K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch:-B-2 Roll No:-16010122151

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:

CO 2 Describe various algorithm design strategies to solve different problems and analyse 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/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

```
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:-

Code for Binary search Iterative Method:-

```cpp
#include <iostream>

using namespace std;

int main() {

    cout << "Enter the size of the sorted array: ";
    int size;
    cin >> size;


    int sorted_array[size];
```

```
    cout << "Enter the sorted array elements: ";
    for (int i = 0; i < size; ++i)
    {
        cin >> sorted_array[i];
    }


    cout << "Enter the target value: ";
    int target_value;
    cin >> target_value;


    int start = 0;
    int end = size - 1;
    int result = -1;

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

        if (sorted_array[mid] == target_value) {
            result = mid;
            break;
        } else if (sorted_array[mid] < target_value)
       {
            start = mid + 1;
        } else
        {
            end = mid - 1;
        }
    }


    if (result != -1)
    {
        cout << "Found "<< endl;
    } else
    {
        cout << " Not found " << endl;
    }

    return 0;
}
```

Output:-

For Iterative Method:-

```
Enter the size of the sorted array: 5
Enter the sorted array elements: 1 2 3 4 5
Enter the target value: 5
Found
```

```
Enter the size of the sorted array: 5
Enter the sorted array elements: 1 2 3 4 5
Enter the target value: 6
 Not found
```

For Recursive Method:-

```cpp
    #include <iostream>

using namespace std;

int binary_search_recursive(int arr[], int target, int start, int end)
  {
    if (start > end)
    {
        return -1;
    }

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

    if (arr[mid] == target)
    {
        return mid;
    }
    else if (arr[mid] < target)
    {
        return binary_search_recursive(arr, target, mid + 1, end);
    }
    else
    {
        return binary_search_recursive(arr, target, start, mid - 1);
```

```
    }
}

int main() {

    cout << "Enter the size of the sorted array: ";
    int size;
    cin >> size;


    int sorted_array[size];
    cout << "Enter the sorted array elements: ";
    for (int i = 0; i < size; ++i)
    {
        cin >> sorted_array[i];
    }


    cout << "Enter the target value: ";
    int target_value;
    cin >> target_value;


    int result = binary_search_recursive(sorted_array, target_value, 0, size -
1);


    if (result != -1)
    {
        cout << "Target " << target_value << " found at index " << result <<
endl;
    } else
    {
        cout << "Target " << target_value << " not found in the array" << endl;
    }

    return 0;
}
```

Output:-

For Recursive Method:-

```
Enter the size of the sorted array: 5
Enter the sorted array elements:  1 2 3 4 5
Enter the target value: 5
Target 5 found at index 4
```

```
Enter the size of the sorted array: 5
Enter the sorted array elements: 1 2 3 4 5
Enter the target value: 45
Target 45 not found in the array
```

Hyder Pasha
160

Binary Search

Iterative method :-

