**DAA ASSIGNMENT-5**

**GROUP MEMBERS:**

**1) U19CS008- KRINA PATEL**

**2) U19CS009- BRIJESH ROHIT**

**3) U19CS015- DHRUV GANDHI**

**4) U19CS038- SUMIT SHETTY**

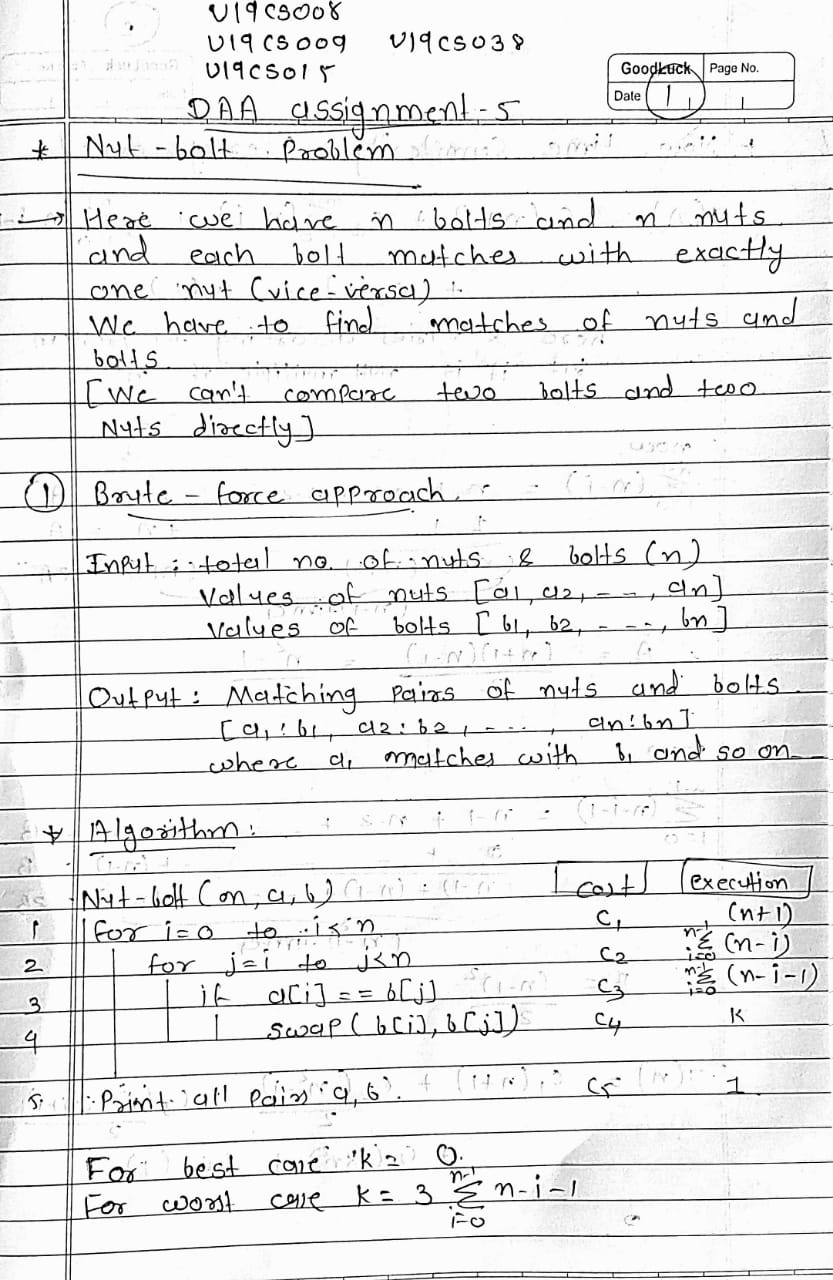
* 1. **(T) Find a computational problem that you can solve using the divide and conquer approach. The problem should be different than the problems discussed in class, and it should be unique and interesting.**

