge tree for merge sort:**

**Merge Sort**



**Quick Sort**



**CONCLUSION:**

We learnt about sorting algorithms like merge sort and quick sort and their time and space complexities