**Batch: A1 Roll No.: 1911004**

**Experiment No. 2**

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

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

|  |
| --- |
| **Title: Implementation of Binary search/Max-Min algorithm** |

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

**CO to be achieved:**

|  |  |
| --- | --- |
| Sr. No | Objective |
| CO 1 | Analyze the asymptotic running time and space complexity of algorithms. |
| CO 2 | Describe various algorithm design strategies to solve different problems and analyze  Complexity. |
| CO 3 | Develop string matching techniques |
| CO 4 | Describe the classes P, NP, and NP-Complete |

**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/Binary\_search\_algorithm**
4. **https://www.princeton.edu/~achaney/tmve/wiki100k/docs/Binary\_search\_algorithm.html**
5. **http://video.franklin.edu/Franklin/Math/170/common/mod01/binarySearchAlg.html**
6. **http://xlinux.nist.gov/dads/HTML/binarySearch.html**
7. **https://www.cs.auckland.ac.nz/software/AlgAnim/searching.html**

**Pre Lab/ Prior Concepts:** Data structures

**Historical Profile:**

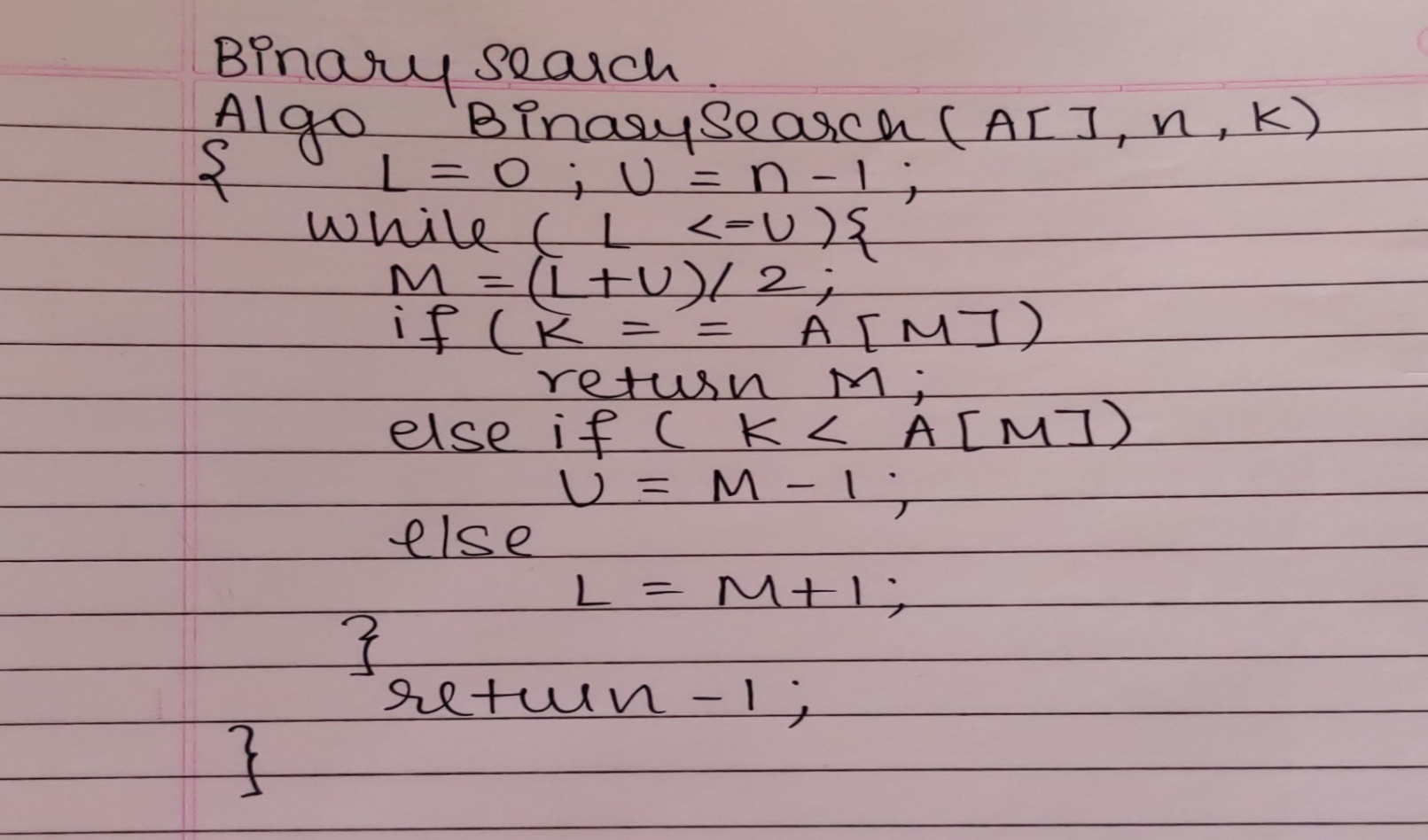
Finding maximum and minimum or Binary search are few problems those are solved with the divide-and-conquer technique. This is one the simplest strategies which basically works on dividing the problem to the smallest possible level.

