Design and Analysis of Algorithms (CS206)

LAB Assignment – 5

1.1. (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**.

HELP BOB!

Bob, the Builder is working in Skyline Real Estate Company, and his Company has assigned him Project 'DEMOLITION'.

In Project DEMOLITION, Skyline Real Estate Developers is planning to demolish a number of old, unoccupied buildings and construct a shopping mall in their place.

Bob's task is to find the largest solid area in which the mall can be constructed.
There are a number of buildings in a certain two-dimensional landscape.

Each building has a height, given by h[i] where i belongs [1, n].

If you Join k adjacent buildings, they will form solid rectangle of area:

$$k \times min(h[i], h[i+1], \ldots, h_[i+k-1]).$$

But, Bob would get Promotion, if the Area Demolished is Maximum!

Would you Help Bob to solve this Problem to get Maximum Area, he can demolished?

Input Format

Input File Containing space-separated integers, each representing the height of a building.

Constraints

$$1 \leq h[i] \leq 10^6$$

Output Format

Print long integer, representing the maximum area of rectangle that can be formed.
Remember that this Rectangle must be aligned at common base line.

For Example,

Sample Input:

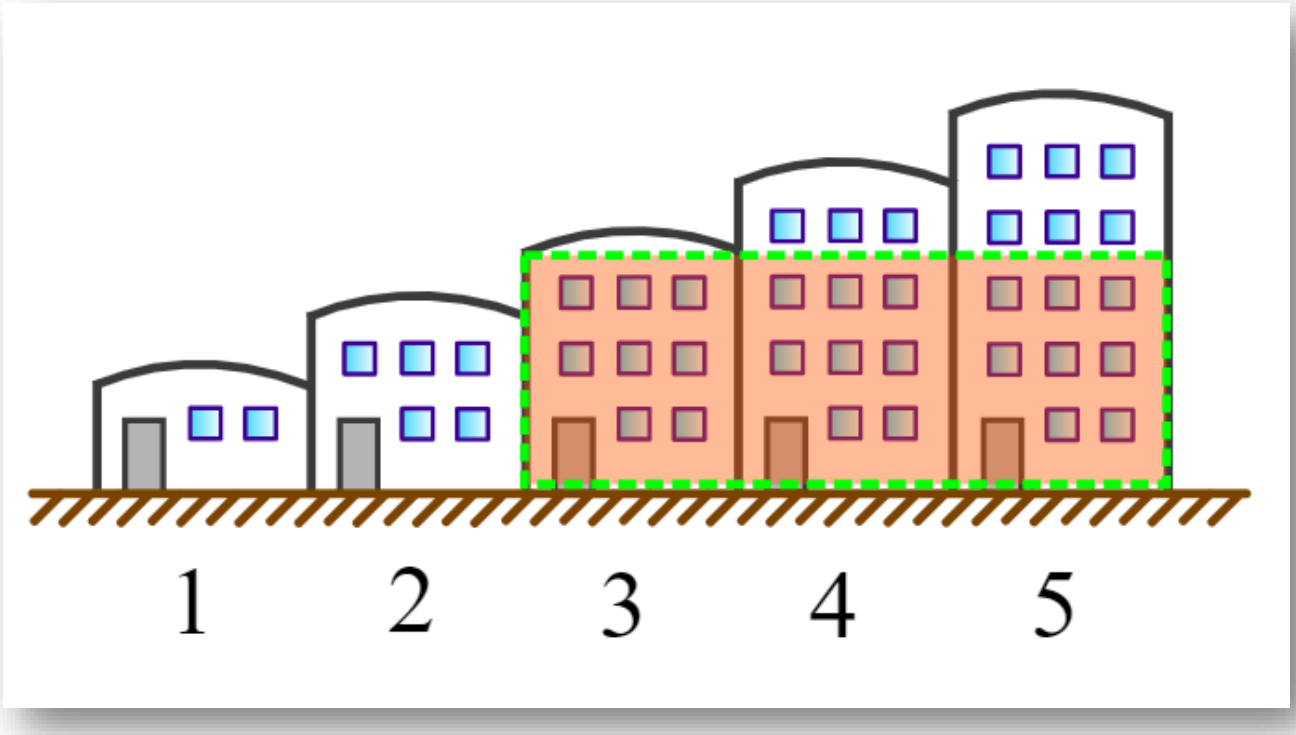
```
1 2 3 4 5
```

Sample Output:

```
9
```

Explanation:

