**Name:** Vishal Shashikant Salvi.

**UID:** 2019230069

**Batch:** C

**Class:** SE Comps

**Experiment No 4**

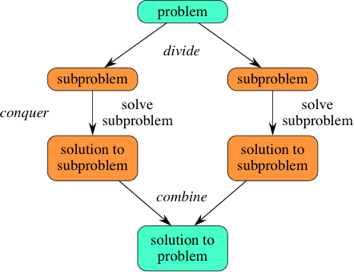
### Aim: Min-Max Using Divide and Conquer

**Theory:**

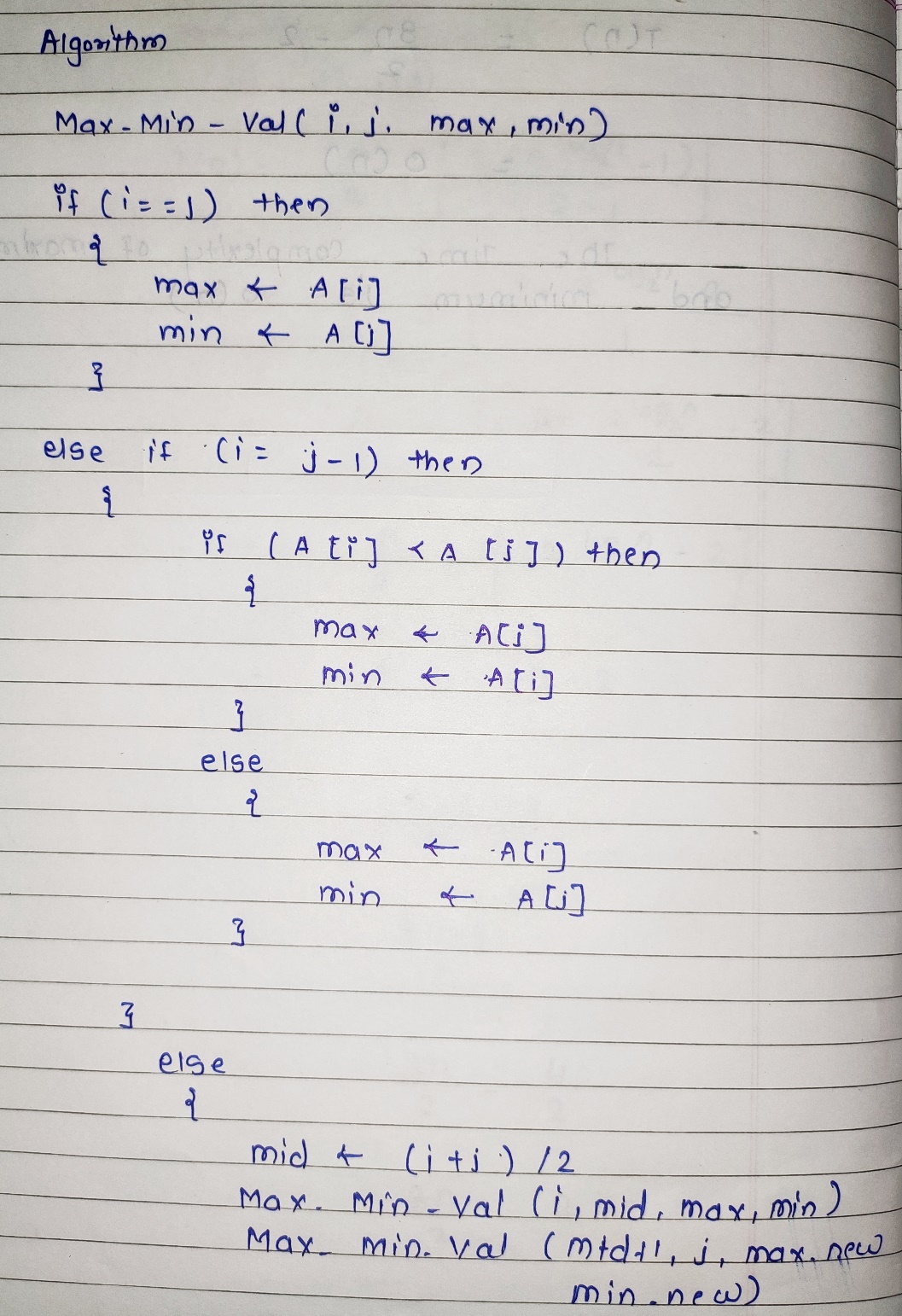
**Discuss divide and Conquer strategy in general.**

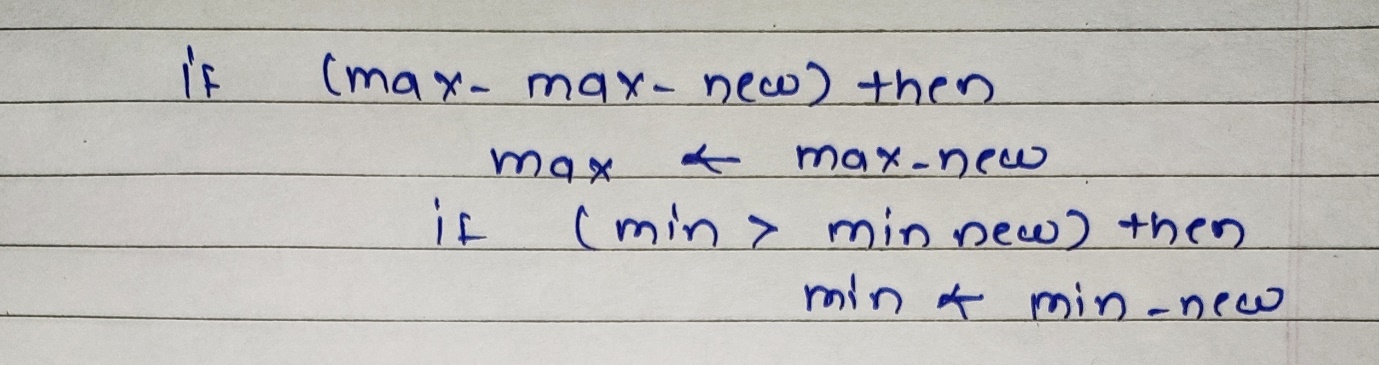
1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem.
2. **Conquer** the subproblems by solving them recursively. If they are small enough, solve the subproblems as base cases.
3. **Combine** the solutions to the subproblems into the solution for the original problem.