Binary Search is an extremely well-known instance of divide-and-conquer paradigm. Given an ordered array of n elements, the basic idea of binary search is that for a given element , "probe" the middle element of the array. Then continue in either the lower or upper segment of the array, depending on the outcome of the probe until the required (given) element is reached.

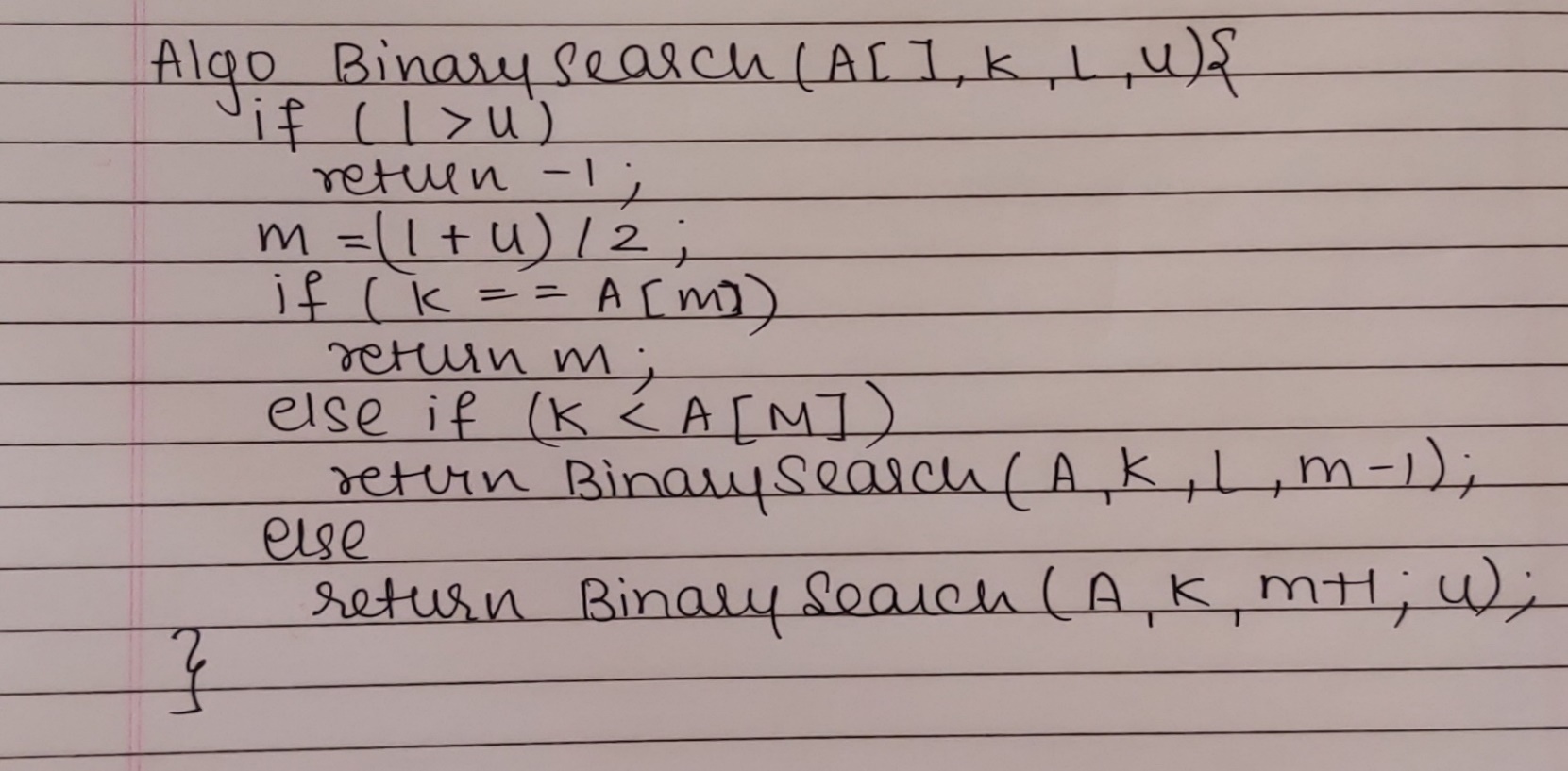
**New Concepts to be learned:**

Number of comparisons, Application of algorithmic design strategy to any problem, Classical problem solving Vs Divide-and-Conquer problem solving.

**Algorithm IterativeBinarySearch**



**Algorithm RecursiveBinarySearch**



**CODE :**

import java.util.\*;

class Main {

public static void main(String[] args) {

Scanner ob = new Scanner(System.in);

System.out.println("Enter no of Elements ");

int n=ob.nextInt();

System.out.println("Enter Sorted Array Only ");

int A[]=new int[n];

System.out.println("Enter Elements ");

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

A[i]=ob.nextInt();

System.out.println("Enter Key to find ");

int K = ob.nextInt();

System.out.println("Binary Search By Recurrsion ");

int pos1 = BinarySearch(A,K,0,A.length);

if(pos1>-1)

System.out.println(K+" Key Found at position "+(pos1+1));

else

System.out.println(K+" Key not Found");

System.out.println("Binary Search By Iteration ");

int pos2 = BinarySearch(A,A.length,K);

if(pos2>-1)

System.out.println(K+" Key Found at position "+(pos2+1));

else

System.out.println(K+" Key not Found");

}

public static int BinarySearch(int A[],int n,int K){

int L=0,U=n-1,M;

while(L<=U){

M = (L+U)/2;

if(K==A[M])

return M;

else if (K<A[M])

U=M-1;

else

L=M+1;

} return -1;

}

public static int BinarySearch(int[] A,int K,int l,int u){

if(l>u)

return -1;

int m = (l+u)/2;

if(K==A[m])

return m;

else if (K<A[m])

return BinarySearch(A, K, l, m-1);

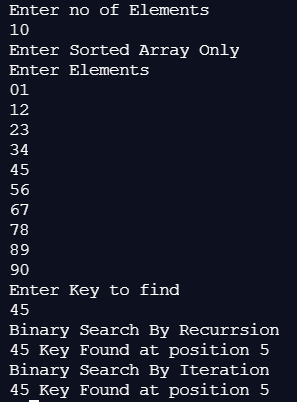
else

return BinarySearch(A, K, m+1, u);

