PRACTICAL - 5

GIVEN AN ARRAY OF INTEGERS HEIGHTS REPRESENTING THE HISTOGRAM'S BAR HEIGHT WHERE THE WIDTH OF EACH BAR IS 1, RETURN THE AREA OF THE LARGEST RECTANGLE IN THE HISTOGRAM.

```c
#include <stdio.h>
#define MAX(a, b) ((a) > (b) ? (a) : (b))

int largest_rectangle_area(int heights[], int n) {
    int stack[n];
    int top = -1;
    int max_area = 0;
    int i = 0;

    while (i < n) {
        if (top == -1 || heights[i] >= heights[stack[top]]) {
            stack[++top] = i;
            i++;
        }
        else {
            int top_of_stack = stack[top--];
            int area = heights[top_of_stack] * (top == -1 ? i : i - stack[top] - 1);
            max_area = MAX(max_area, area);
        }
    }

    while (top != -1) {
        int top_of_stack = stack[top--];
        int area = heights[top_of_stack] * (top == -1 ? i : i - stack[top] - 1);
        max_area = MAX(max_area, area);
    }

    return max_area;
}

int main() {
    int histogram[] = {2, 1, 5, 6, 2, 3};
    int size = sizeof(histogram) / sizeof(histogram[0]);
    int area = largest_rectangle_area(histogram, size);
    printf("Largest rectangle area: %d\n", area);  // Output: 10

    return 0;
}
```

```
#include <stdio.h>
#define MAX(a, b) ((a) > (b) ? (a) : (b))
```

1. Here, we include the necessary header file <stdio.h> for standard input and output operations. We also define a macro MAX(a, b) that will be used to find the maximum of two numbers a and b.

```
int largest_rectangle_area(int heights[], int n) {
```

2. We define a function 'largest_rectangle_area' that takes two arguments: an array of integers 'heights' representing the heights of bars in a histogram and an integer 'n' representing the number of elements in the array.

```
int stack[n];
int top = -1;
int max_area = 0;
int i = 0;
```

3. Inside the function, we declare some variables:
   'stack[n]': An integer array called stack to store indices of the heights array.
   'top': An integer variable initialized to -1, which will be used to keep track of the top of the stack.
   'max_area': An integer variable initialized to 0, which will store the maximum rectangle area found.
   'I': An integer variable initialized to 0, which will be used to traverse the heights array.

```
while (i < n) {
```

4. We start a while loop to iterate through the heights array. This loop continues until we reach the end of the array.

```
if (top == -1 || heights[i] >= heights[stack[top]]) {
```

5. Inside the loop, we check if the stack is empty (top is -1) or if the current height (heights[i]) is greater than or equal to the height at the index stored in the top of the stack. If either condition is true, we push the current index i onto the stack and increment i.

```
stack[++top] = i;
            i++;
        }
        else {
            int top_of_stack = stack[top--];
            int area = heights[top_of_stack] * (top == -1 ? i : i - stack[top] - 1);
            max_area = MAX(max_area, area);
        }
    }
```

6. If the current height is less than the height at the top of the stack, we enter the else block. We pop the top of the stack and calculate the area of the rectangle formed using the popped height. The area is calculated by multiplying the popped height by the width of the rectangle, which is (i - stack[top] - 1).

7. We update max_area by taking the maximum of the current max_area and the newly calculated area.

8. The loop continues until we have processed all heights in the heights array.

```
while (top != -1) {
```

9. After processing all heights, we enter another loop to handle any remaining elements in the stack.

```
int top_of_stack = stack[top--];
        int area = heights[top_of_stack] * (top == -1 ? i : i - stack[top] - 1);
        max_area = MAX(max_area, area);
    }
```

10. Similar to the previous loop, we calculate the area for the remaining elements in the stack and update ‘max_area’.

```
return max_area;
}
```

11. Finally, the function returns max_area, which represents the largest rectangle area in the histogram.

Summary

1. Includes the standard input/output library (<stdio.h>).
2. Defines a macro MAX(a, b) to find the maximum of two values.
3. Defines the largest_rectangle_area function, which takes an array of histogram heights and its size as input and returns the largest rectangle area.
4. Initializes an integer stack, top, and max_area.
5. Initializes a variable i to traverse the histogram.
6. The while loop iterates through the histogram:
   If the stack is empty or the current height is greater than or equal to the height at the top of the stack, it pushes the current index onto the stack and increments i.
   If the current height is less than the height at the top of the stack, it calculates the area using the popped height and updates max_area.
7. After the loop, it processes any remaining elements in the stack.
8. Finally, the main function defines the histogram, calculates its size, calls largest_rectangle_area to find the largest rectangle area, and prints the result.