You can easily remember the steps of a divide-and-conquer algorithm as *divide, conquer, combine*. Here's how to view one step, assuming that each divide step creates two subproblems (though some divide-and-conquer algorithms create more than two):

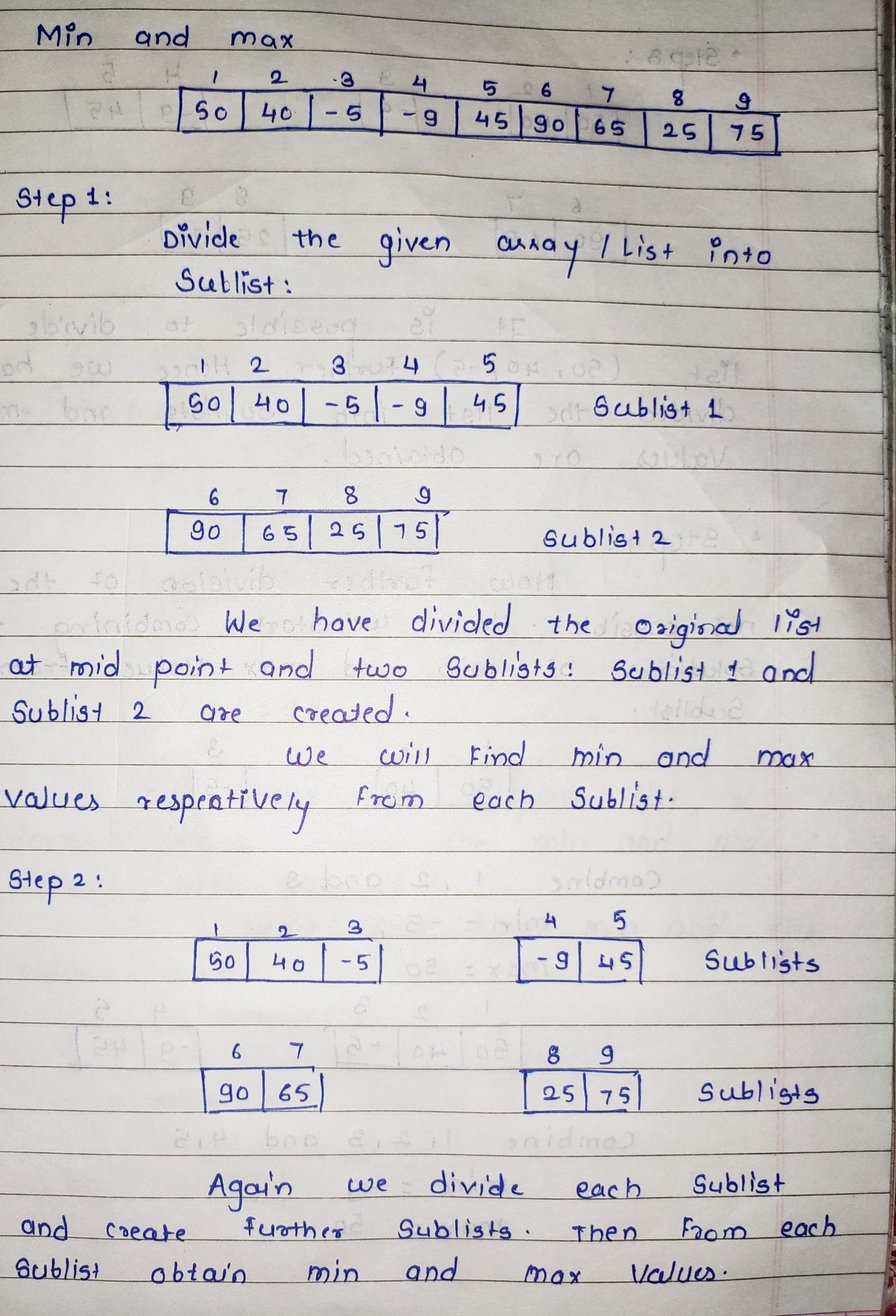


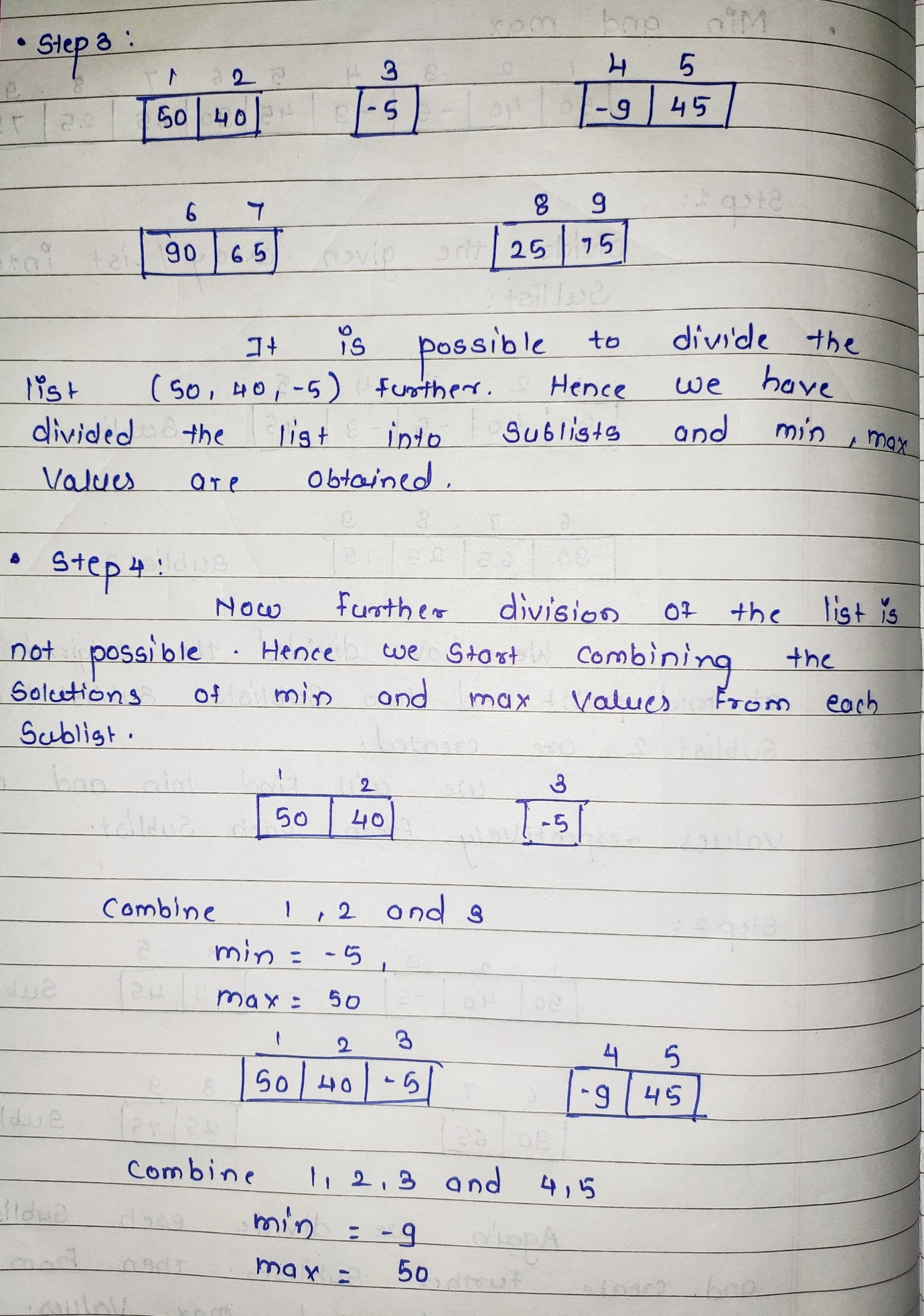
**Algorithm:**

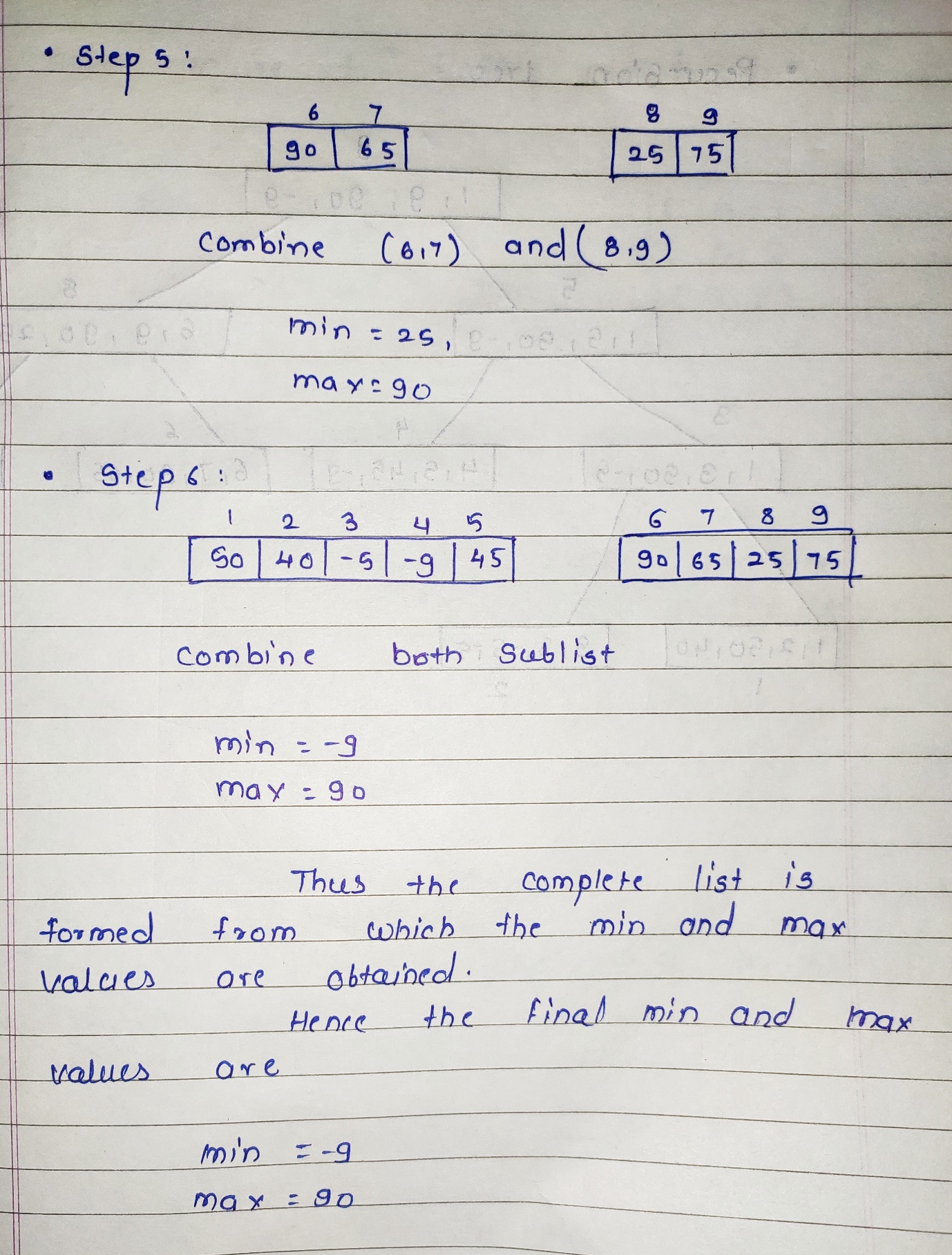




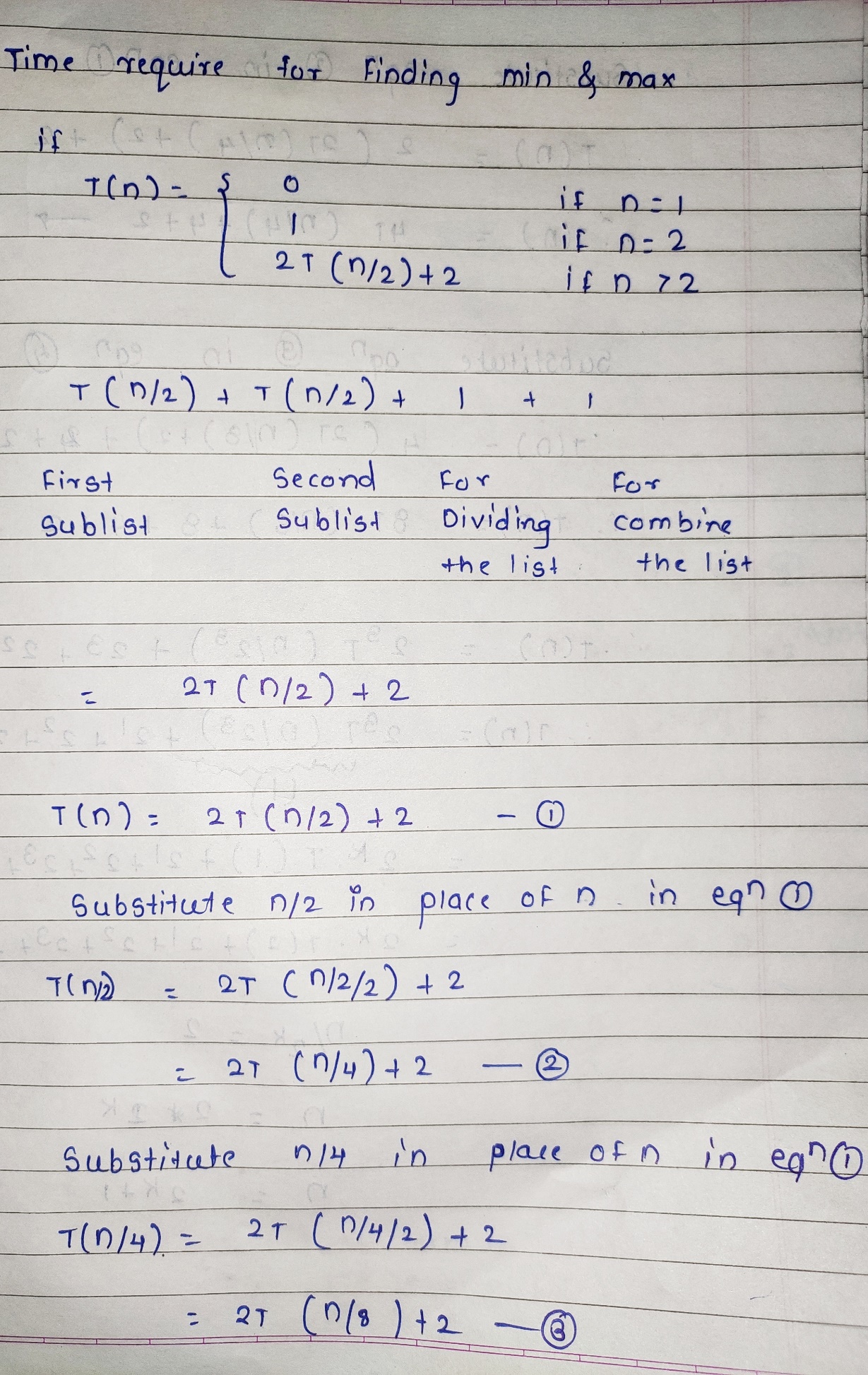
**Explain the problem:**

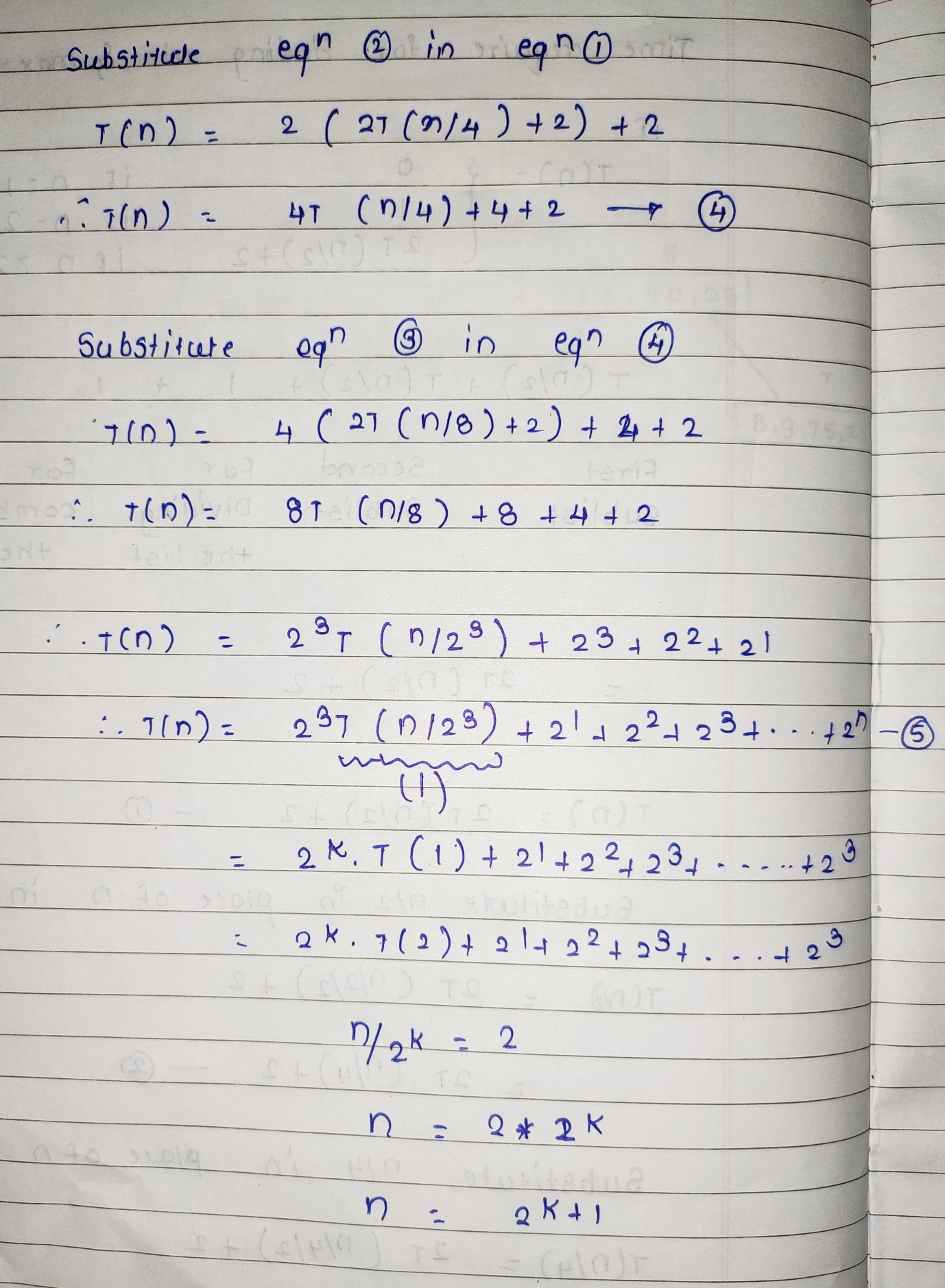