```
int start = 0, mid, end;
  end = n-1;
while( start <= end)
  {
mid = start + (end - start) /2;
if (arr[mid] == key)
  {
  cout << "Yes";
  }
if ( arr[mid] < key)
  {
    start = mid + 1;
  }
else
  {
    end = mid - 1;
  }
```

Time complexity :-

a) Best case : O(1)

Assuming mid is the key

b) Worst case :- $O(\log_2 n)$

Assuming key is at start or end of the array

c) Avg case : $O(\log_2 n)$

key is bet start & mid . or bet mid & end

so we have search half of $\frac{\log_2 n}{2}$

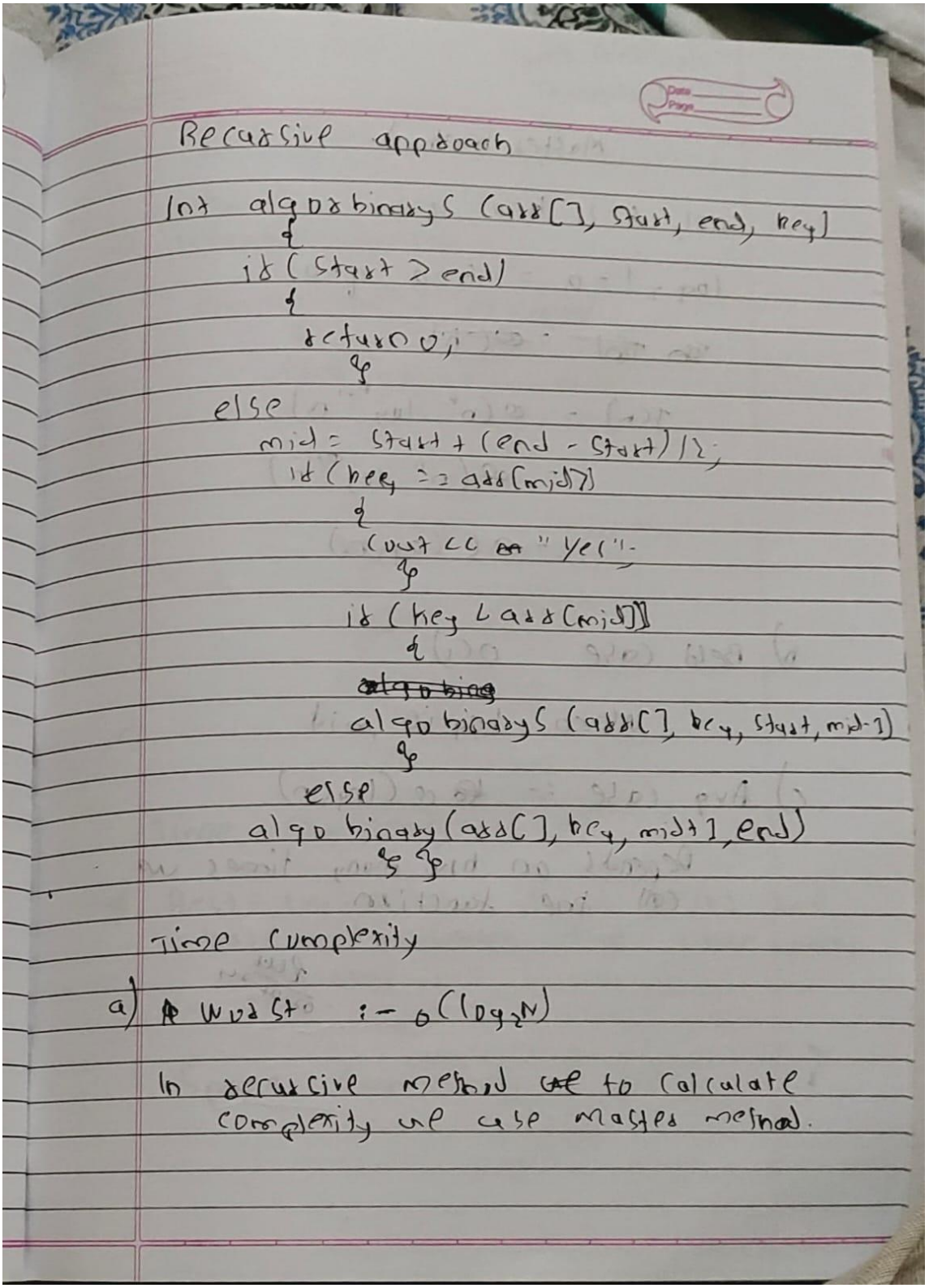
so $O(\log_2 n)$

Space complexity = $O(1)$

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;
            }
}
```

The space complexity of Recursive Binary Search:-



The

Time complexity of Binary Search:-

Master method.

$a=1,\ b=2,\ k=0,\ q=0$

$\log_2 1 = 0 = k = 0,\ q = 0$

~~T(n) = O(n^k~~

$T(n) = O(n^k \log^{q+1} n)$

$= O(n^0 \log^{0+1} n)$

$= O(\log_2 n)$

b) Best case $= O(1)$

When key is mid

c) Avg case $= O(\log_2 n)$

Depends on how many times we call the function.

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 sub problems. 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;

    }
  }
```

Code for MaxMin Iterative Method:-

```cpp
#include <iostream>

using namespace std;
```

```cpp
int main() {

    cout << "Enter the size of the array: ";
    int size;
    cin >> size;


    int arr[size];
    cout << "Enter the array elements: ";
    for (int i = 0; i < size; ++i)
    {
        cin >> arr[i];
    }


    int max_value = arr[0];
    int min_value = arr[0];


    for (int i = 1; i < size; ++i)
    {
        if (arr[i] > max_value)
        {
            max_value = arr[i];
        }
        if (arr[i] < min_value)
         {
            min_value = arr[i];
        }
    }


    cout << "Maximum value in the array: " << max_value << endl;
    cout << "Minimum value in the array: " << min_value << endl;

    return 0;
}
```

```
Enter the size of the array: 5
Enter the array elements: 10 20 65 90 100
Maximum value in the array: 100
Minimum value in the array: 10
```

```cpp
#include <iostream>

using namespace std;
```

```
void find_max_min_recursive(int arr[], int start, int end, int& max_value, int&
min_value) {

    if (start == end)
    {
        max_value = arr[start];
        min_value = arr[start];
        return;
    }


    int mid = start + (end - start) / 2;
    int max_left, min_left, max_right, min_right;

    find_max_min_recursive(arr, start, mid, max_left, min_left);


    find_max_min_recursive(arr, mid + 1, end, max_right, min_right);


    max_value = max(max_left, max_right);
    min_value = min(min_left, min_right);
}

int main()
 {

    cout << "Enter the size of the array: ";
    int size;
    cin >> size;


    int arr[size];
    cout << "Enter the array elements: ";
    for (int i = 0; i < size; ++i)
   {
        cin >> arr[i];
    }


    int max_value, min_value;


    find_max_min_recursive(arr, 0, size - 1, max_value, min_value);


    cout << "Maximum value in the array: " << max_value << endl;
```

```
    cout << "Minimum value in the array: " << min_value << endl;