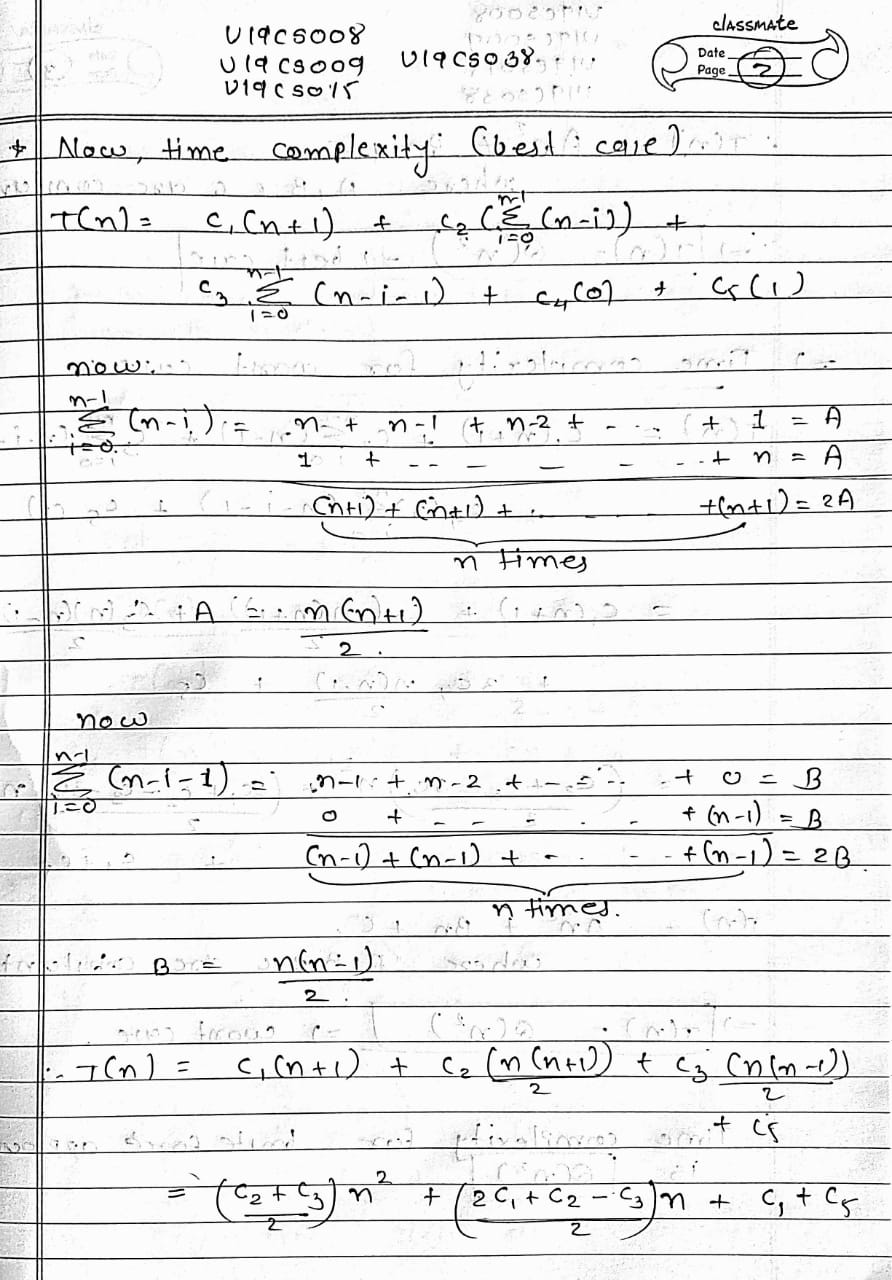
**PROBLEM STATEMENT:**

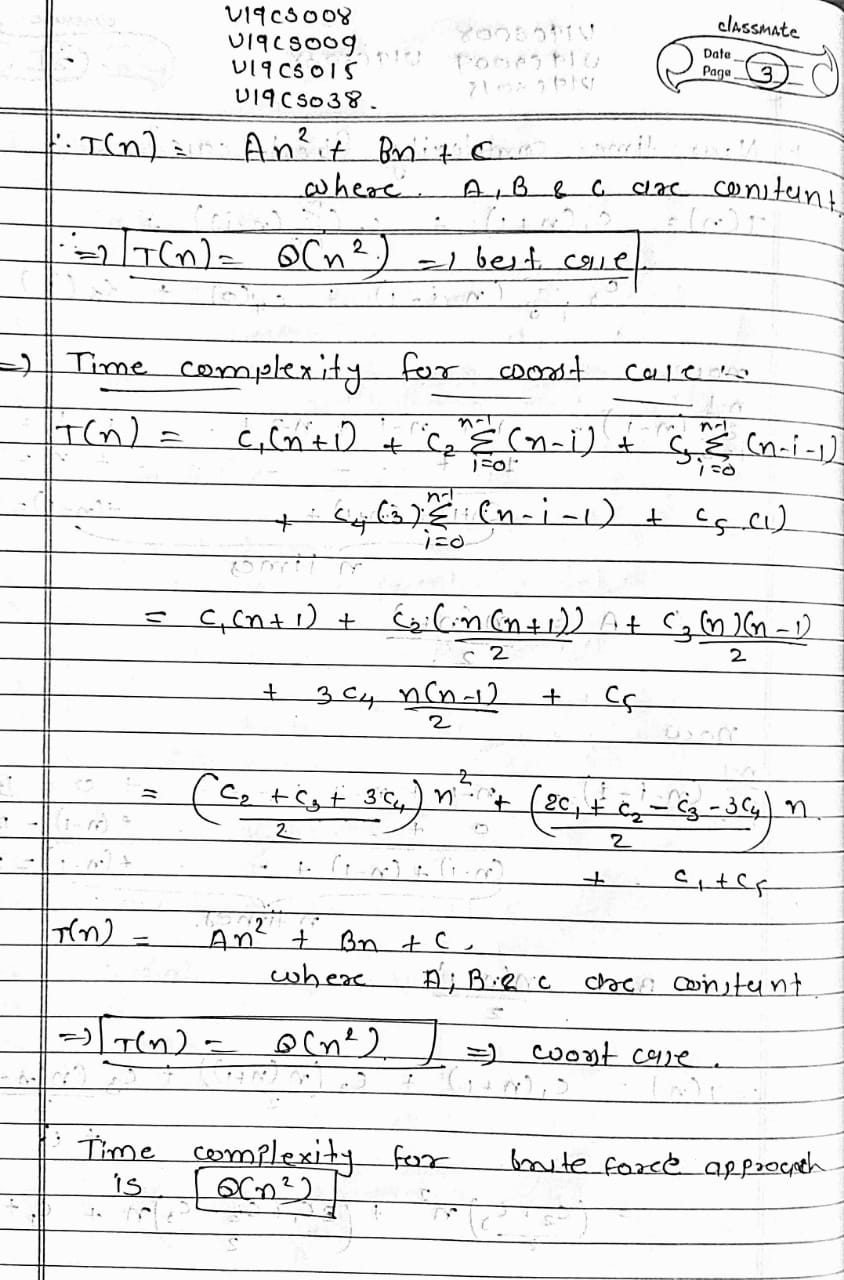
**nut\_bolt\_problem**

**We are given a box which contains bolts and nuts. Assume there are n nuts and n bolts and that each nut matches exactly one bolt (and vice versa). By trying to match a bolt and a nut we can see which one is bigger, but we cannot compare two bolts or two nuts directly. Design an efficient algorithm for matching the nuts and bolts.**

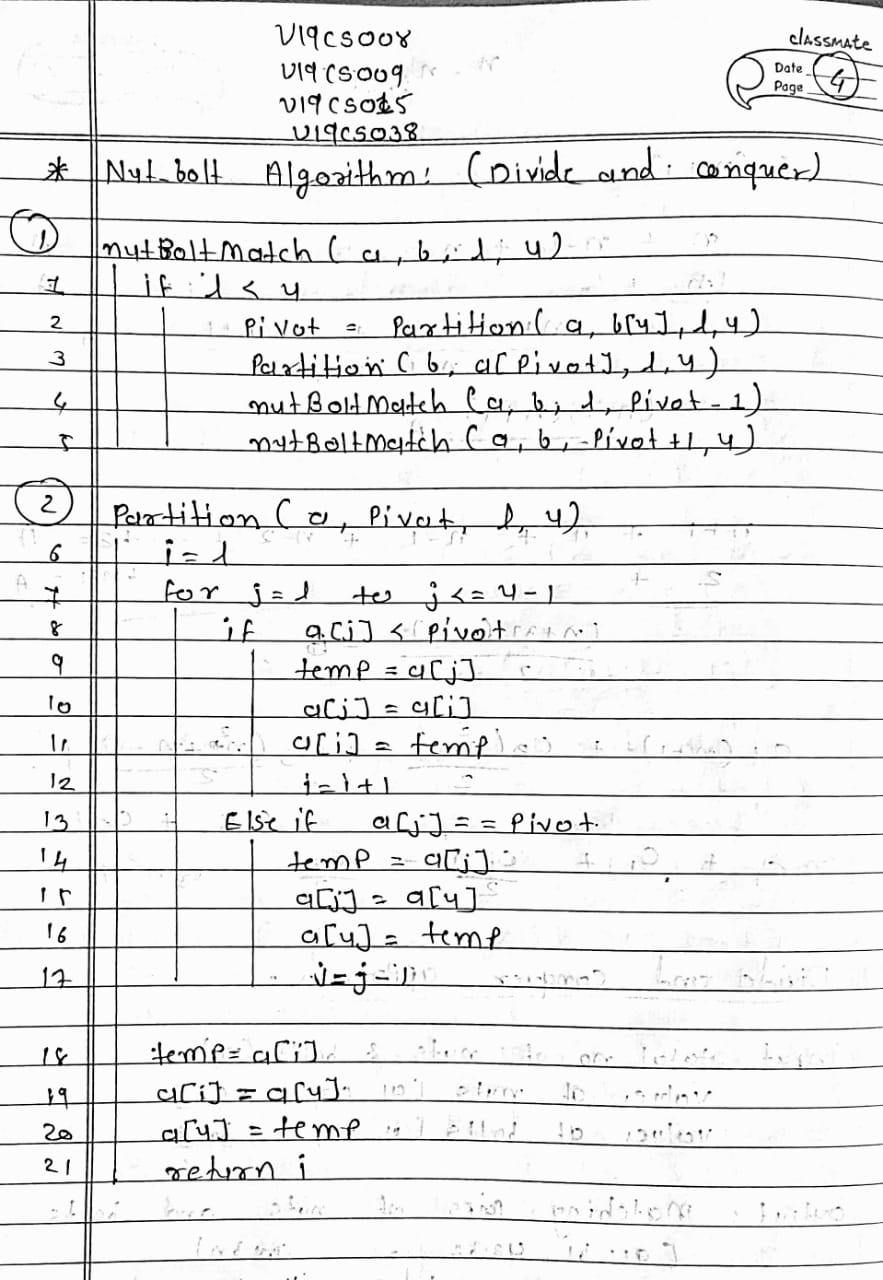
* 1. **(T) Write pseudocodes to design algorithms for the above mentioned computational problem using the brute-force approach (incremental approach) and the divide and conquer approach.**