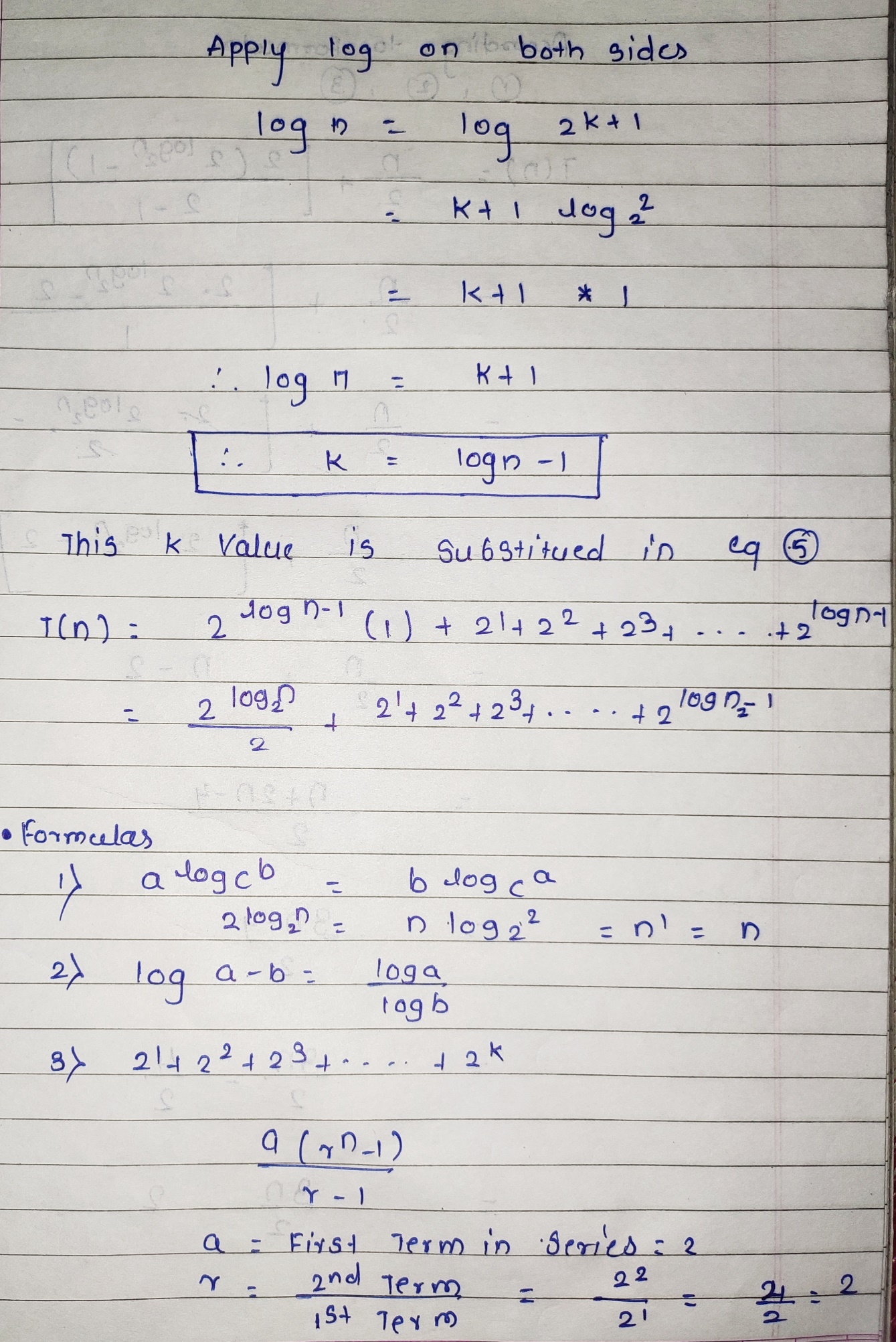


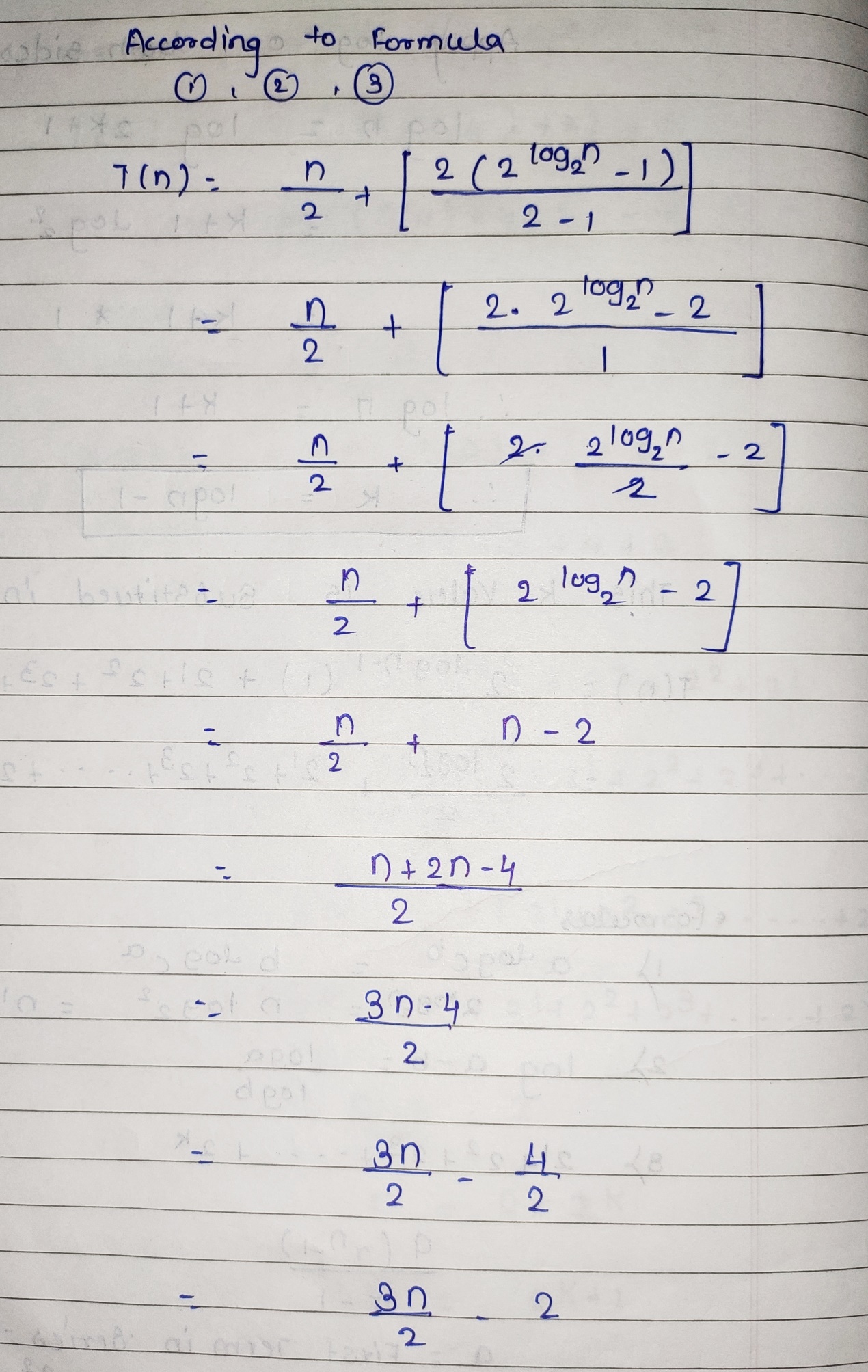


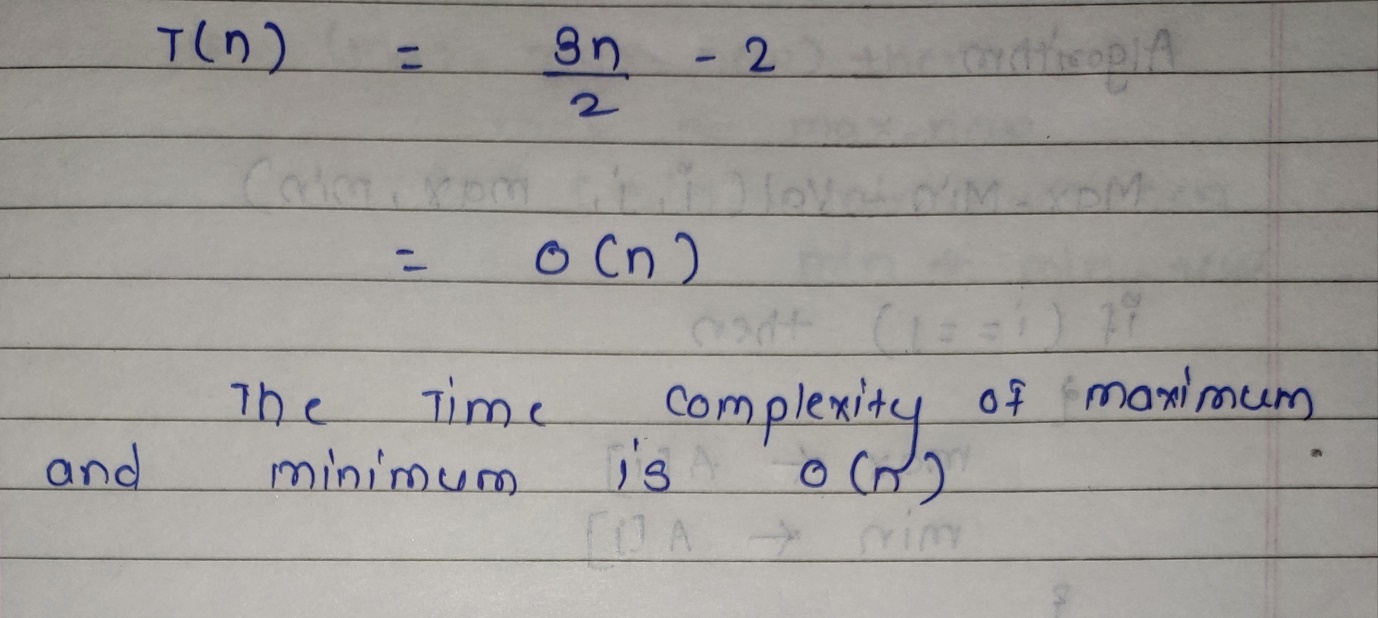
**Discuss the time complexity of straight Minmax :**





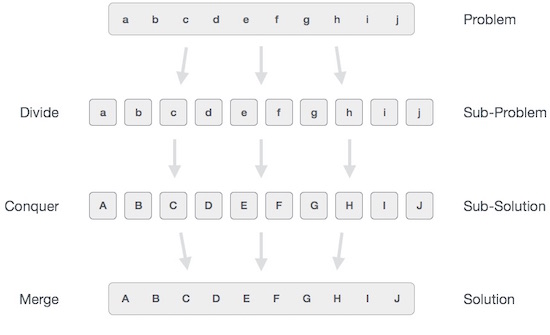






**Divide and Conquer approach for finding MinMax:**

In divide and conquer approach, the problem in hand, is divided into smaller sub-problems and then each problem is solved independently. When we keep on dividing the subproblems into even smaller sub-problems, we may eventually reach a stage where no more division is possible. Those "atomic" smallest possible sub-problem (fractions) are solved. The solution of all sub-problems is finally merged in order to obtain the solution of an original problem.