Maximum Area {3,4,5} = 3*[3] = 9



1.2. (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.

Pseudo-Code INCREMENTAL

```
// Iterative Approach [PSUEDO CODE]

// heights : Array of Heights of Buildings

maxAreaIterative (heights)

1.  maxArea = 0
2.  minHeight = 0

3.  for i = 0 to height.length-1
// Area of Height of Building * Width [1]
4.      maxArea = max(heights[i], maxArea)
// Assume Current Height as Min Height
5.      minHeight = heights[i]
// Traverse on Left Side
6.      for j = i - 1 to 0
7.          minHeight = min(heights[j], minHeight)
8.          width = (i - j + 1)
9.          maxArea = max(maxArea, (minHeight * width))
// Final Answer containing Maximum Area
10.return maxArea
```

Pseudo-Code DIVIDE AND CONQUER

```
// DIVIDE AND CONQUER APPROACH

heights : Array Of Height of Buil
left    : Left Boundery
right   : right Boundary

maxAreaDnC ( heights, left, right)

// BASE CASE [1 Building]
1.  if left == right
        return heights[left]
// BASE CASE [2 Building]
2.  if left + 1 == right
3.      minH = min(heights[left], heights[right])
4.      return max( minH * 2, max(heights[left], heights[right]))

5.  mid = (left + right) / 2
// left max area
6.  leftArea = maxAreaDnC(heights, left, mid-1)
// right max area
7.  rightArea = maxAreaDnC(heights, mid+1, right)

// mid max area, including current bar
8.  i = mid, j = mid
9.  width, midArea = 0
10. height = heights[mid]
11.     while i >= left && j <= right

12.         width = j - i + 1
13.         height = min(height, min(heights[i], heights[j]))
14.         midArea = max(midArea, width * height)
// Go to Right Side
15.          if i == left
16.              j += 1
// Go to Left Side
17.         else if j == right
18.             i -= 1
// If Left Side Building is Taller
19.         else if heights[i-1] >= heights[j+1]
20.             i -= 1
// If Right Side Building is Taller
21.         else
22.             j += 1

// Return the Maximum of all three Parts of Area
23. return max( midArea, max(leftArea, rightArea) )
```

Analysis

(A) Iterative approach

	maxAreaIterative (heights)	Cost/oper^n	Best case	Worst case
①	maxArea = 0	c_1	1	1
②	minHeight = 0	c_2	1	1
③	for i = 0 to height.length - 1	c_3	2	n+1
④	maxArea = max(heights[i], maxArea)	c_4	1	n
⑤	minHeight = heights[i]	c_5	1	n
⑥	for j = i-1 to 0	c_6	2	$\sum_{j=0}^{n+1}(j)$
⑦	minHeight = min(heights[j], minHeight)	c_7	1	$\sum_{j=0}^{n}(j)$
⑧	width = i-j+1	c_8	1	
⑨	maxArea = max(maxArea, (minHeight × width))	c_9	1	
⑩	return maxArea	c_{10}	1	1

Best case: $T(n) = c_1 + c_2 + 2\times c_3 + c_4 + c_5 + c_6 \times 2 + c_7 + c_8 + c_9 + c_{10}$

$= c$ (c = constant)

$= \boxed{O(1)}$

Worst case: $T(n) = c_1 + c_2 + c_3 \times (n+1) + (c_4 + c_5) \times n + \sum_{j=0}^{n+1} j\,(c_6) + (c_7 + c_8 + c_9) \sum_{j=0}^{n} j + c_{10}$

$= c_1 + c_2 + c_3 + c_{10} + n(c_3 + c_5 + c_4) + \frac{n(n+1)}{2}(c_7 + c_8 + c_9) + \frac{(n+1)(n+2)}{2} c_6$

$= c_A + n(c_B) + \frac{n^2}{2}(c_c) + \frac{n}{2}(c_c) + \frac{n^2}{2}(c_6) + \frac{n}{2}(c_6)$

$= \left(\frac{c_c + c_6}{2}\right) n^2 + \left(\frac{c_c}{2} + \frac{c_6}{2} + c_B\right) n + c_A$ c_A, c_B, c_c, c_6 = constant

$= An^2 + Bn + C = \boxed{O(n^2)}$

vision

(B) Divide and Conquer Approach

[n : size of heights array]

let time take be T(n)

	maxAreaDnC (heights, left, right)	Cost	Best case	Worst case
①	if left == right	c_1	1	1
②	return heights [left]	c_2	1	0
③	if left+1 == right	c_3	1	1
④	minH = min (heights [left], heights [right])	c_4	1	0
⑤	return max (minH * 2, max (heights[left], heights[right]))	c_5	1	0
⑥	mid = (left + right) / 2	c_6	1	1
⑦	leftArea = maxAreaDnC (heights, left, mid-1)	c_7	T(n/2)	
⑧	rightArea = maxAreaDnC (heights, mid+1, right)	c_8	T(n/2)	
⑨	i = mid, j = mid	c_9	1	1
⑩	width, midArea = 0	c_{10}	1	1
⑪	height = heights[mid]	c_{11}	1	1
⑫	While i>=left && j<=right	c_{12}	2	n+1
⑬	width = j-i+1	c_{13}	1	n
⑭	height = min (height, min(height[i], height[j]))	c_{14}	1	n
⑮	midArea = max (midArea, width * height)	c_{15}	1	n
⑯	if i == left j++	c_{16}	1	n
⑰	else if j == right i--	c_{17}	1	n
⑱	else if heights[i-1] >= heights[j+1] i--	c_{18}	1	n
⑲	else j++	c_{19}	1	n
⑳	return max ({ leftArea, rightArea, midArea })	c_{20}	1	1

vision

(A) Best case : $T(n) = (C_1+C_2+C_3+C_4+C_5+C_6)\,1 + 2T\left(\frac{n}{2}\right) + (C_9+C_{10}+C_{11} + 2C_{12}+C_{13} + C_{14}+C_{15}+C_{16}+C_{17}+C_{18}+C_{19}+C_{20})\times 1$

$$\left[T(n) = C + 2T\left(\frac{n}{2}\right)\right]$$ – Best case recurrence relation

Using Master theorem, $T(n) = \theta(n^{\log_2 2})$

$$\boxed{T(n) = \theta(n)} \quad \text{——①}$$

(B) Worst-Case : $T(n) = (C_1+C_3+C_6) + 2T\left(\frac{n}{2}\right) + (C_9+C_{10}+C_{11}) + C_{12}\times(n+1) + (C_{13}+C_{14}+C_{15}+C_{16})n + C_{20}$

$$= C_A + (C_B)(n) + 2T\left(\frac{n}{2}\right)$$

$$T(n) = 2T\left(\frac{n}{2}\right) + \theta(n)$$

Using master theorem, $T(n) = \theta(n^{\log_2 2}\ (\log^{(0+1)}(n)))$

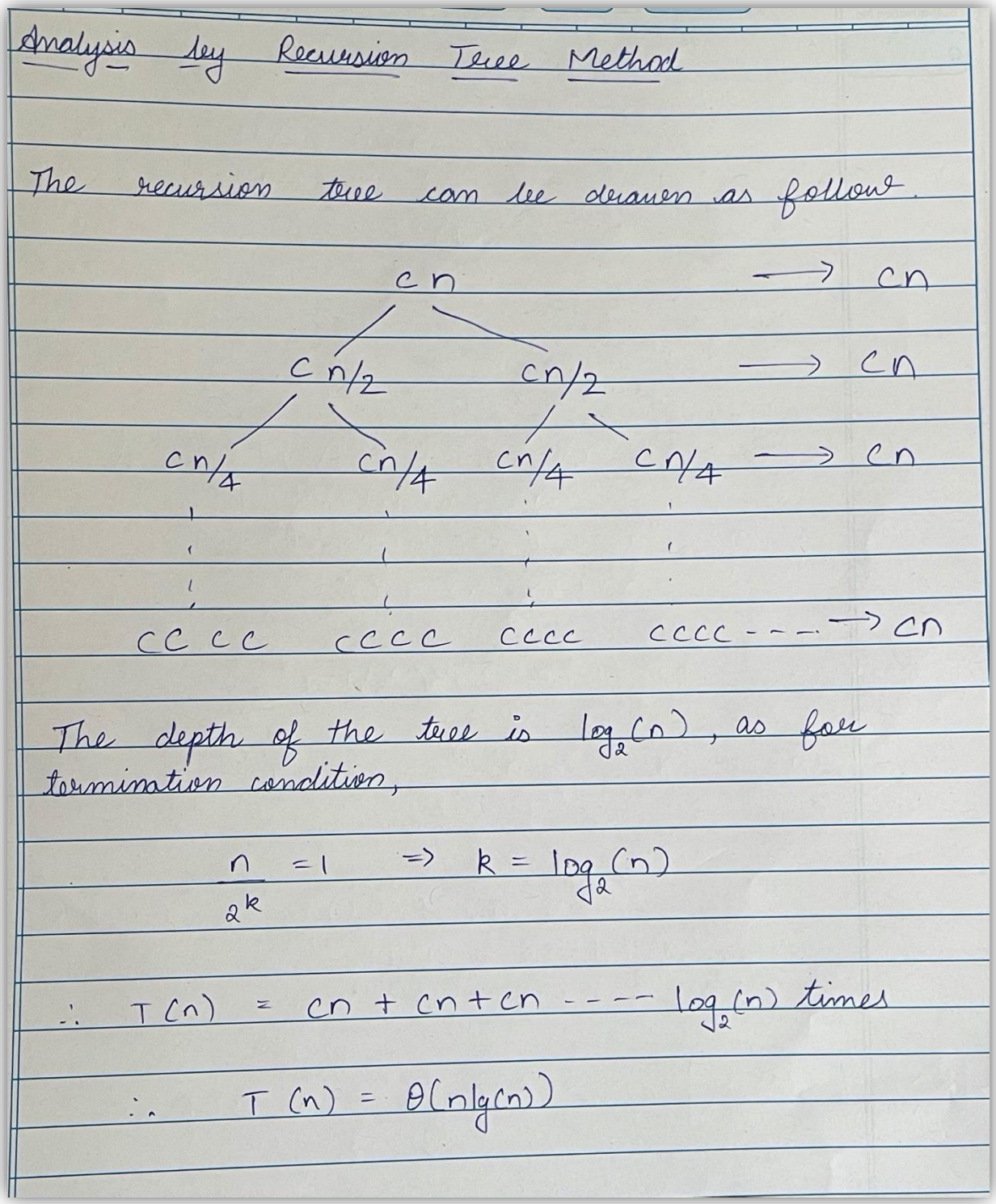
$$\boxed{T(n) = \theta(n \log(n))} \quad \text{——②}$$

vision

1.3. (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

Analysis by Iterative method

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

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

$T(n/4) = 2T(n/8) + cn/4$

⋮

$T(1) = c \rightarrow$ Termination condition.

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

$= 4T(n/4) + cn + cn$

$= 4(2T(n/8) + cn/4) + cn + cn$

$= 8T(n/8) + cn + cn + cn$

⋮

$= 2^k T(n/2^k) + cn(1+1+1 \text{ ---- } k \text{ times})$

let $\frac{n}{2^k} = 1 \Rightarrow 2^k = n \Rightarrow k = \log_2(n)$

$\therefore T(n) = 2^k T(1) + cn \cdot k$

$= 2^{\log_2(n)} T(1) + cn \cdot \log_2(n)$

$= n \cdot c + cn \cdot \log_2(n)$

$\therefore T(n) = \theta(n \lg(n))$ $\quad [\lg(n) = \log_2(n)]$

(3) Master Method

Analysis by Master's theorem

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

Here, $a = 2$, $b = 2$ and $f(n) = cn$

$f(n) = cn = \theta(n^{\log_2 2}) = \theta(n)$

∴ we can say that,

$T(n) = \theta(n^{\log_2 2} \lg(n))$

$\therefore T(n) = \theta(n \lg(n))$

(4) Substitution Method.

Analysis by substitution method

Assume that $T(n) = \theta(n \lg(n))$

Now, $T(n) = 2T(n/2) + cn$

$\therefore T(n) \leq 2(cn/2) \lg(n/2) + cn$

$\Rightarrow T(n) \leq cn[\lg(n) - \lg(2)] + cn$

$\Rightarrow T(n) \leq cn\lg(n) - cn + cn$

$\Rightarrow T(n) \leq cn\lg(n)$

$\therefore T(n) = \theta(n \lg(n))$

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

Hardware Details of Laptop used for testing:

PARAMETER	LAPTOP CONFIGURATION
Operating System	Microsoft Windows **10 v-**20H2
Processor	Intel(R) Core(TM) i5-9300H [Core **i5 9th Gen**]
CPU	**2.30GHz(base)**, **4GHz(boost)**, **4** Core(s), 8 Logical Processor(s)
System Type	x64-based PC [**64 Bit**]
RAM	**8.00** GB
Hard Drive/SSD	512 GB **SSD**

Software Used:

PARAMETER	LAPTOP CONFIGURATION
Code Editor	**Intellij IDEA Community Edition 2020.3.1**
Compiler	jdk-13.0.2
Time	Measured using **System.nanotime()**
Programming Language Used	**Java**

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

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

public class maxAreaHistogram {

    public static void main(String[] args) {
        try {
            File output = new File("output.txt");
            output.createNewFile();
            FileWriter writer = new FileWriter(output);
            ArrayList<Integer> l = new ArrayList<>();
            long startTime, endTime, timeTaken;
            String file;
            for (int i = 1; i < 4; i++) {
                file = String.format("File %d.txt", i);
```

```
                try {
                    loadFile(l, file);
                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                }
                int[] heights = l.stream()
                .mapToInt(Integer::intValue)
                .toArray();
                int size = heights.length;
                int ans = 0;
                writer.write("File " + i + "\n");
                System.out.println("File " + i);

                // Iterative
                timeTaken = 0;
                startTime = System.nanoTime();
                for (int j = 0; j < 20; j++) {
                    ans = maxAreaIterative(heights);
                }
                endTime = System.nanoTime();
                timeTaken = (endTime - startTime);
                writer.write(ans + ", ");
                System.out.print(ans + ", ");
                writer.write(timeTaken / 20 + " ns, ");
                System.out.print(timeTaken / 20 + " ns, ");

                // Divide and conquer
                timeTaken = 0;
                startTime = System.nanoTime();
                for (int j = 0; j < 20; j++) {
                    ans = maxAreaDnC(heights, 0, size - 1);
                }
                endTime = System.nanoTime();
                timeTaken = (endTime - startTime);
                writer.write(ans + ", ");
                System.out.print(ans + ", ");
                writer.write(timeTaken / 20 + " ns\n");
                System.out.println(timeTaken / 20 + " ns");
                l.clear();
            }
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void loadFile(ArrayList<Integer> A, String file) throws FileNotFoundExcepti
on {
        Scanner s = new Scanner(new File(file));
        int temp;
```

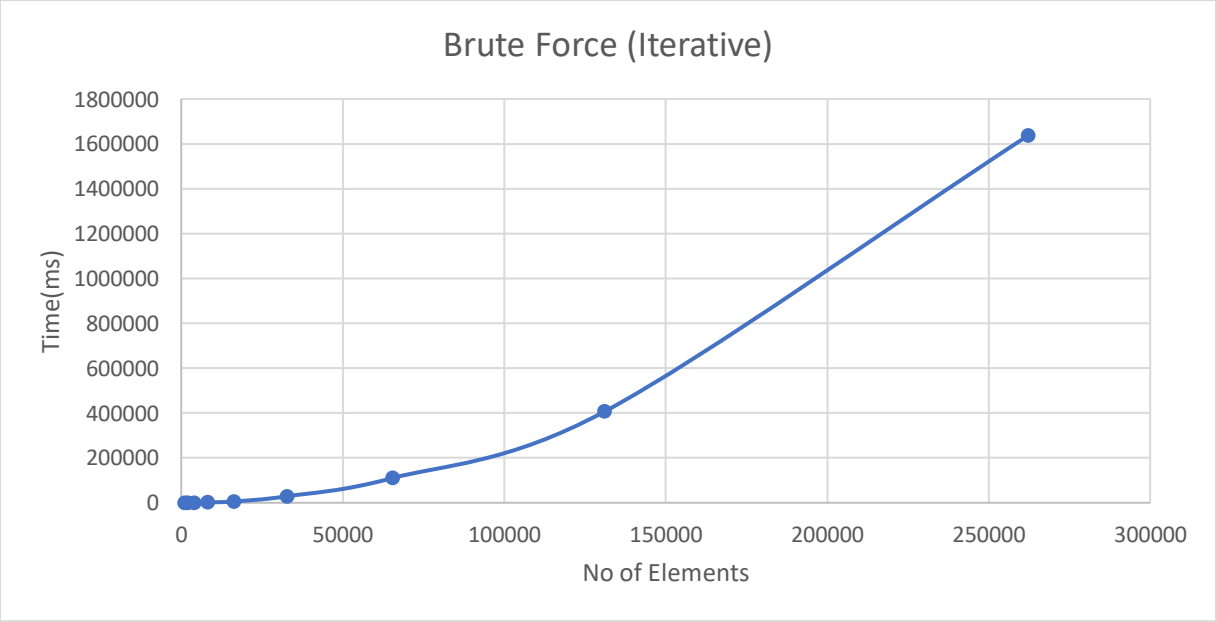
```
        while (s.hasNext()) {
            temp = s.nextInt();
            A.add(temp);
        }
    }

    public static int maxAreaIterative(int[] heights) {
        int maxArea = 0;
        int minHeight = 0;
        for (int i = 0; i < heights.length; i++) {
            maxArea = Math.max(heights[i], maxArea);
            minHeight = heights[i];
            for(int j = i - 1; j >= 0 ; j--) {
                minHeight = Math.min(heights[j], minHeight);
                int width = (i - j + 1);
                maxArea = Math.max(maxArea, (minHeight * width));
            }
        }
        return maxArea;
    }

    public static int maxAreaDnC(int[] heights, int left, int right) {
        if (left == right) {
            return heights[left];
        }
        if (left + 1 == right) {
            int minH = Math.min(heights[left], heights[right]);
            return Math.max(minH * 2, Math.max(heights[left], heights[right]));
        }
        int mid = (left + right) / 2;
        // left max area
        int leftArea = maxAreaDnC(heights, left, mid-1);
        // right max area
        int rightArea = maxAreaDnC(heights, mid+1, right);
        // mid max area, including current bar
        int i = mid, j = mid;
        int width, midArea = 0;
        int height = heights[mid];
        while (i >= left && j <= right) {
            width = j - i + 1;
            height = Math.min(height, Math.min(heights[i], heights[j]));
            midArea = Math.max(midArea, width * height);
            if (i == left) {
                j += 1;
            } else if (j == right) {
                i -= 1;
            } else if (heights[i-1] >= heights[j+1]) {
                i -= 1;
            } else {
                j += 1;
```

```
            }
        }
        return Math.max( midArea, Math.max(leftArea, rightArea) );
    }
}
```

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

BRUTE FORCE (ITERATIVE)

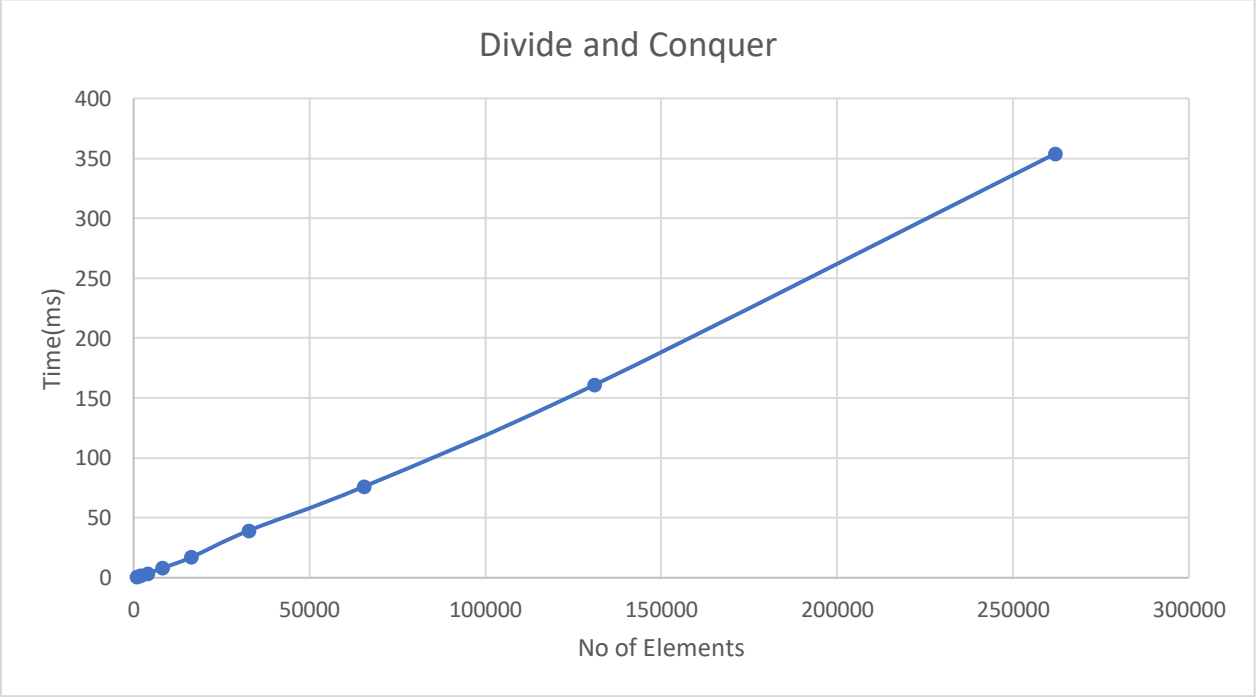
Sr. No.	No of Elements	Brute Force (ms)
1	1024	21.51845
2	2048	96.03287
3	4096	424.20708
4	8192	1539.63372
5	16384	5947.25934
6	32768	29003.11334
7	65536	110500.4987
8	131072	406734.3615
9	262144	1636508.868



⇨ Here $T(n) = 0.0000236585*n^2 + 0.0359048*n + 473.34$

DIVIDE AND CONQUER

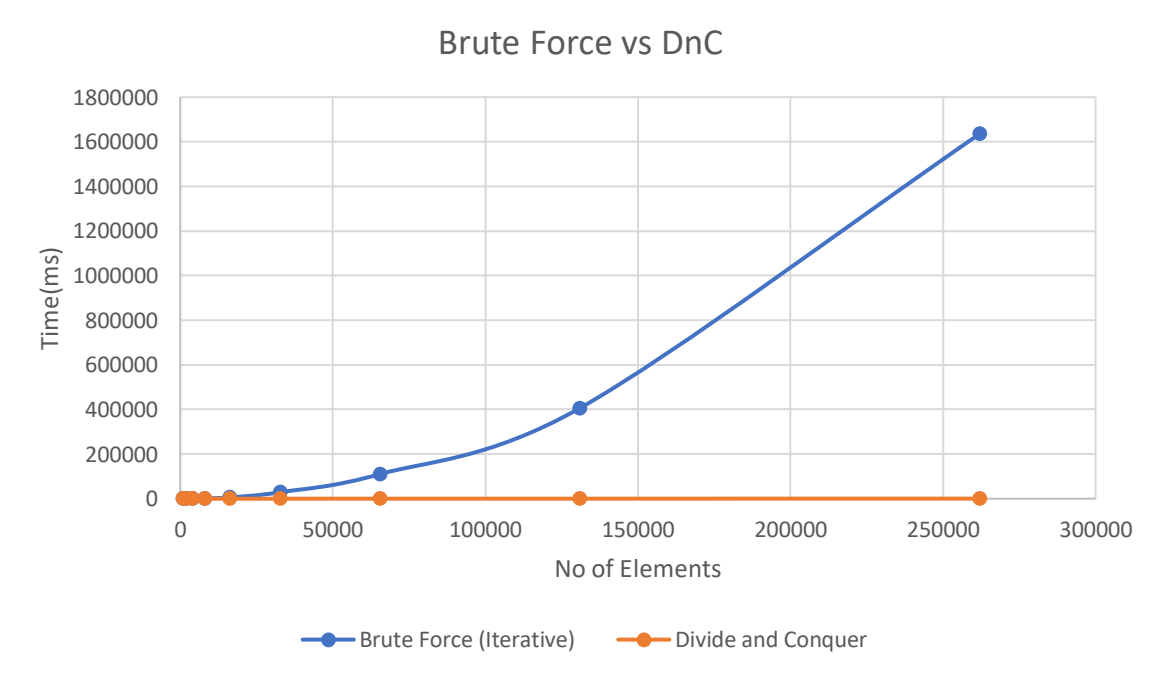
Sr. No.	No of Elements	Divide and Conquer (ms)
1	1024	0.88949
2	2048	1.66137
3	4096	3.6958
4	8192	8.0214
5	16384	17.09201
6	32768	39.39826
7	65536	76.16316
8	131072	161.02624
9	262144	354.04978



⇨ Here $T(n) = 0.00010009*n*\log_2 n - 0.000459491*n + 1.2401$

1.7. (L) Comparatively Analyze the performance of above algorithms and plot a graph.

Sr. No.	No of Elements	Brute Force (ms)	Divide and Conquer (ms)
1	1024	21.51845	0.88949
2	2048	96.03287	1.66137
3	4096	424.20708	3.6958
4	8192	1539.63372	8.0214
5	16384	5947.25934	17.09201
6	32768	29003.11334	39.39826
7	65536	110500.4987	76.16316
8	131072	406734.3615	161.02624
9	262144	1636508.868	354.04978



SUBMITTED BY:

Sr. No.	Admission No.
1	U19CS011
2	U19CS012
3	U19CS049
4	U19CS080