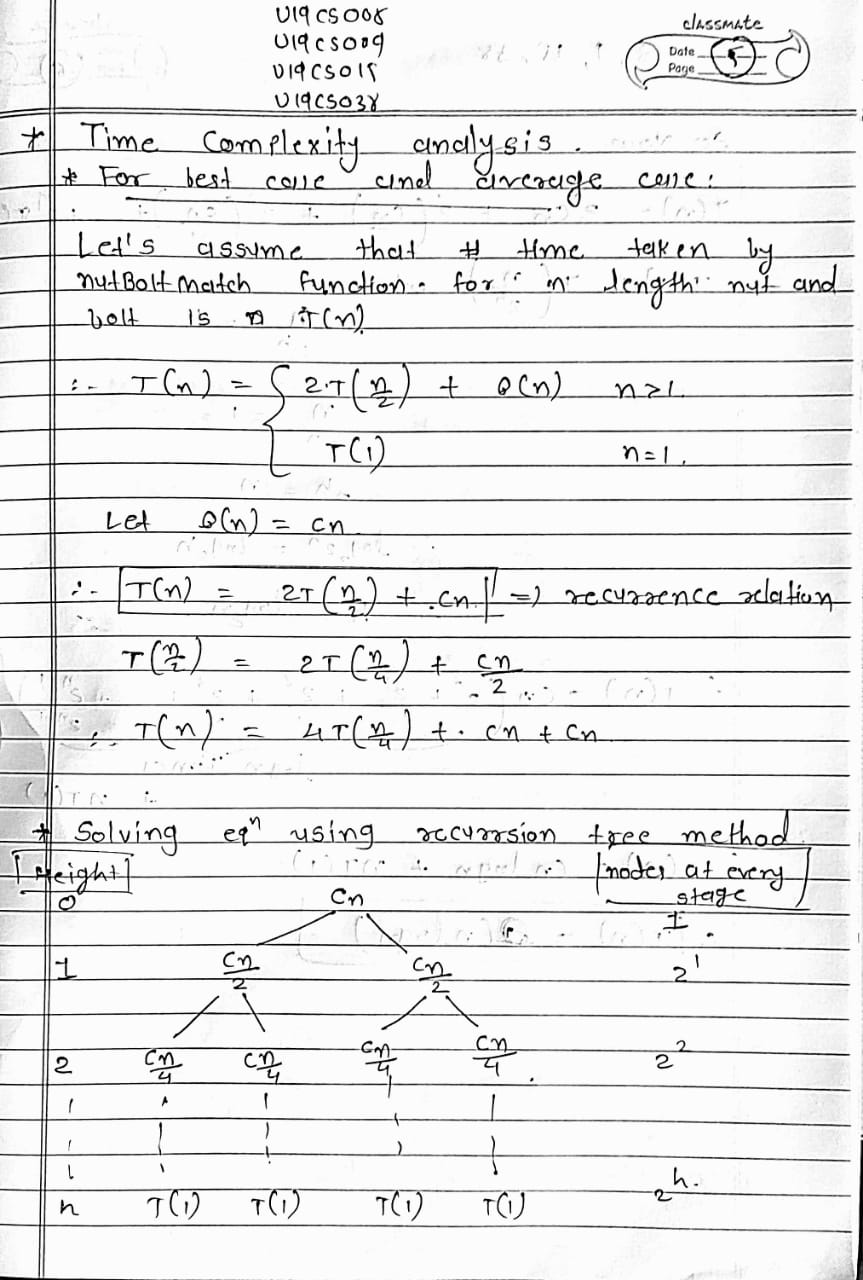


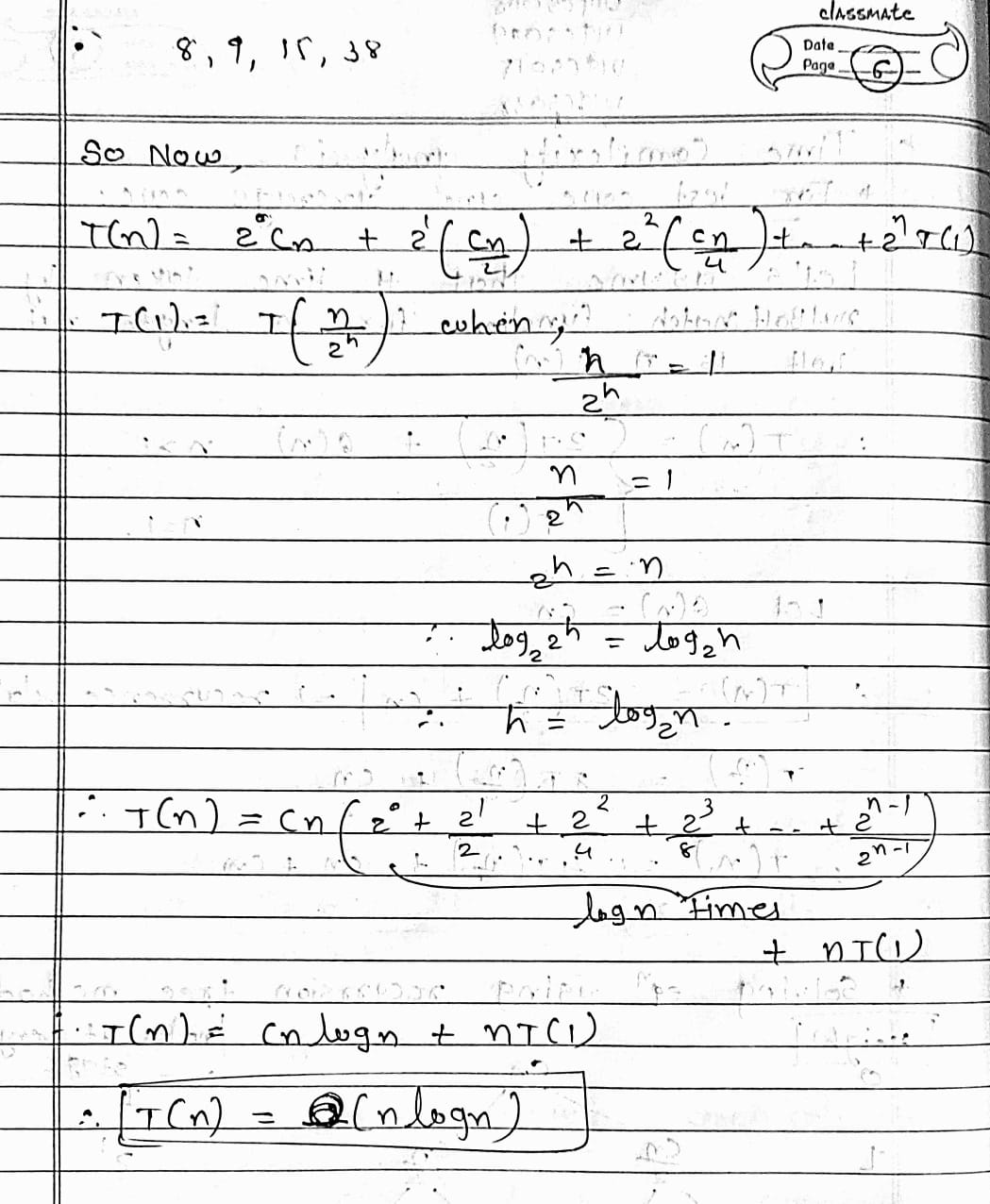


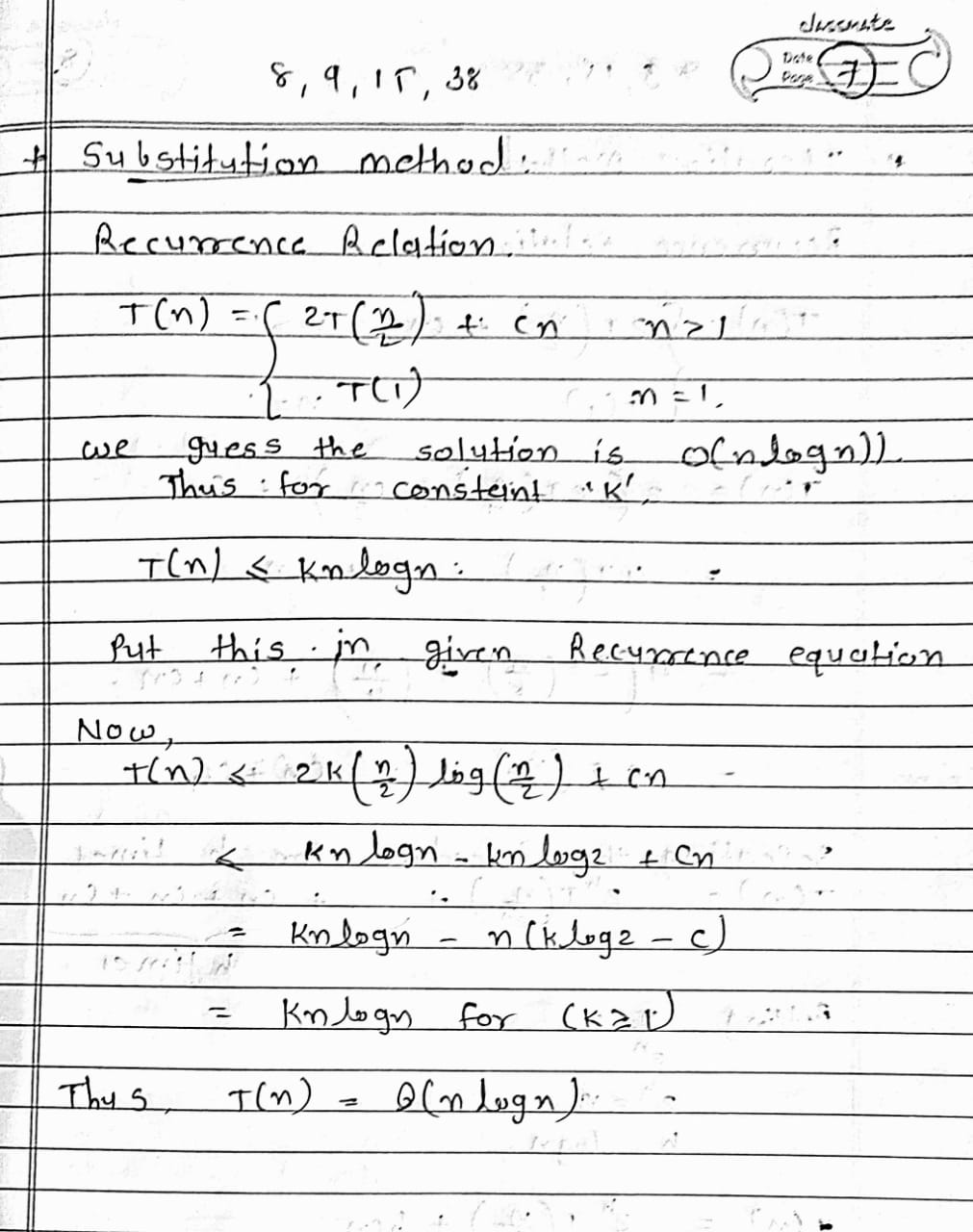
**DIVIDE AND CONQUER:**

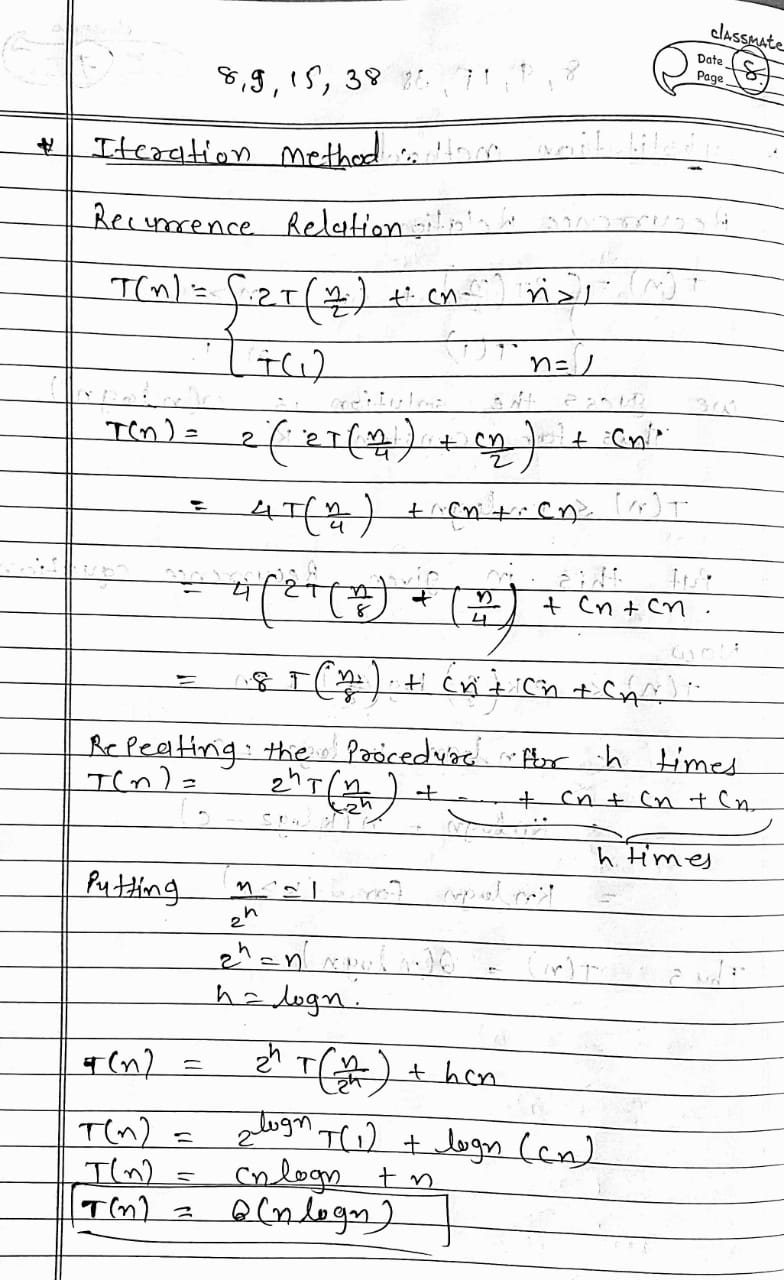


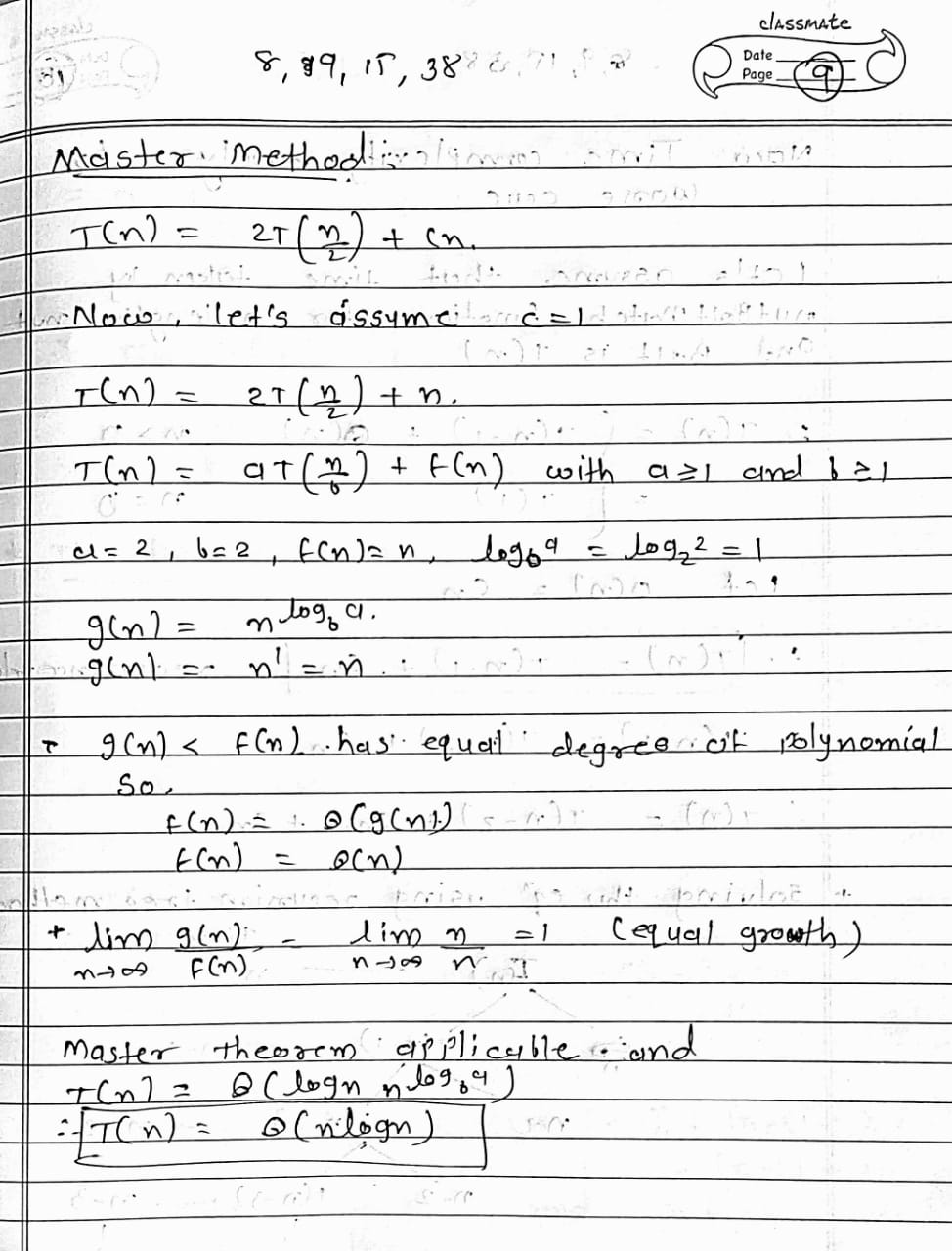
* 1. **(T) Analyze the time complexity of above algorithms. Analyze the divide and conquer algorithm using different methods such as (1) Recursion Tree Method (2) Iterative Method (3) Master Method (4) Substitution Method. Try to analyze the algorithm using as many methods as discussed above.**

