Course Name:	**Analysis of Algorithms**	**Semester:**	**IV**
Date of Performance:	**12 / 02 / 2024**	**Batch No:**	**EXCP B1**
Faculty Name:	**Prof. Payal Varangoankar**	**Roll No:**	**16014022096**
Faculty Sign & Date:		**Grade/Marks:**	

Experiment No: 2

Title: Binary search Finding Minimum and Maximum.

Aim and Objective of the Experiment:

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

COs to be achieved:

CO2: Describe various algorithm design strategies to solve different problems.

Theory:

Historical Profile:

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

Binary Search is an extremely well-known instance of the 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:

1. Number of comparisons
2. Application of algorithmic design strategy to any problem
3. Classical problem-solving vs. Divide-and-Conquer problem-solving.

Code:

BINARY SEARCH ITERATIVE:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int binary_search(int A[], int key, int imin, int imax, int *iterations)
{
    *iterations = 0;

    while (imax >= imin) {
        (*iterations)++;
        int imid = (imin + imax) / 2;

        if (A[imid] == key) {
            return imid;
        } else if (A[imid] < key) {
            imin = imid + 1;
        } else {
            imax = imid - 1;
        }
    }
    return -1;
}

int main() {

    int n, value, iterations;

    printf("Enter value of n: ");
    scanf("%d", &n);

    int arr[n];
    printf("Original Array: ");
    for (int i = 0; i < n; i++) {
        arr[i] = i;
        printf("%d ", arr[i]);
    }
    printf("\n");

    printf("Enter value you want to search for: ");
    scanf("%d", &value);

    clock_t l1, l2;
```

```
    l1 = clock();
    int index = binary_search(arr, value, 0, n - 1, &iterations);
    l2 = clock();

    if (index != -1) {
        printf("Value found at index: %d\n", index);
    } else {
        printf("Value not found\n");
    }

    double t1 = 1000000 * ((double)(l2 - l1)) / CLOCKS_PER_SEC;
    printf("TIME (in micro seconds) : %f \n", t1);
    printf("Number of iterations: %d\n", iterations);

    return 0;
}
```

BINARY SEARCH RECURSIVE:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void binary_search(int A[], int key, int imin, int imax, int *iterations) {
    if (imax < imin) {
        printf("KEY NOT FOUND");
    } else {
        int imid = (imin + imax) / 2;
        (*iterations)++;
        if (A[imid] < key) {
            binary_search(A, key, imid + 1, imax, iterations);
        } else if (A[imid] > key) {
            binary_search(A, key, imin, imid - 1, iterations);
        } else {
            printf("\nKEY %d FOUND AT %d position", A[imid], imid);
        }
    }
}

int main() {
    int n, value;
    clock_t l1, l2;
```

```
    printf("Enter value of n: ");
    scanf("%d", &n);

    int arr[n];
    printf("Original Array: ");
    for (int i = 0; i < n; i++) {
        arr[i] = i;
        printf("%d ", arr[i]);
    }
    printf("\n");

    printf("Enter value you want to search for: ");
    scanf("%d", &value);

    l1 = clock();
    int iterations = 0;
    binary_search(arr, value, 0, n - 1, &iterations);
    l2 = clock();

    double t1 = ((double)(l2 - l1)) / CLOCKS_PER_SEC;
    printf("TIME : %f \n", t1);
    printf("Number of iterations: %d\n", iterations);

    return 0;
}
```