Broadly, we can understand **divide-and-conquer** approach in a three-step process.

## **Divide/Break**

This step involves breaking the problem into smaller sub-problems. Sub-problems should represent a part of the original problem. This step generally takes a recursive approach to divide the problem until no sub-problem is further divisible. At this stage, sub-problems become atomic in nature but still represent some part of the actual problem.

## **Conquer/Solve**

This step receives a lot of smaller sub-problems to be solved. Generally, at this level, the problems are considered 'solved' on their own.

## **Merge/Combine**

When the smaller sub-problems are solved, this stage recursively combines them until they formulate a solution of the original problem. This algorithmic approach works recursively and conquer & merge steps works so close that they appear as one.

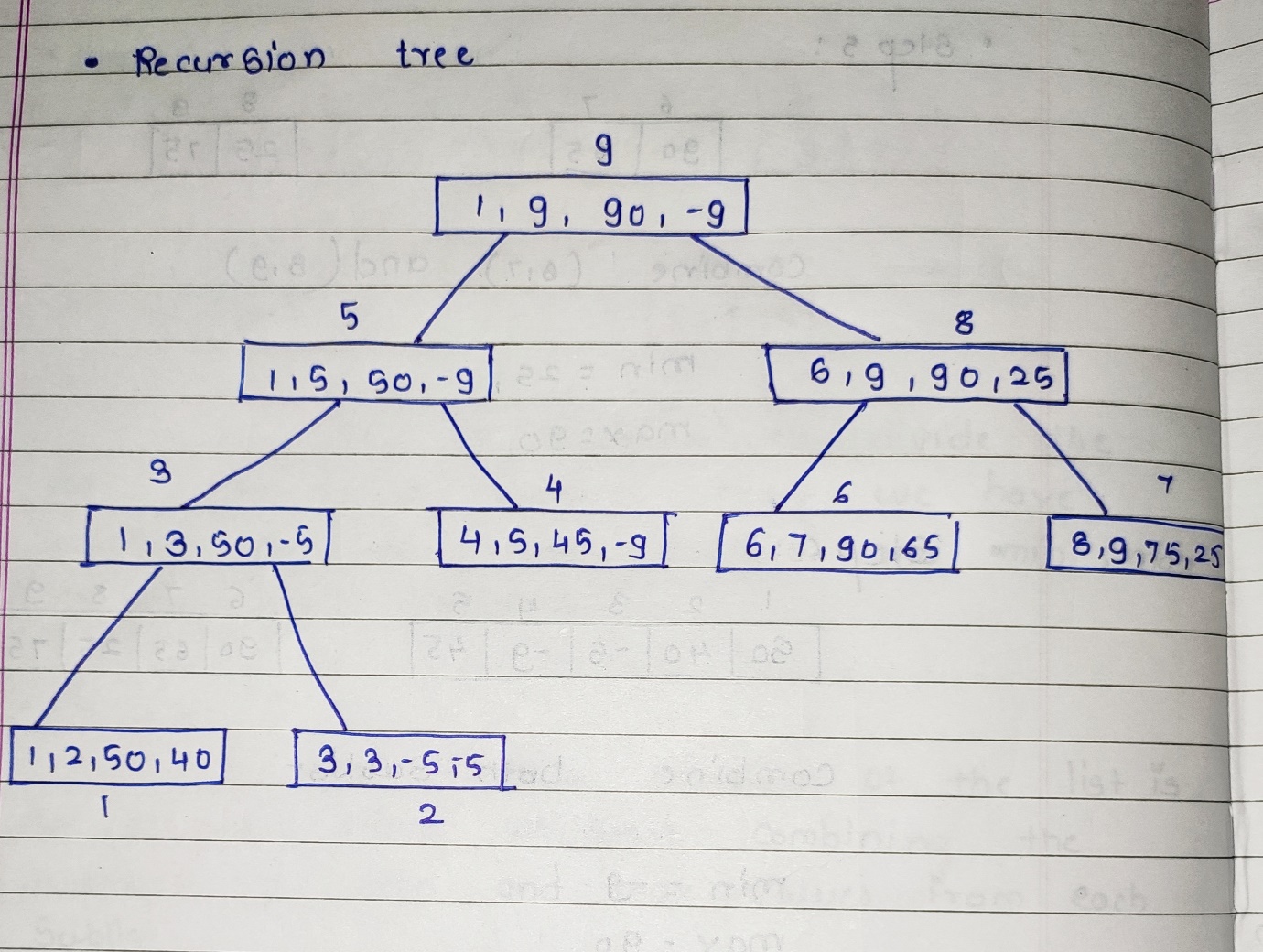
**Comment On Time Complexity:**

Number of comparisons requires applying the divide and conquering algorithm on n elements/items = Max - Min Problem

Number of comparisons requires applying general approach on n elements = (n-1) + (n-1) = 2n-2

Time Complexity: O(n)

**Draw Recursion tree showing calculation for your input numbers.**



**Code:**

#include<stdio.h>

#include<stdlib.h>

void findmaxmin( int , int , int \* , int \*);

int a[100] , n ;

int main(){

int l, h, i , min , max ;

printf("Enter the number of elements: ");

scanf("%d",&n);

printf("Enter %d elements: ",n);

for(i=0;i<n;i++)

{

scanf("%d",&a[i]);

}

l=0;

h=n-1;

findmaxmin(l , h , &max , &min );

printf("----------Final Result----------");

printf("\n Maximum element: %d",max);

printf("\n Minimum element: %d",min);

}

void findmaxmin( int l , int h , int \*max ,int \*min )

{

int lmax , rmax , lmin , rmin , mid ;

if( l == h )

{

\*max = a[h];

\*min = a[l];

printf("One element Present in sublist:{ %d } \n Minimum Element: %d\t\n Maximum Element:%d\n",a[l],\*min,\*max);

return;

}

else if ( l + 1 == h )

{

if ( a[l] >= a[h] )

{

\*max = a[l];