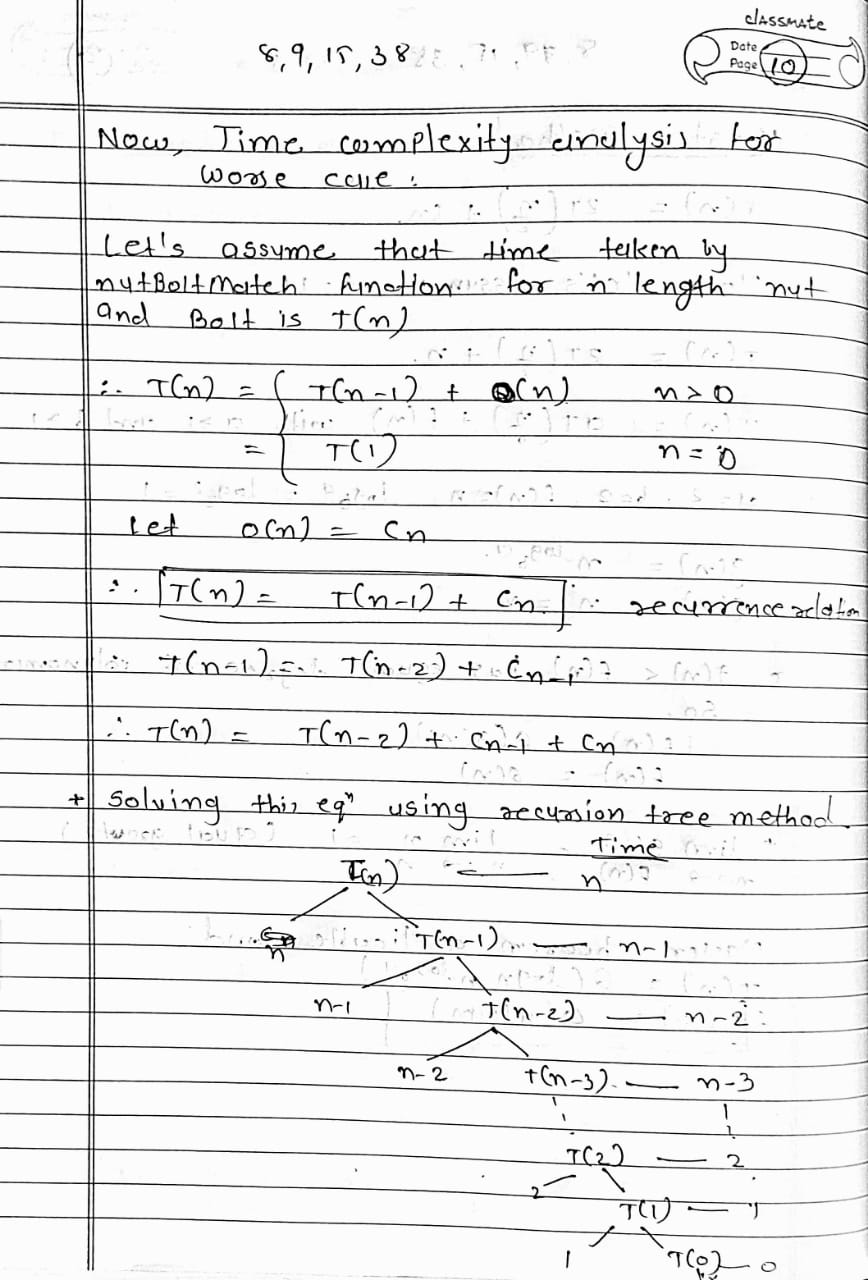


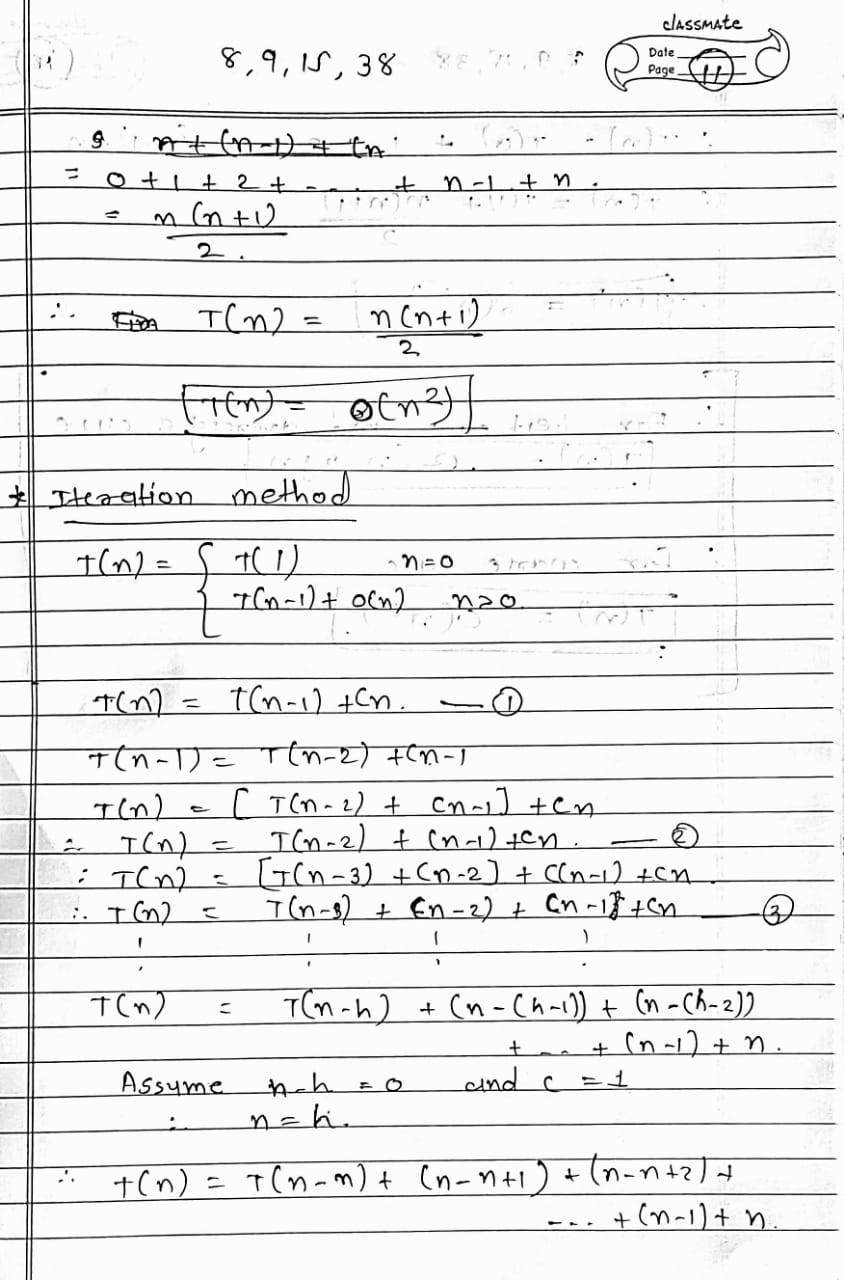


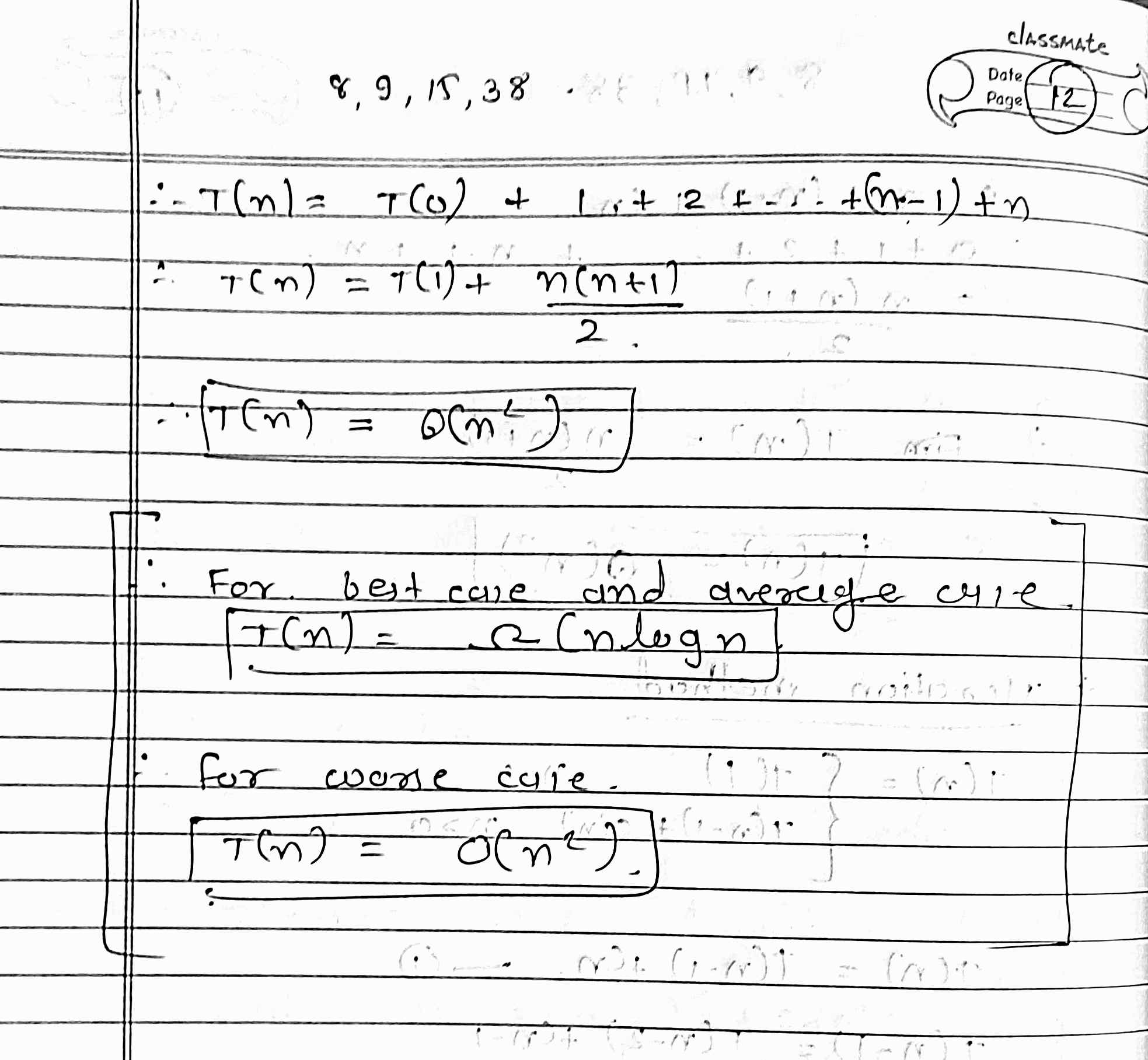




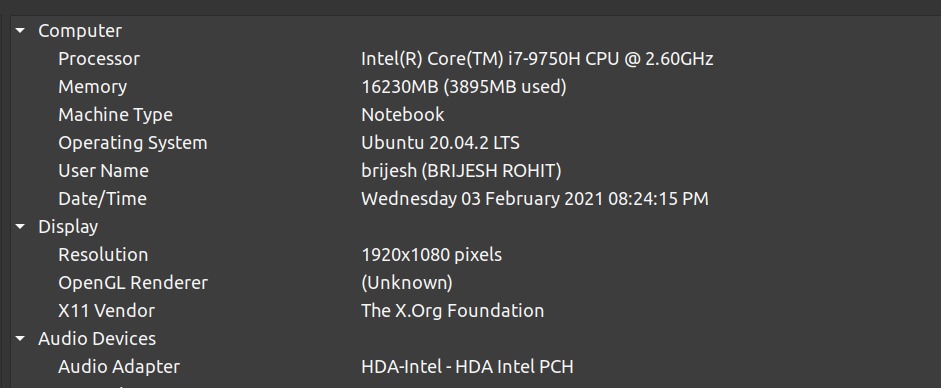








* 1. **(L) Provide the details of Hardware/Software you used to implement algorithms and to measure the time.**



* 1. **(L) Implement the above algorithms and submit the code (complete programs).**

**Brute Force method:**

#include <bits/stdc++.h>

using namespace std;

void nutBoltMatch(vector<int> &a,vector<int> &b,int n)

{

    for(int i=0;i<n;i++) //taking elements from nuts

    {

        for(int j=i;j<n;j++) //taking elements from bolts

        {

            if (a[i]==b[j]) //compareing each nut to each bolt

            {

                swap(b[i],b[j]);

            }

        }

    }

}

int main()

{

    for (int k = 1; k < 7; k++)

    {

        ifstream fin;

        fin.open("InputNutsavg" + to\_string(k) + ".txt");

        vector<int> nuts, bolts;

        while(!fin.eof())

        {

            int temp;

            fin >> temp;

            nuts.push\_back(temp);

        }

        fin.close();

        fin.open("InputBolts"+to\_string(k)+".txt");

        while(!fin.eof())

        {

            int temp;

            fin >> temp;

            bolts.push\_back(temp);