Max-Min Method:

```
#include <stdio.h>
#include <time.h>

void MaxMin_linear(int a[], int n, int* maxl, int* minl, int* iterations) {
    *minl = *maxl = a[0];
    *iterations = 0;

    for(int i = 1; i < n; i++) {
        (*iterations)++;
        if (a[i] >= *maxl) {
            *maxl = a[i];
        } else if (a[i] <= *minl) {
            *minl = a[i];
        }
```

```
    }
}

void MaxMin_dac(int a[], int i, int j, int* max, int* min, int* iterations) {
    if (i == j) {
        *min = *max = a[i];
    } else if (i == j - 1) {
        (*iterations)++;
        if (a[i] < a[j]) {
            *max = a[j];
            *min = a[i];
        } else {
            *max = a[i];
            *min = a[j];
        }
    } else {
        int max1, min1;
        int mid = (i + j) / 2;

        MaxMin_dac(a, i, mid, max, min, iterations);
        MaxMin_dac(a, mid + 1, j, &max1, &min1, iterations);

        if (*max < max1) {
            *max = max1;
        }
        if (*min > min1) {
            *min = min1;
        }
        *iterations += 2;
    }
}

int main() {
    int maxl, minl, max, min, n, iterations_linear, iterations_dac;
    clock_t l1, l2, d1, d2;

    printf("Enter value of n: ");
    scanf("%d", &n);

    int arr[n];
    printf("Original Array: ");
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 10;
        printf("%d ", arr[i]);
```

```
    }
    printf("\n");

    l1 = clock();
    MaxMin_linear(arr, n, &maxl, &minl, &iterations_linear);
    l2 = clock();

    d1 = clock();
    MaxMin_dac(arr, 0, n - 1, &max, &min, &iterations_dac);
    d2 = clock();

    double t1 = 1000000 * ((double)(l2 - l1)) / CLOCKS_PER_SEC;
    double t2 = 1000000 * ((double)(d2 - d1)) / CLOCKS_PER_SEC;

    printf("LINEAR : \n");
    printf("MAX = %d, MIN = %d \n", maxl, minl);
    printf("TIME (in microseconds) : %.2f \n", t1);
    printf("Number of iterations: %d\n", iterations_linear);

    printf("DIVIDE AND CONQUER : \n");
    printf("MAX = %d, MIN = %d \n", max, min);
    printf("TIME (in microseconds): %.2f \n", t2);
    printf("Number of iterations: %d\n", iterations_dac);

    return 0;
}
```

Stepwise-Procedure / Algorithm:

Algorithm Iterative Binary Search

int binary_search(int A[ ], int key, int imin, int imax)
//The algorithm takes as parameters an array A[1.. n] , the search key and lower-higher index pair of the array.
// Output- The algorithm returns index of the search key in the given array, if it's present.

```
{
  // continue searching while [imin, imax] is not empty
  WHILE (imax >= imin)
    {
      // calculate the midpoint for roughly equal partition
      int imid = midpoint(imin, imax);
      IF(A[imid] == key)
        // key found at index imid
```

```
        return imid;
      // determine which subarray to search
      ELSE If (A[imid] < key)
        // change min index to search upper subarray
        imin = imid + 1;
      ELSE
        // change max index to search lower subarray
        imax = imid - 1;
    }
  // key was not found
  RETURN KEY_NOT_FOUND;
}
```

The space complexity of Iterative Binary Search:

Algorithm Recursive Binary Search

```
int binary_search(int A[], int key, int imin, int imax)
//The algorithm takes as parameters an array A[1.. n] , the search key and lower-higher index pair of the array.
 // Output- The algorithm returns index of the search key in the given array, if it's present.
{
  // test if array is empty
  IF (imax < imin)
    // set is empty, so return value showing not found
    RETURN KEY_NOT_FOUND;
  ELSE{
      // calculate midpoint to cut set in half
      int imid = midpoint(imin, imax);
       // three-way comparison
      IF (A[imid] > key)
        // key is in  lower subset
        RETURN binary_search(A, key, imin, imid-1);
      ELSE IF (A[imid] < key)
        // key is in  higher subset
        RETURN binary_search(A, key, imid+1, imax);
      ELSE
        // key has been found
        RETURN imid;
    }
}
```

Algorithm StraightMaxMin:

```
VOID StraightMaxMin (Type a[], int n, Type& max, Type& min)
// Set max to the maximum and min to the minimum of a[1:n].
{   max = min = a[1];
```

```
FOR (int i=2; i<=n; i++)
{
IF (a[i]>max) then max = a[i];
IF (a[i]<min) min = a[i];
}
}
```

Algorithm: Recursive Max-Min

```
 VOID MaxMin(int i, int j, Type& max, Type& min)
// A[1:n] is a global array. Parameters i and j are integers, 1 <= i <= j <= n.
 //The effect is to set max and min to the largest and smallest values in a[i:j], respectively.
{
IF (i == j) max = min = a[i]; // Small(P)
ELSE IF (i == j-1) { // Another case of Small(P)
          IF (a[i] < a[j])
 max = a[j]; min = a[i];
ELSE { max = a[i]; min = a[j];
}
ELSE {     Type max1, min1;

// If P is not small divide P into subproblems.  Find where to split the set.

int mid=(i+j)/2;
// solve the sub problems.
 MaxMin(i, mid, max, min);
MaxMin(mid+1, j, max1, min1);
// Combine the solutions.
IF (max < max1) max = max1;
IF (min > min1) min = min1;

}
}
```

Output:

BINARY METHOD ITERATIVE :

```
Enter value of n: 5
Original Array: 0 1 2 3 4
Enter value you want to search for: 0
Value found at index: 0
TIME (in micro seconds) : 2.000000
Number of iterations: 2
```