}

}

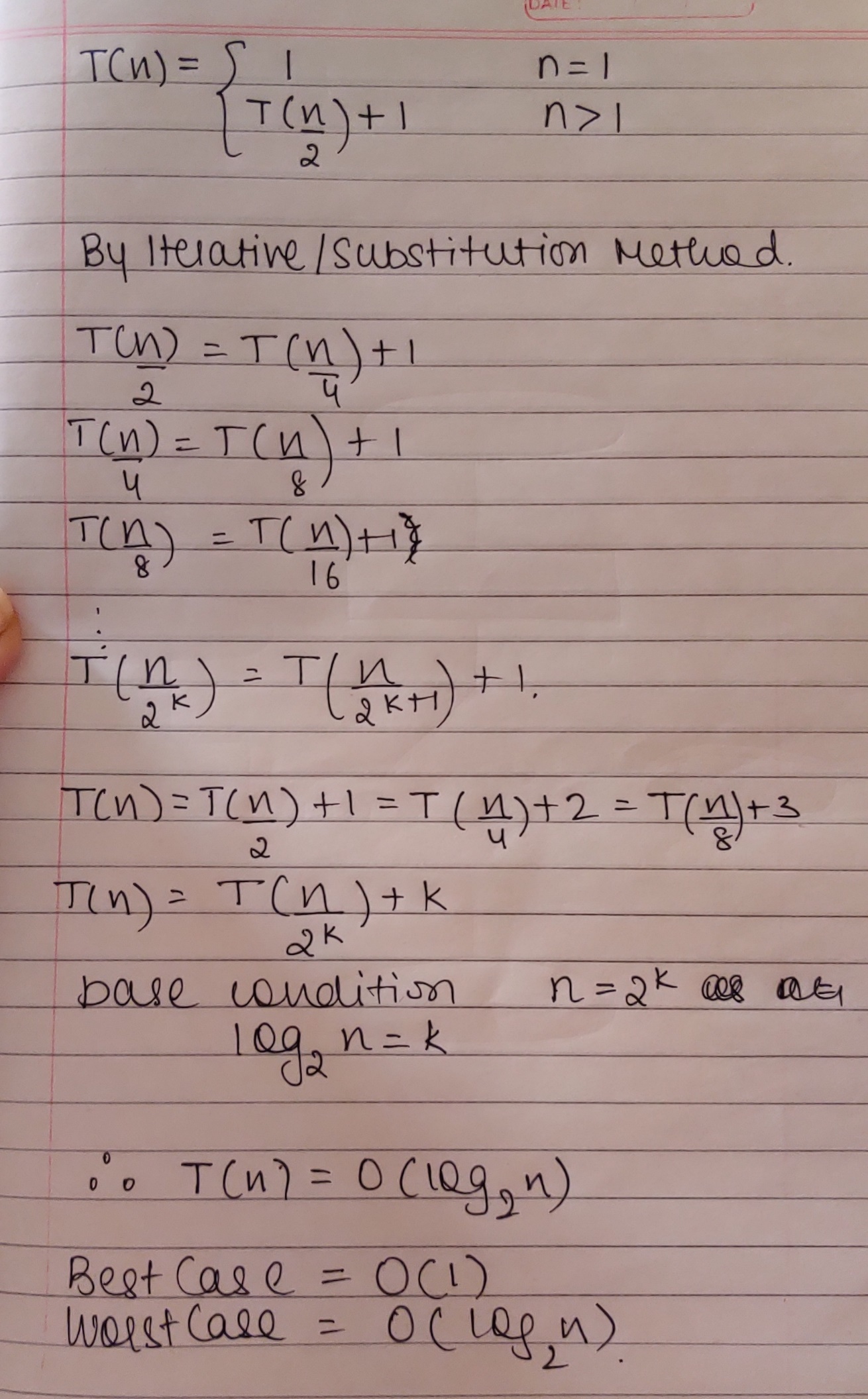
**OUTPUT:**



**The space complexity of Recursive Binary Search: O(log n)**

**The