Luis Gascon, Ethan Webb, Femi Dosumu
COSC 336
March 29, 2023

Assignment 4

Exercise 1

For each of the following functions, give a $\Theta(t(n))$ estimation with the simplest possible $t(n)$.

1. $13n^2 - 2n + 56 = \Theta(n^2)$
2. $2.5 \log(n) + 2 = \Theta(\log n)$
3. $n(12 + \log n) = \Theta(n \log n)$
4. $1 + 2 + 3 + \ldots + 2n = \Theta(n^2)$
5. $1 + 2 + 3 + \ldots + n^2 = \Theta(n^3)$
6. $\log(n^3) + 10 = \Theta(\log n)$
7. $\log(n^3) + n \log n = \Theta(n \log n)$
8. $n \log(n^3) + n \log n = \Theta(n \log n)$
9. $2^{2 \log n} + 5n + 1 = \Theta(n^2)$

Exercise 2

1. Evaluate the following postfix arithmetic expression: $10\ 3\ 4 - 5 * /$

$$-2$$

2. Convert the following infix arithmetic expression to postfix notation: $(((2+3)*5)-15)$

$$2 \quad 3 \quad + \quad 5 \quad * \quad 15 \quad -$$

Exercise 3

Consider the following algorithms A and B for the problem of computing $2^n \pmod{317}$

```
Algorithm A.

mod_exp_A(n) {
   if (n == 0) return 1;
   else {
          t = mod_exp_A(n/2);
          if (n is even) return t*t (mod 317);
          if (n is odd) return t*t*2 (mod 317);
}}
```

```
Algorithm B.

mod_exp_B(n) {
   if (n == 0) return 1;
   else {
          if (n is even)
            return mod_exp_B(n/2) * mod_exp_B(n/2) (mod 317);
          if (n is odd)
            return mod_exp_B(n/2) * mod_exp_B(n/2) *2 (mod 317);
}}
```

1. Write the reucrrence for the runtime $T_A(n)$ of algorithm A and solve the recurrence to find a $\Theta(\cdot)$ estimation of $T_A(n)$

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

2. Write the recurrence for the runtime $T_B(n)$ of algorithm B, and solve the recurrence to find a $\Theta(\cdot)$ estimation of $T_B(n)$.

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

3. Which algorithm is faster?

$$T_A(n) = T\left(\frac{n}{2}\right) + 2 \qquad\qquad T_B(n) = T\left(\frac{n}{2} + 2\right) + 2$$
$$a = 1 \quad b = 2 \quad f(n) = 2 \qquad\qquad a = 2 \quad b = 2 \quad f(n) = 2$$
$$n^{\log_2 1} \quad \text{vs.} \quad 2 \qquad\qquad n^{\log_2 2} \quad \text{vs.} \quad 2$$
$$T_A(n) = \Theta(\log n) \qquad\qquad T_B(n) = \Theta(n)$$

$$\therefore T_A(n) \text{ is the faster algorithm.}$$

Exercise 4

Give a $\Theta(\cdot)$ evaluation for the runtime of the following code:

```
i= 1; x=0;
while(i <= n) {
      j=1;
      while (j <= i) { x=x+1; j= 2*j; }
      i= 2*i;
}
```

Assume that n is a power two. Then i from the outer loop takes successively the values: $1, \quad 2, \quad 2^2, \quad 2^3, \quad \ldots, 2^{\log n}$

$$\Theta(\log^2 n)$$

The getMaxArea method finds the largest rectangular area possible in a given histogram. The histogram is represented by an array of heights (arr) and an array of corresponding widths. The variable n represents the length of both arrays. The algorithm starts by creating an empty stack of integers and pushes the indices onto it. It then initializes two arrays, left_smaller and right_smaller, with default values of -1 and n, respectively, that represents the left and right sides of each histogram bar. For each element arr[i], it pops elements from the top of the stack until the top element is smaller than arr[i] or the stack is empty. For each popped element, it sets the corresponding index in the right_smaller array to i-1. It then sets the value of left_smaller[i] to either the index at the top of the stack or i if the stack is empty. Finally, it pushes i onto the stack. After the loop completes, the method calculates the area of the largest rectangle by iterating over each element of arr and calculating the area of the rectangle with that element as the height. It uses the left_smaller and right_smaller arrays to determine the width of the rectangle. It keeps track of the largest area it has found so far and returns the maximum area within the histogram.

Input	Output
input-4.1	26
input-4.2	12
input-4.3	12
input-4.4	1,474,869

```
1 public static int getMaxArea(String file) throws
    FileNotFoundException {
2     int n = 0;
3     int count = 0;
4     Scanner inputFile = new Scanner(new File(file));
5
6     n = inputFile.nextInt();
7     inputFile.nextLine();
```

```
    int[] histX = new int[n];
    int[] arr = new int[n];

    while (inputFile.hasNext()) {
      histX[count] = inputFile.nextInt();
      arr[count] = inputFile.nextInt();
      count++;
    }

    Stack<Integer> s = new Stack<>();
    s.push(0);
    int max_area = arr[0];
    int left_smaller[] = new int[n];
    int right_smaller[] = new int[n];
    for (int i = 0; i < n; i++) {
      left_smaller[i] = -1;
      right_smaller[i] = n;
    }

    int i = 0;
    while (i < arr.length) {
      while (!s.empty() && s.peek() != -1 && arr[i] < arr[s.peek
  ()]) {
        right_smaller[s.peek()] = (int) i - 1;
        s.pop();
      }
      if (i > 0 && arr[i] == arr[(i - 1)]) {
        left_smaller[i] = left_smaller[(int) (i - 1)];
      } else {
        left_smaller[i] = (s.peek() == -1 ? i : s.peek());
      }
      s.push(i);
      i++;
    }

    for (i = 0; i < n; i++) {
      max_area = Math.max(max_area,
          arr[i] * ((right_smaller[i] == histX.length ? histX[
   right_smaller[i] - 1] : histX[right_smaller[i]])
              - histX[left_smaller[i]]));
    }
    return max_area;
  }
```