        }

        fin.close();

        int n = nuts.size();

        auto t1 = std::chrono::high\_resolution\_clock::now();

        nutBoltMatch(nuts,bolts,n);

        auto t2 = std::chrono::high\_resolution\_clock::now();

        auto durationavg = std::chrono::duration\_cast<std::chrono::nanoseconds>(t2 - t1).count(); //calculating time

        nuts.clear();

        bolts.clear();

        fin.open("InputNutsworst" + to\_string(k) + ".txt");

        while(!fin.eof())

        {

            int temp;

            fin >> temp;

            nuts.push\_back(temp);

        }

        fin.close();

        fin.open("InputBolts"+to\_string(k)+".txt");

        while(!fin.eof())

        {

            int temp;

            fin >> temp;

            bolts.push\_back(temp);

        }

        fin.close();

        t1 = std::chrono::high\_resolution\_clock::now();

        nutBoltMatch(nuts,bolts,n);

        t2 = std::chrono::high\_resolution\_clock::now();

        auto durationworst = std::chrono::duration\_cast<std::chrono::nanoseconds>(t2 - t1).count(); //calculating time

        ofstream fout;

        fout.open("TimeMeasureBruteForce.csv",ios\_base::app);

        fout << n << ", " << durationavg << ", " << durationworst << "\n";

        fout.close();

    }

    return 0;

}

**Divide and Conquer method:**

#include <bits/stdc++.h>

using namespace std;

#define rep(i, n) for (int i = 0; i < n; i++)

typedef long long ll;

int partition(vector<int> &a, int pivot, int l, int u) // this function divide array into 2 parts and set actual place of pivot element

{

    int j = l;

    for (int i = l; i < u; i++)

    {

        if (a[i] < pivot)

        {

            swap(a[i], a[j]);

            j++;

        }

        if (a[i] == pivot)

        {

            swap(a[i], a[u]);

            i--;

        }

    }

    swap(a[j], a[u]);

    return j;

}

void nutBoltMatch(vector<int> &nuts, vector<int> &bolts, int l, int u)

{

    if (l < u)

    {

        int pivot = partition(nuts, bolts[u], l, u); //taking last element of bolt array as pivot for nuts array

        partition(bolts, nuts[pivot], l, u); //taking pivot element of nut array as pivot for bolt array

        nutBoltMatch(nuts, bolts, l, pivot - 1); //repeating same process for left side of arrays

        nutBoltMatch(nuts, bolts, pivot + 1, u); //repeating same process for right side of arrays

    }

}

int main()

{

    for (int k = 1; k < 6; k++)

    {

        ifstream fin;

        fin.open("InputBolts"+to\_string(k)+".txt");

        vector<int> nuts, bolts;

        int temp;

        while (!fin.eof())

        {

            fin >> temp;

            bolts.push\_back(temp);

        }

        fin.close();

        fin.open("InputNutsavg"+to\_string(k)+".txt");

        while (!fin.eof())

        {

            fin >> temp;

            nuts.push\_back(temp);

        }

        fin.close();

        int n = nuts.size() - 1;

        auto t1 = std::chrono::high\_resolution\_clock::now();

        nutBoltMatch(nuts, bolts, 0, n - 1);

        auto t2 = std::chrono::high\_resolution\_clock::now();

        auto durationavg = std::chrono::duration\_cast<std::chrono::nanoseconds>(t2 - t1).count(); //Mearusing time for avg case files

        fin.open("InputBolts"+to\_string(k)+".txt");

        vector<int> nuts, bolts;

        int temp;

        while (!fin.eof())

        {

            fin >> temp;

            bolts.push\_back(temp);

        }

        fin.close();

        fin.open("InputNutsworst"+to\_string(k)+".txt");

        while (!fin.eof())

        {

            fin >> temp;

            nuts.push\_back(temp);

        }

        fin.close();

        int n = nuts.size() - 1;

        t1 = std::chrono::high\_resolution\_clock::now();

        nutBoltMatch(nuts, bolts, 0, n - 1);

        t2 = std::chrono::high\_resolution\_clock::now();

        auto durationworst = std::chrono::duration\_cast<std::chrono::nanoseconds>(t2 - t1).count(); //Measuring time for worst case files

        ofstream fout;

        fout.open("TimeMeasureDNC.csv",ios\_base::app);

        fout << n+1 << ", " << durationavg << ", " << durationworst << "\n";

        cout << k << " file complete" << endl;

        fout.close();

    }

    return 0;

}

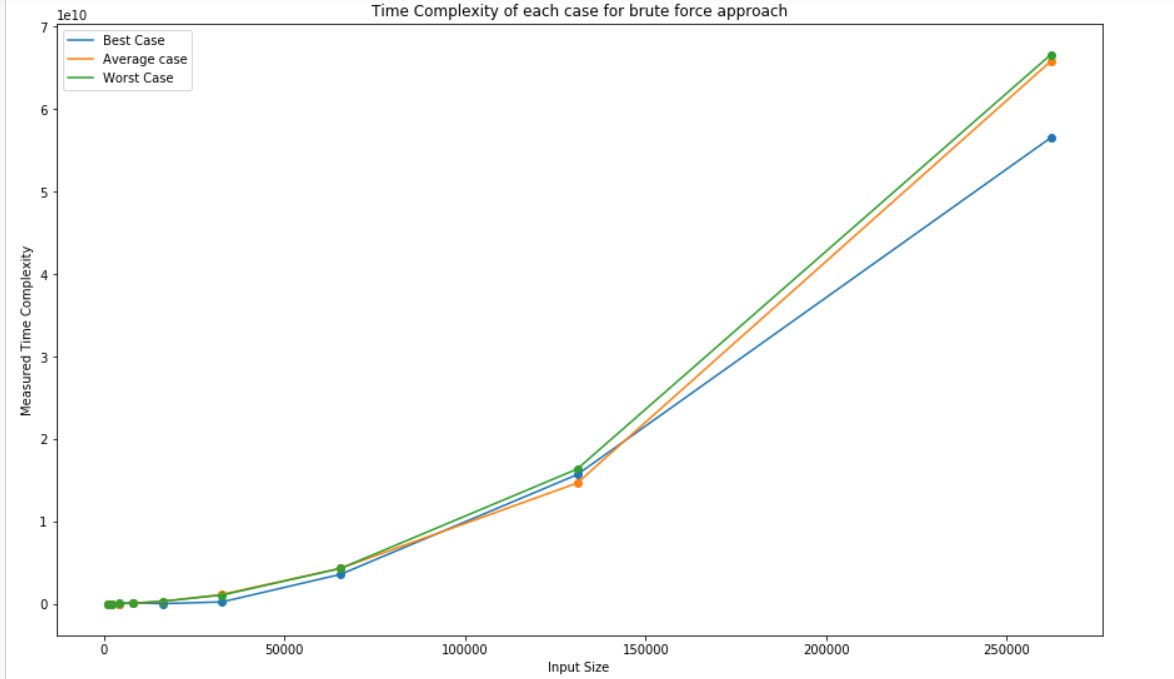
* 1. **(L) Analyze the performance of both the implemented algorithms (performance of algorithms on your computers). Plot a graph.**

**Analysis of Brute Force method:**

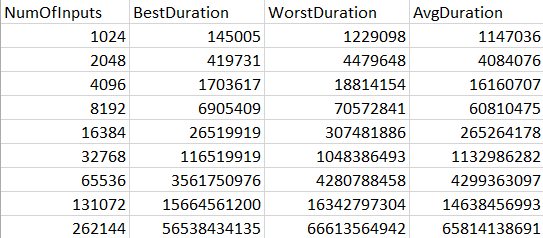
**Solution1\_Brute\_Force\_Approach:** Start with the first bolt and compare it with each nut until we find a match. *In the worst case, we require n comparisons*. Repeat this for successive bolts on all remaining gives **θ(n^2)** complexity.

After analysing graph we can conclude all things mentioned above.