space complexity of Iterative Binary Search: O(1)**

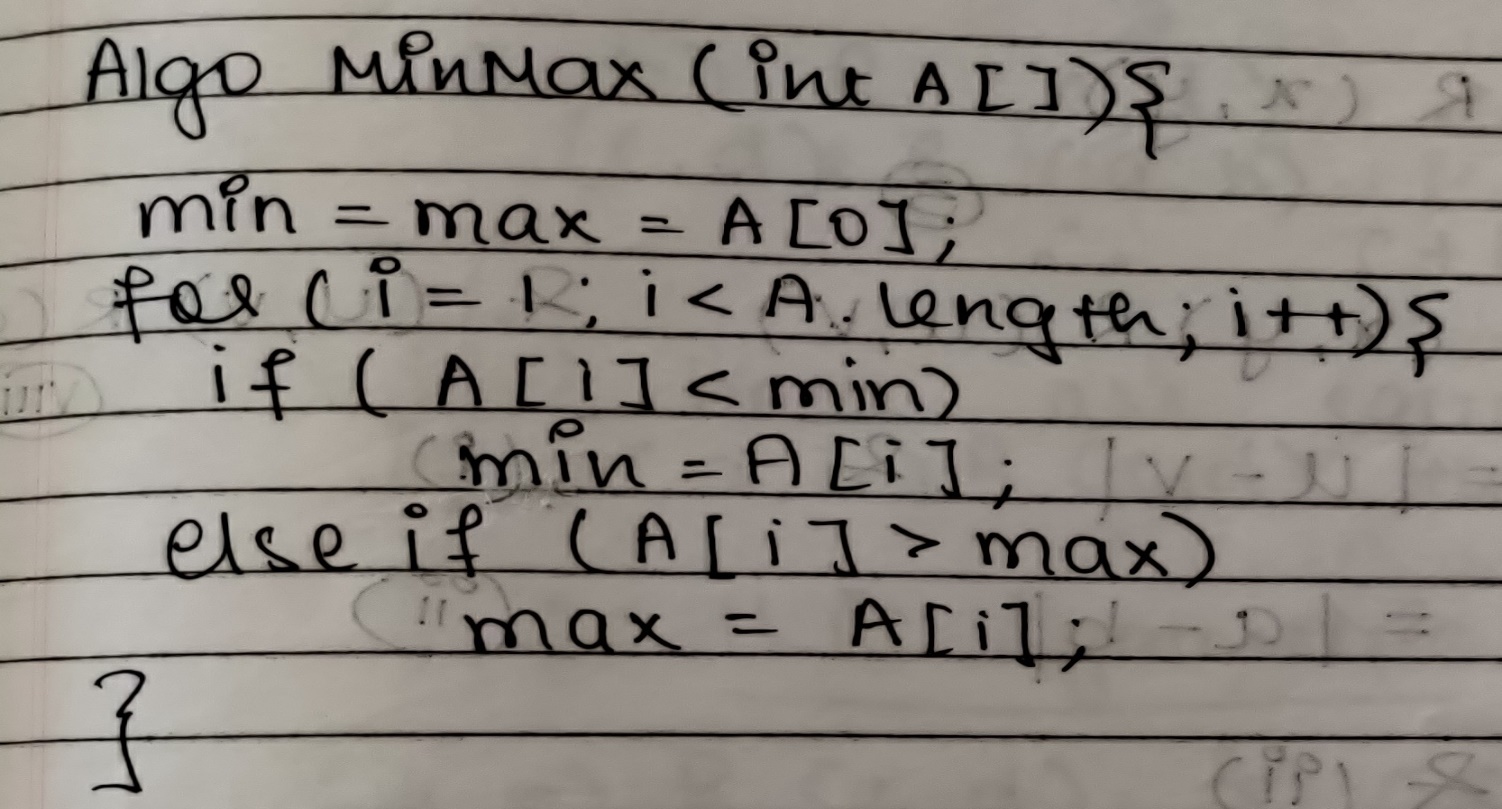
**The Time complexity of Binary Search:**

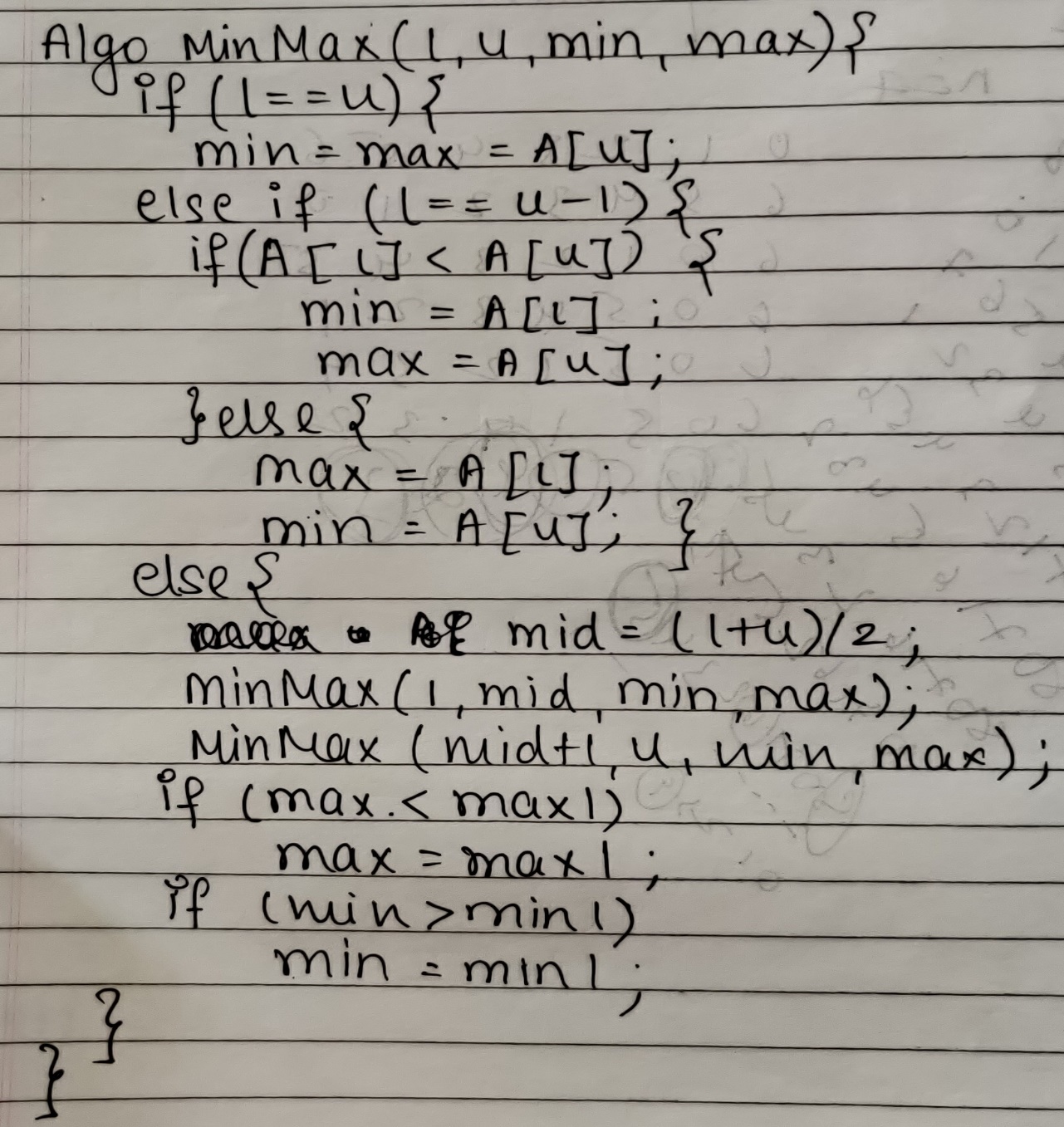


Best Case = O(1)

Average Case = Worst Case = O(log n)

**Algorithm Straight MaxMin:**

**Algorithm: Recursive Max-Min**



**The space complexity of Max-Min:**

**1. Normal O(1)**

**2. Recursive O(log n)**

**CODE :**

import java.util.\*;

class Main {

public static int[] A;

public static void main(String[] args) {

Scanner ob = new Scanner(System.in);

System.out.println("Enter num of elements");

int n = ob.nextInt();

A = new int[n];

int mM[]=new int[2];

System.out.println("Enter elements");

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

A[i]=ob.nextInt();

long start1 = System.currentTimeMillis();

minMax();

long end1 = System.currentTimeMillis();

System.out.println("\nTime Taken for Iterative = "+(end1 - start1)+" millisec");

long start2 = System.currentTimeMillis();

mM = minMax(0,A.length-1,A[0],A[0]);