BINARY METHOD RECURSIVE:

```
Enter value of n: 5
Original Array: 0 1 2 3 4
Enter value you want to search for: 0

KEY 0 FOUND AT 0 position
TIME : 20.000000
Number of iterations: 2
```

Max-Min METHOD:

```
Enter value of n: 5
Original Array: 3 6 7 5 3
LINEAR :
MAX = 7, MIN = 3
TIME (in microseconds) : 2.00
Number of iterations: 4
DIVIDE AND CONQUER :
MAX = 7, MIN = 3
TIME (in microseconds): 1.00
Number of iterations: 6
```

Observation Table:

BINARY SEARCH METHOD:

	abscissa x →	ordinate y or f(x) ↑
1	5	2
2	10	3
3	50	5
4	100	7
5	150	8
6	...	...

★ Origins ◉ automatically calculated
○ Set to (0,0)
★ Horizontal x-axis name: n
★ Vertical y-axis name: count
★ Legend: linear binary search
★ Dot size: 10
★ Display mode ○ Only points
◉ Draw line between points

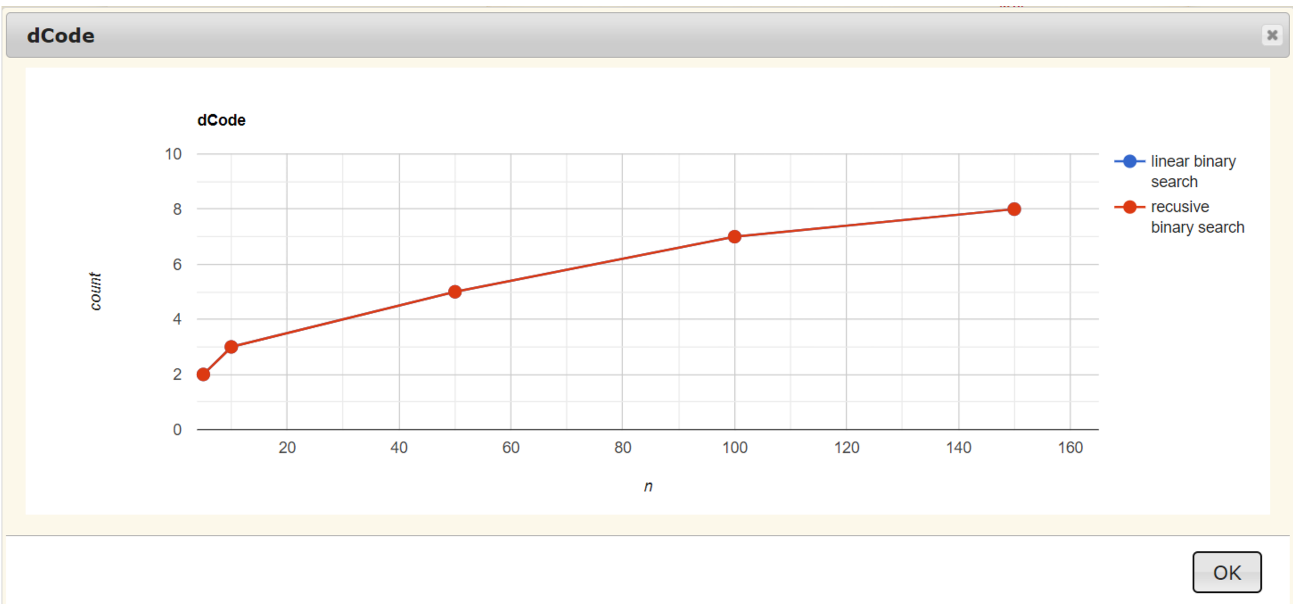
▶ PLOT

See also: Function Equation Finder — Y-Intercept

2nd Plot (on the same graph)

	abscissa x →	ordinate y or f(x) ↑
1	5	2
2	10	3
3	50	5
4	100	7
5	150	8
6	...	...

★ Legend: recusive binary sea...


dCode
dCode
10
8
6
4
2
0
count
20
40
60
80
100
120
140
160
n
linear binary search
recusive binary search
OK

MAX MIN METHOD:

	abscissa x ⇢	ordinate y or f(x) ↑
1	5	4
2	10	9
3	100	49
4	150	99
5	...	...
6	...	...