\*min = a[h];

printf("Two elements Present in Sublist:{ %d\t%d } \n Minimum Element: %d\t\n Maximum Element:%d\n",a[l],a[h],\*min,\*max);

return;

}

else

{

\*max = a[h];

\*min = a[l];

printf("Two elements Present in Sublist:{ %d\t%d } \n Minimum Element: %d\t\n Maximum Element:%d\n",a[l],a[h],\*min,\*max);

return;

}

}

else

{

mid = (l + h) / 2 ;

findmaxmin( l , mid , &lmax , &lmin );

findmaxmin( mid + 1 , h , &rmax , &rmin ) ;

if ( lmax > rmax )

\*max = lmax;

else

\*max = rmax;

if ( lmin > rmin )

\*min = rmin ;

else

\*min = lmin ;

printf("Every elements in sublist:");

printf("{");

for(int i=l;i<h+1;i++){

printf("%d\t",a[i]);

}

printf("}");

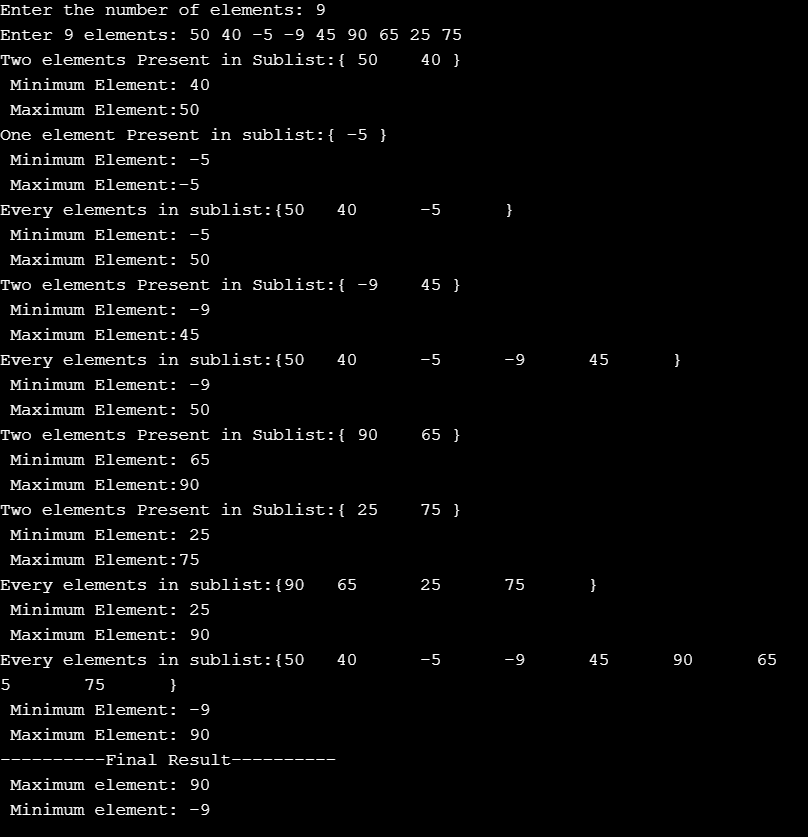
printf("\n Minimum Element: %d\t\n Maximum Element: %d\n",\*min,\*max);

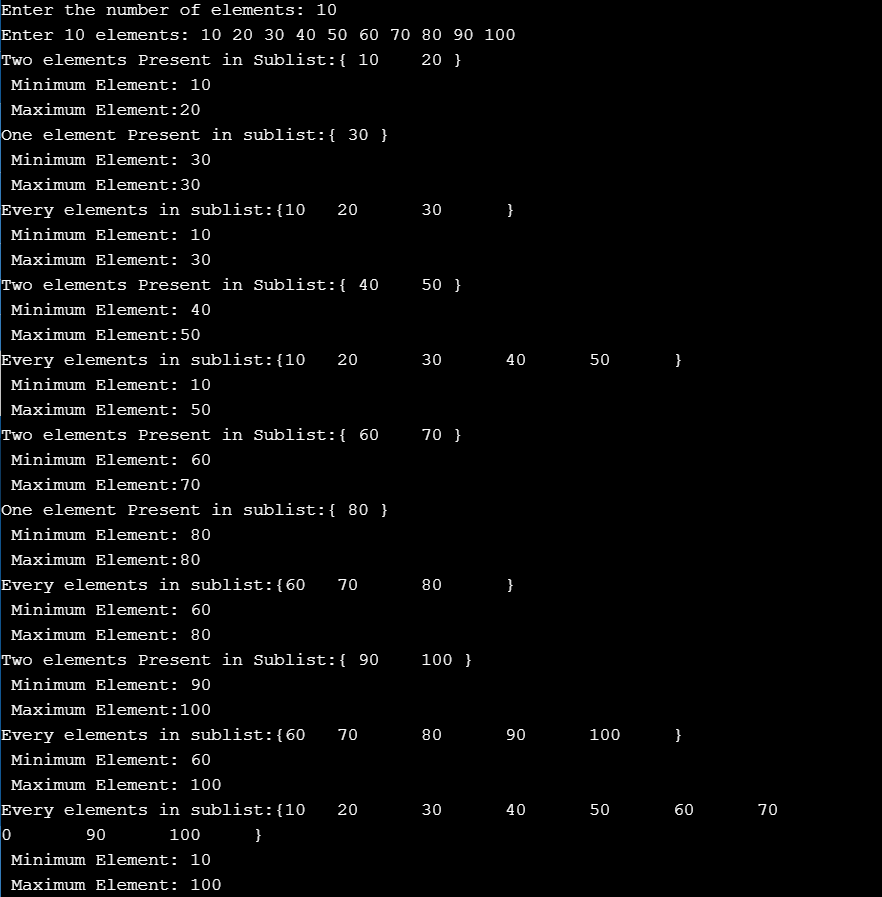
return;

}

}

**Output:**





**Conclusion:** In minimum maximum the list of elements is divided at the mid in order to obtain two sublists. From both the sublist maximum and minimum elements are chosen. Two maxima and minima are compared and from them real maximum and minimum elements are determined.Also learn understood how to reduce the number of comparison while finding the minimum and maximum value from a set of numbers.