System.out.println("Min-Max Using Recursive "+"\nMIN = "+mM[0]+"\nMAX = "+mM[1]);

long end2 = System.currentTimeMillis();

System.out.println("\nTime Taken for Recursive = "+(end2 - start2)+" millisec");

}

public static void minMax(){

int min = A[0],max = A[0],i;

for(i=1;i<A.length;i++){

if (A[i]>max)

max=A[i];

else

if (A[i]<min)

min = A[i];

} System.out.println("Min-Max Using Iterative "+"\nMIN = "+min+"\nMAX = "+max);

}

public static int[] minMax(int l,int u,int min,int max){

int mid,m1[] = new int[2],m[]= new int[2];

if (l==u){

max = A[u];

min = A[l];

}else if (l==u-1){

if (A[l]<A[u]){

max = A[u];

min = A[l];

}else{

max = A[l];

min = A[u];

}

}else{

mid = (l+u)/2;

m1 = minMax(l,mid,min,max);

m1[0] = A[mid+1];

m1[1] = A[u];

minMax(mid+1,u,m1[0],m1[1]);

if(max<m1[1]) max = m1[1];

if (min>m1[0]) min = m1[0];

}

m[0]=min;

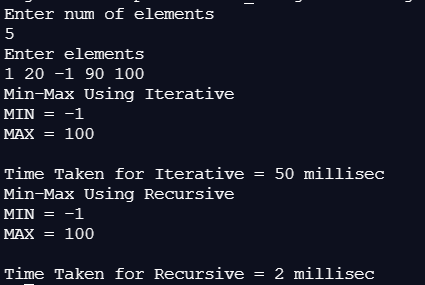
m[1]=max;

return m;

}

}

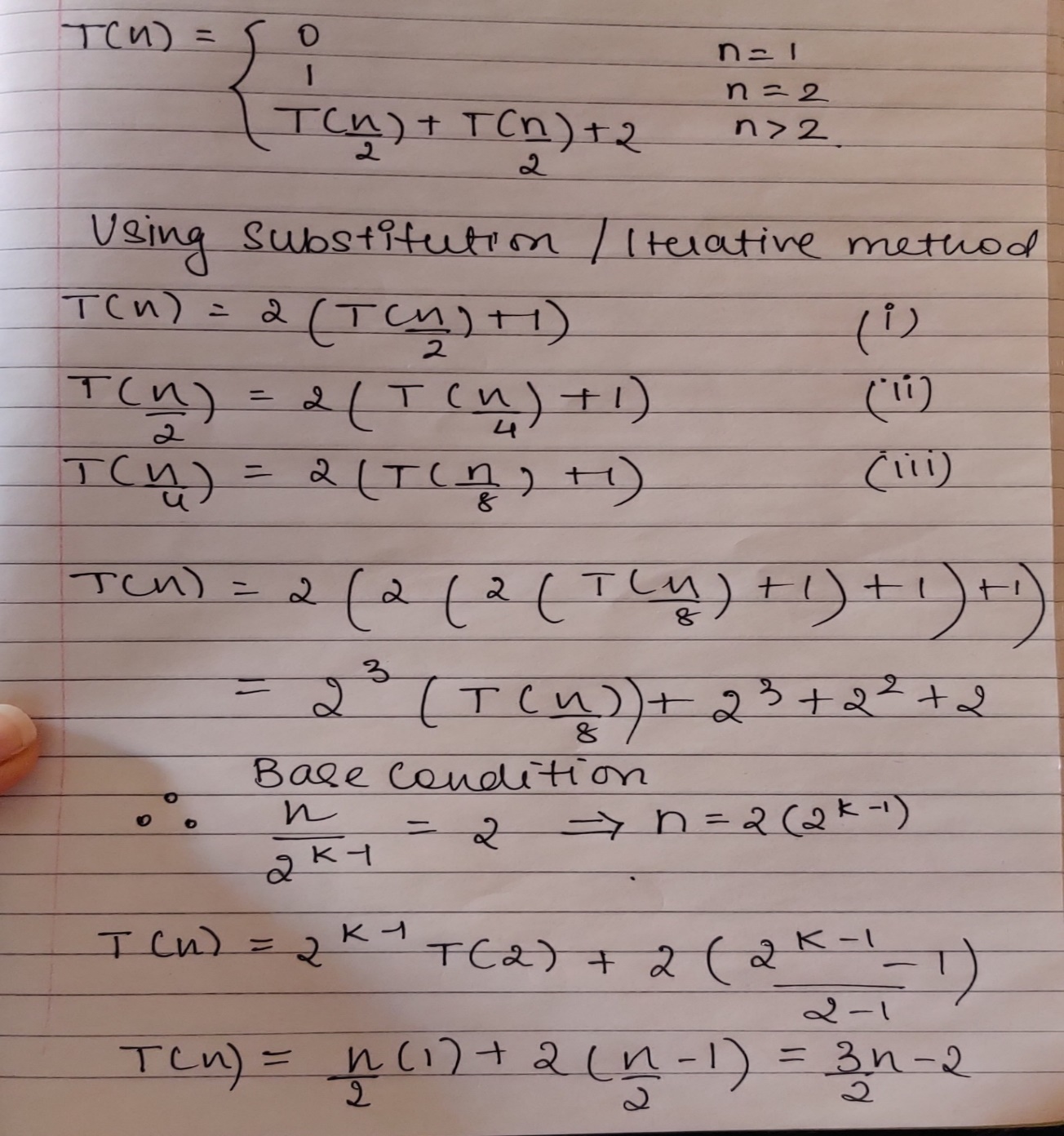
**OUTPUT:**



**Time complexity for Max-Min:**

Best Case = O(1)

Worst Case = O(n)



**Conclusion:**

We successfully implemented the concept of Divide & Conquer using Binary Search & Min-Max Algorithms in java by getting correct necessary output & Complexity associated with them.