**Graph:**



**Measured time:**



**Analysis of Divide and Conquer method:**

**Solution2\_DivideAndConquer:** This solution is very similar to randomized Quick Sort. For simplicity let us assume that bolts and nuts are represented in two arrays B and N. The algorithm first performs a partition operation as follows:

pick a random bolt B[t]. Using this bolt, rearrange the array of nuts into three groups of elements:

*First the nuts smaller than B[i]*

*Then the nut that matches B[i], and*

*Finally, the nuts larger than B[i].*

Next, using the nut that matches B[i], perform a similar partition on the array of bolts. This pair of partitioning operations can easily be implemented in O(n) time, and it leaves the bolts and nuts nicely partitioned so that *the “pivot” bolt and nut are aligned with each other and all other bolts and nuts are on the correct side of these pivots – smaller nuts and bolts precede the pivots, and larger nuts and bolts follow the pivots.* Our algorithm then completes by recursively applying itself to the subarray to the left and right of the pivot position to match these remaining bolts and nuts. We can assume by induction on n that these recursive calls will properly match the remaining bolts.

To analyze the running time of our algorithm, we can use the same analysis as that of randomized Quick Sort. Therefore, applying the analysis from Quick Sort, the time complexity of our algorithm is **O(nlogn)**.

In above process we are dividing array by element no position we cannot find same division every time so in analysis part we talk about average case and worst case and find upper bound and lower bound of time complexity.

*In best case we assume that every time pivot element is median element and hence every time array divide in half so time complexity will be Ω(nlogn).*

*In worst case we assume that pivot element is largest element of array and hence every time array size reduce by 1 only so time complexity will be O(n2).*

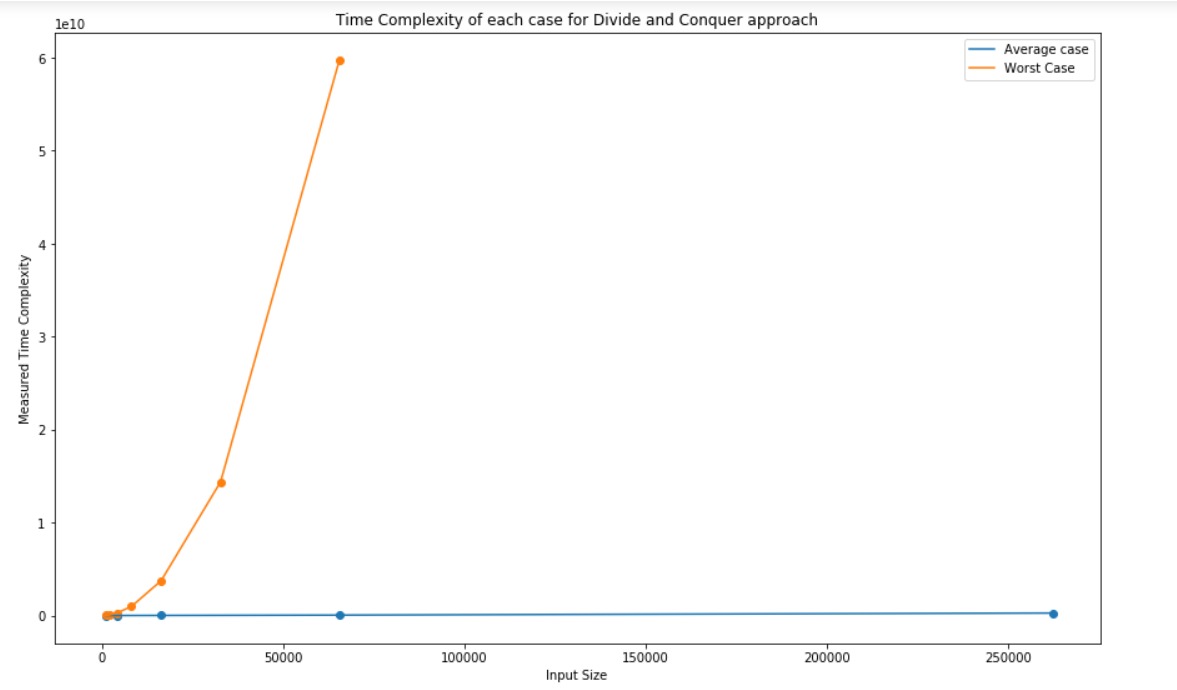
*In average case even if we assume very must unbalance division of array like n/10 and 9n/10 then also we can easily find using master theorem that time complexity will be θ(nlogn).*

Because of that this algorithm gives pretty much advantage in time complexity.

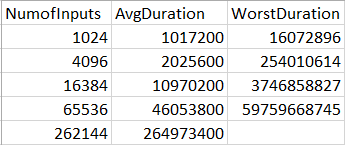
Even by applying this algorithm we are using constant extra space so this algorithm is very good in terms of space complexity too.

By analysing graph we can conclude all things mentioned above.

**Graph:**



**Measured time:**

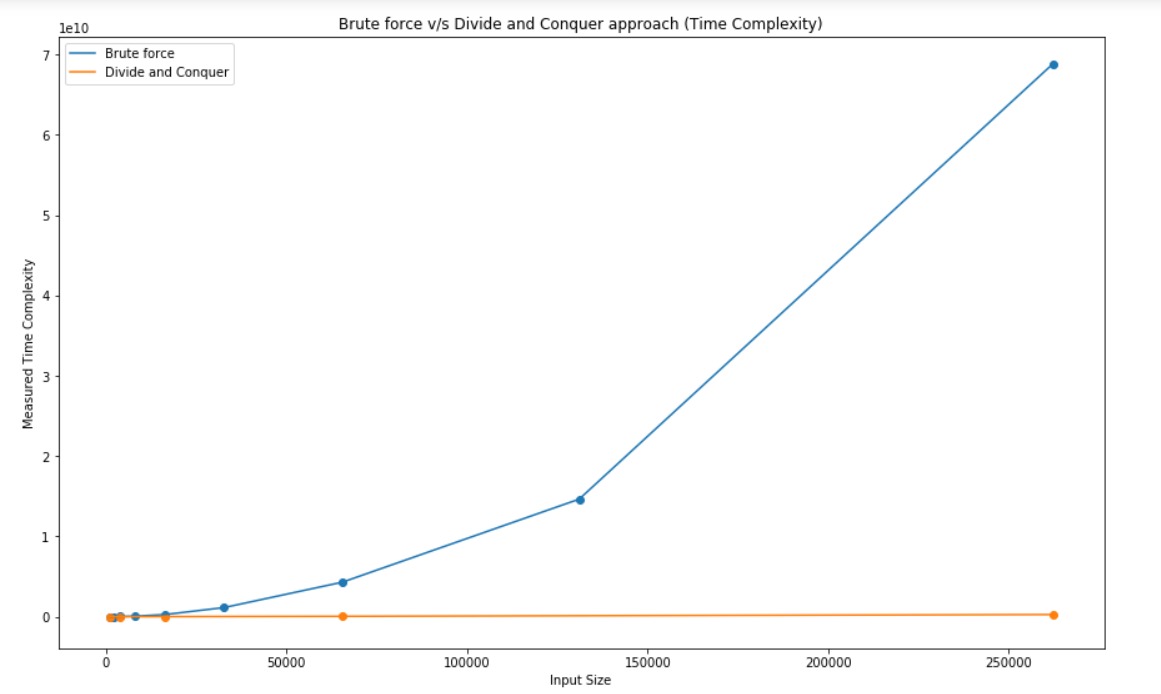


* 1. **(L) Comparatively analyze the performance of above algorithms and plot a graph.**

For best case and average case time complexity of divide and conquer algorithm is Ω(nlogn) and for brute force algorithm is θ(n2).

So time taken by divide and conquer algorithm is significantly less as compare to brute force algorithm.

We can conclude same from graph.



As for the worst case time complexity of brute force and Divide and conquer algorithm i.e. O(n2). But as we compare steps of algorithm time taken by divide and conquer algorithm will be higher than brute force algorithm.