    return 0;
}
```

```
Enter the size of the array: 5
Enter the array elements: 10 29 65 89 90
Maximum value in the array: 90
Minimum value in the array: 10
```

Hyder Rasool
16010122151

Min & max

-> Iterative approach

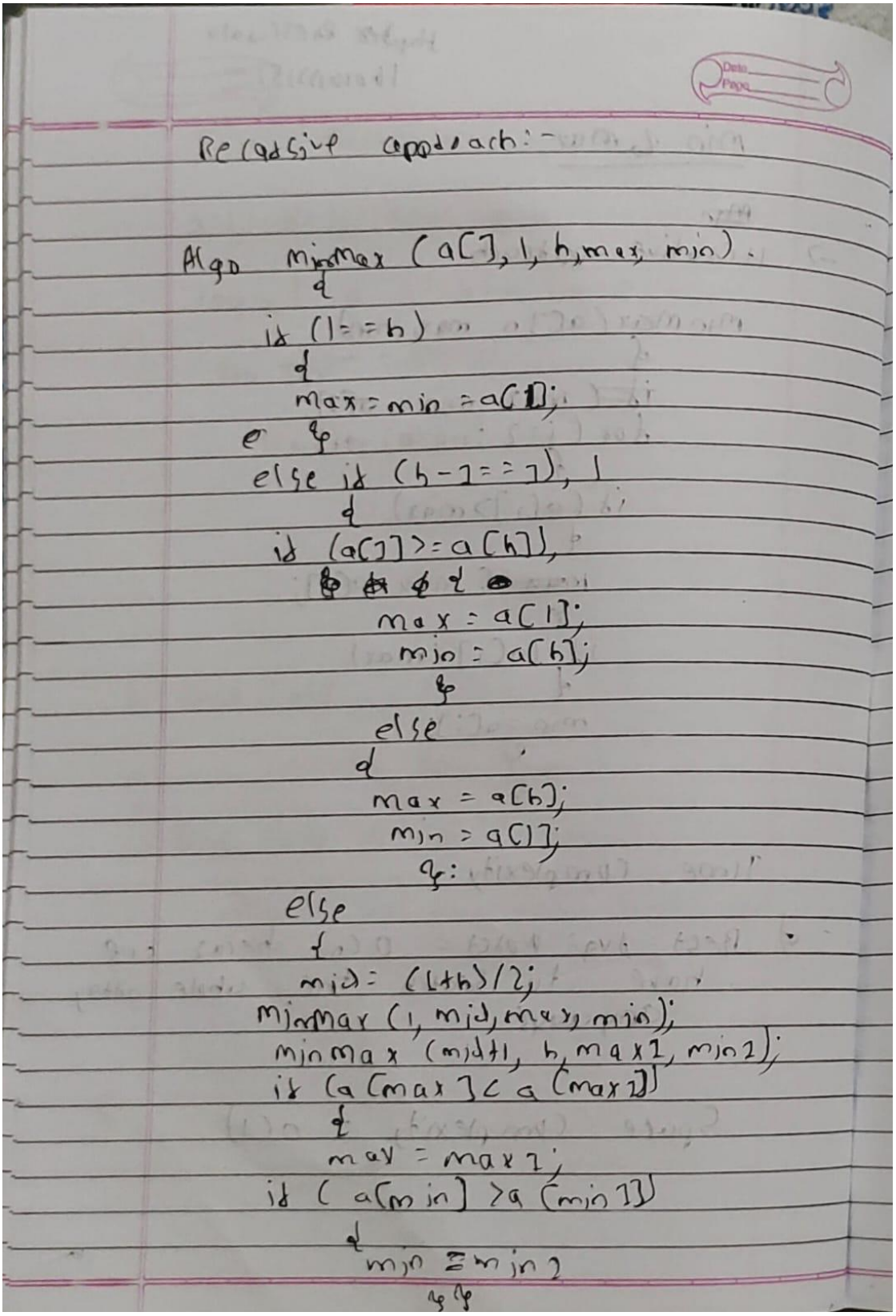
```
MinMax(a[], n, max, min)
{
if( max = min = a[1];
for( i=2 ; i<=n; i++);
  {
if (a[i]>max)
  {
max = a[i];
  }
if (a[i]<max)
  {
min = a[i];
  }
}
```

Time Complexity :-

Best = Avg = Worst = O(n) because we have to search the whole array anyways

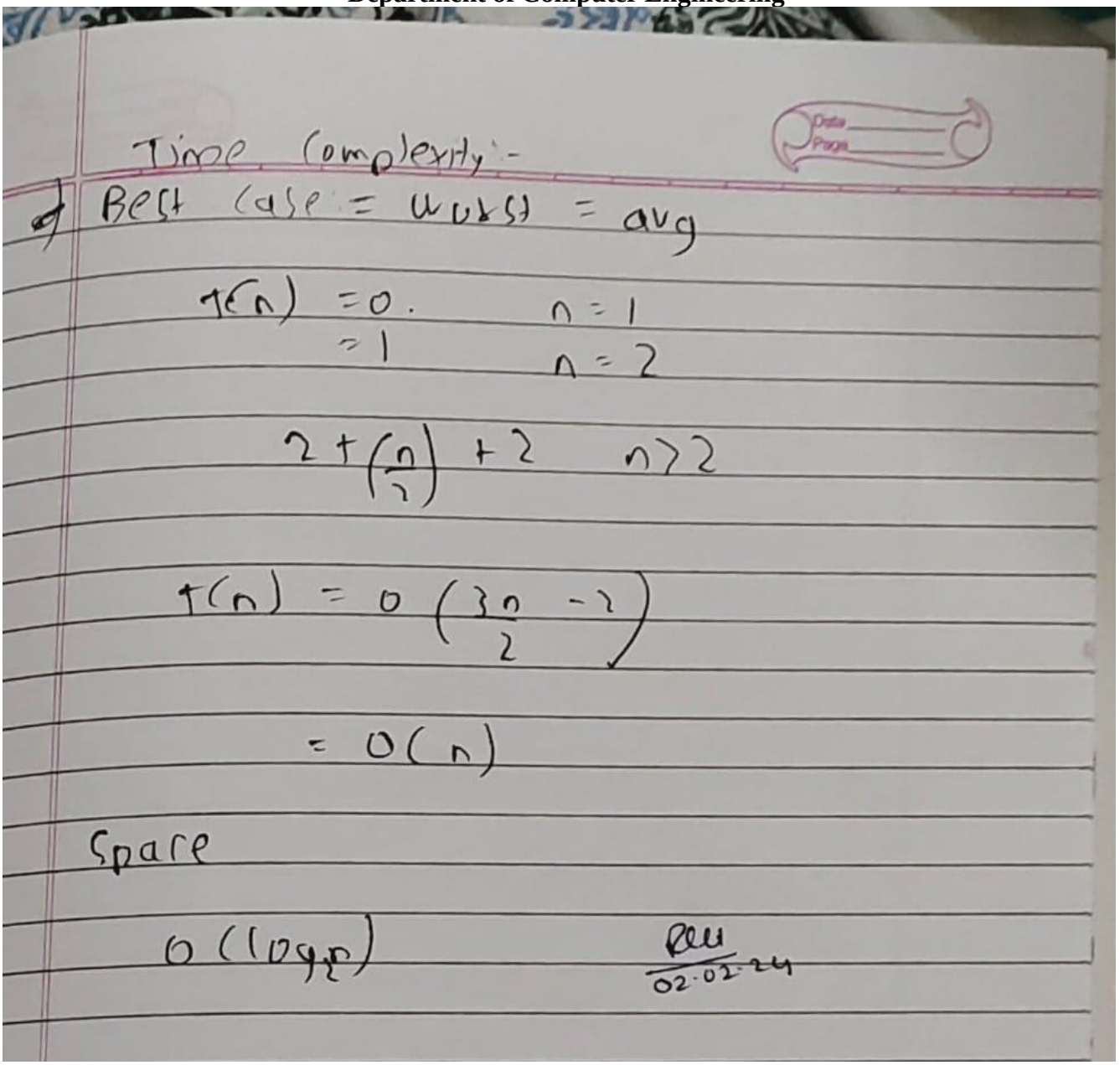
Space Complexity = O(1)

Time complexity for Max-Min:-



Recursive approach:-

```
Algo minmax (a[], l, h, max, min).
{
  if (l==h)
  {
    max=min=a[l];
  }
  else if (h-l==1), l
  {
    if (a[l]>=a[h]),
    {
      max = a[l];
      min = a[h];
    }
    else
    {
      max = a[h];
      min = a[l];
    }
  else
  {
    mid = (l+h)/2;
    minmax (l, mid, max, min);
    minmax (mid+1, h, max1, min1);
    if (a[max]<a[max1])
    {
      max = max1;
    if ( a[min] >a [min1])
    {
      min = min1
    }
  }
```



CONCLUSION:- Binary Search, the iterative way is memory-efficient but less intuitive, while the recursive way is more readable but can use more memory. In finding max and min, the iterative method is straightforward and efficient, while the recursive approach, though elegant, might use more memory, especially for large datasets. The decision depends on readability, familiarity, and efficiency for the specific problem and data size.