- ORIGINS ◉ AUTOMATICALLY CALCULATED
  ○ SET TO (0,0)
- HORIZONTAL X-AXIS NAME: n
- VERTICAL Y-AXIS NAME: count
- LEGEND: linear max min
- DOT SIZE: 10
- DISPLAY MODE ○ ONLY POINTS
  ◉ DRAW LINE BETWEEN POINTS

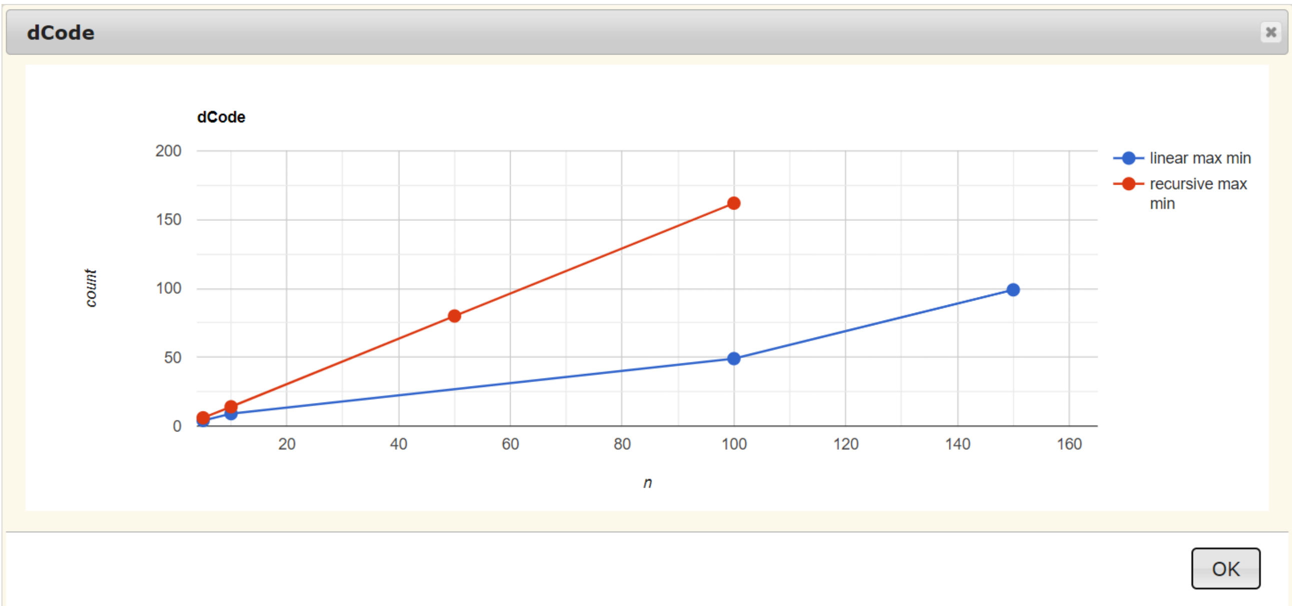
▶ PLOT

See also: **Function Equation Finder** – **Y-Intercept**

2ND PLOT (ON THE SAME GRAPH)

	abscissa x ⇢	ordinate y or f(x) ↑
1	5	6
2	10	14
3	50	80
4	100	162
5	...	...

- LEGEND: recursive max min


dCode
dCode
200
150
100
50
0
count
20
40
60
80
100
120
140
160
n
linear max min
recursive max min
OK

Calculations:

16014022096

```
Binary_Search (int A[], int low, int high, int key)
{
  if low > high
    return -1
  else
  int middle = midpoint (low, high)
    IF( A[middle] > key
      RETURN Binary_search (A, low, middle-1, key)
    ELSE IF ( A[middle] > key)
      RETURN Binary_search (A, middle+1, high, key)
    ElSE
      RETURN middle
```

from above

$T(n) = T(n/2) + c$

$T(n/2) = T(n/4) + 2c$

$T(n/4) = T(n/8) + 3c$

$\vdots$

$\frac{n}{2^k} = 1$

$k = \log_2 n$

$T(n) = T(1) + (\log_2 n)\,c$

time complexity $O(\log_2 n)$

space complexity $O(\log_2 n)$

```
MAX_MIN ( int A[], int n, int max, int min)
{ max = min = A[0]
   for ( int i = 1, i <= n, i++)
      IF (A[i] > max)
      max = a(i)
      ElSE IF (A[i] < min)
      min = a(i)
}
```

from above
time complexity : $O(n)$

space complexity : $O(1)$

Calculations:
The space complexity of Recursive Binary Search: $O(\log_2 n)$ **The Time complexity of Binary Search:** $O(\log_2 n)$ **The space complexity of Max-Min:** $O(\log_2 n)$ **Time complexity for Max-Min:** $O(n)$

Post Lab Subjective/Objective type Questions:

Solve the problems theoretically which was implemented during practical

BINARY SEARCH

0	1	2	3	4

Key → 0

A[imid] = 2
A[imid] ≠ key
A[imid] > key

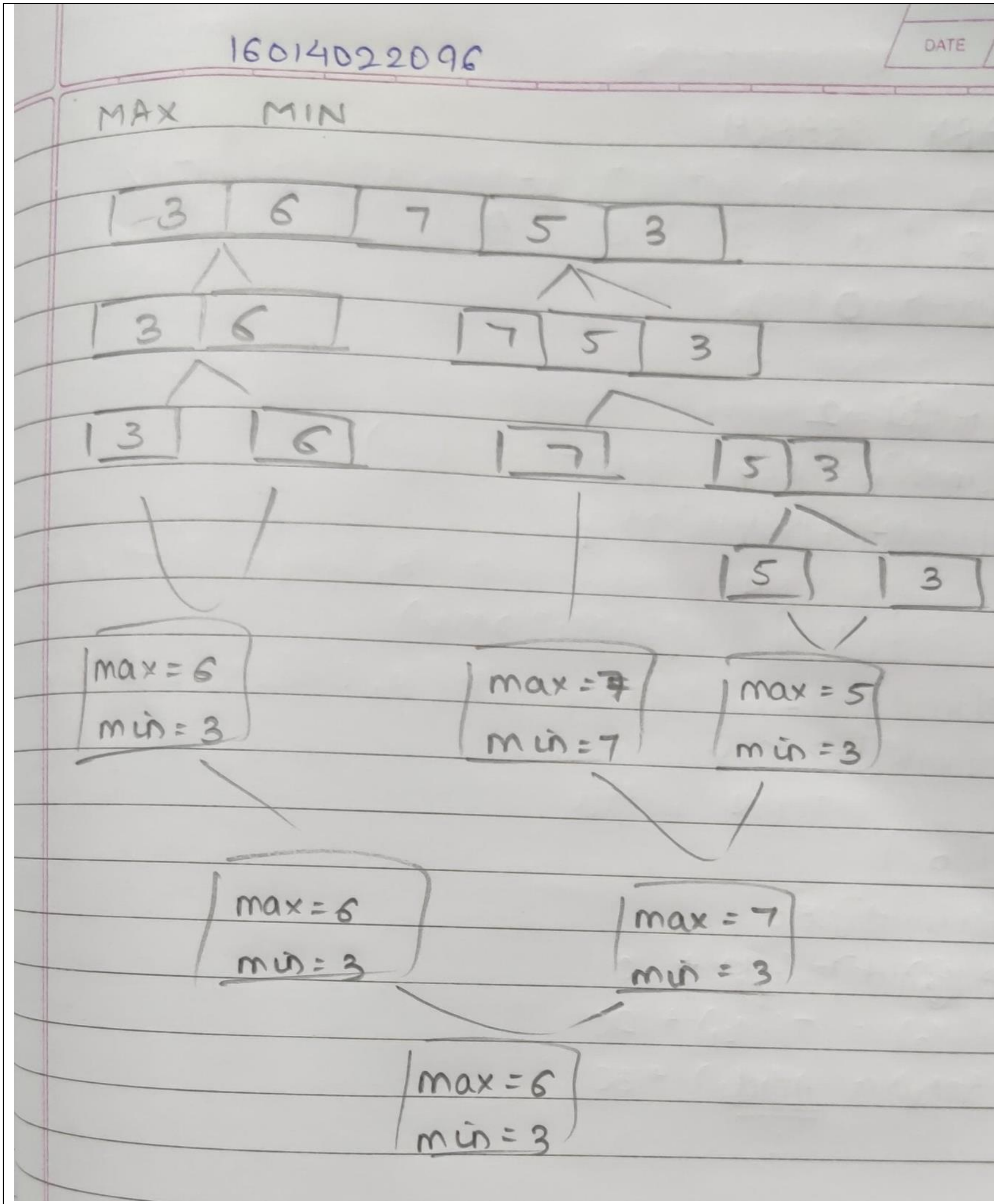
0	1

A[imid] = 1
A[imid] > key

0

A[imid] = 0
A[imid] = key

return imid


16014022096
DATE
MAX MIN
3 6 7 5 3
3 6
7 5 3
3
6
7
5 3
5
3
max = 6
min = 3
max = 7
min = 7
max = 5
min = 3
max = 6
min = 3
max = 7
min = 3
max = 6
min = 3

Conclusion:

We have successfully implemented divide and conquer technique to binary search algorithm and to max min algorithm.

Signature of faculty in-charge with Date: