Tamir Krief, Iaian Milton, Blessing Abumere
COSC 336
9/19/2024

Assignment 2

Exercise 1.

(a) $(4n+1)4^{log(n)} = \Theta(n^3)$

(b)

$$t_1(n) = (n^2+n), t_2(n) = (n^2)$$

$$t_1(n) = \Theta(n^2)$$
$$t_2(n) = \Theta(n^2)$$
$$t_1(n) - t_2(n) = o(n^2)$$

(c) $t_3(n) = (2^n)$

(d) $t_4(n) = (n^2)$

Exercise 2:

A	B	O	o	Ω	ω	Θ
$lg^k n$	n^ϵ	yes	*yes*	*no*	*no*	*no*
n^k	c^n	*yes*	*yes*	*no*	*no*	*no*
$\sqrt{n}$	$n^{\sin n}$	no	*no*	*no*	*no*	*no*
2^n	$2^{n/2}$	no	*no*	*yes*	*yes*	*no*
n^{lgc}	c^{lgn}	yes	*no*	*yes*	*no*	*yes*
$lg(n!)$	$lg(n^n)$	yes	*no*	*yes*	*no*	*yes*

Exercise 3.

(a) $\Theta(n^3)$

(b) $\Theta(n)$

(c) $\Theta(n^4)$

(d) $\Theta(\log n)$

(e) $\Theta(n \log n)$

Exercise 4.

(a) The sum of all integers from 500 to 999 is 374,750.

(b) The sum of all odd integers from 1 to 999 is 250,000.

(c) There are 27,405 possible committees.

(d) The estimation $o(n^3)$ is incorrect for the problem. To find the number of ways 4 people can be chosen out of n people, we use the combination formula which comes to $\frac{n!}{4!(n-4)!}$. Analyzing the asymptotic behavior, the formula comes to $\frac{n^4}{24}$ meaning this is equal to $\Theta(n^4)$. Compared to the asymptotic behavior of $o(n^3)$, $\Theta(n^4)$ grows faster. So the problem's asymptotic behavior is not equal to $o(n^3)$.

Exercise 5.

A $\Theta(.)$ evaluation for the sum S is $\Theta(n^{7/2})$. Using the integral method, the sum S $= \sum_{k=1}^{n} k^2\sqrt{k}$ comes out to S $= \sum_{k=1}^{n} k^{5/2}$. For the integral, $\int_1^n k^{5/2}\,dk = \frac{2}{7}(n^{7/2}-1)$. For both the upper bound $\frac{2}{7}(n^{7/2}-1)+n^{5/2}$ and the lower bound $\frac{2}{7}(n^{7/2}-1)$ both have the dominant term $n^{7/2}$ since $n^{5/2}$ grows slower. Therefore, the Theta evaluation of the sum S is $\Theta(n^{7/2})$.

Programming Task Results.

Table 1: Programming Task 1 Results

10,9,2,5,3,101,7,18	4
186, 359, 274, 927, 890, 520, 571, 310, 916 ...	10
318 , 536 , 390 , 598 , 602 , 408 , 254 , 868 , 379 , 565 , 206 , 619 , 936 , 195...	10

Table 2: Programming Task 2 Results

4, 9,2,5,3,101,7,18,2,1	5
186, 359, 274, 927, 890, 520, 571, 310, 916, 798, 732, 23, 196, 579...	12
318 , 536 , 390 , 598 , 602 , 408 , 254 , 868 , 379 , 565 , 206 , 619 , 936 , 195	10

Java pdfs

```java
/**Tamir Krief, Iaian Milton, Blessing Abumere */



/** You will write a program that computes the length of a longest increasing subsequence
 of a sequence of integers.
 +takes input */

public class Assignment2Task1 {

    public static void main(String[] args) {
        java.util.Scanner input = new java.util.Scanner(System.in);

        int[] user_sequence = Prompt_Sequence(input);

        System.out.print("\nMax Increasing Subsequence: " + MaxIncreasingSubsequence(user_sequence));
    }
    /** a program that computes the length of a longest increasing subsequence
 of a sequence of integers. * */
    public static int MaxIncreasingSubsequence(int[] arr){
        //basecase
        if(arr == null || arr.length == 0)
            return 0;

        //not sure if the grader prefers d like the assignment pdf variable name or an actual
variable name like {counts} ; variable name is just ,like the assignment pdf, d to be safe
        //d[i] corresponds with arr[i]
        int[] d = new int[arr.length]; //keeps track of the COUNTS of the max increasing subsequence
            d[0] = 1; // d[0] will always be initialized to 1 as will any d[i] as that is the
basecase

        int max_count = 1; //intitialization step is  1

        for (int i = 1; i < arr.length;i++){ //i is always bigger than j
            d[i] = 1;   //intitialized to 1 for the base case of no increasing subsequences
            for (int j = 0; j < i; j++){
                //if current element is greater than a previous element in the array and
                //if d[i] is greater than d[j] + 1 then d[i] count will go up
                if (arr[i] > arr[j]){
                    if (d[i] < d[j] + 1)
                        d[i] = d[j] + 1;
                }
            }
            //update max_count if d[i] is bigger
            if (d[i] > max_count)
                max_count = d[i];
        }


        return max_count;
    }

      /**reads the initial sequence which is entered by the user*/
    public static int[] Prompt_Sequence(java.util.Scanner input){
        int[] sequence;
        try {
            System.out.println("\nEnter the length of the sequence");
                sequence = new int[input.nextInt()];

            System.out.println("\nEnter the values of the sequence[]");

            for (int i = 0; i < sequence.length; i++){
                sequence[i] = input.nextInt();
            }
```

```
            System.out.println("sequence[] = " + java.util.Arrays.toString(sequence));
            return sequence;
        }catch(java.util.InputMismatchException e){
            input.nextLine();
            System.err.print("Integer Values only. Please Try again\n");
            Prompt_Sequence(input);
        }

        return null;
    }


}
```

```java
/**Tamir Krief, Iaian Milton, Blessing Abumere */

public class Assignment2Task2 {

    public static void main(String[] args){
        java.util.Scanner input = new java.util.Scanner(System.in);

        int[] user_sequence = Prompt_Sequence(input);

        System.out.print("\nMax Decreasing Subsequence: " + MaxDecreasingSubsequence(user_sequence));

    }

    /**reads the initial sequence which is entered by the user*/
    public static int[] Prompt_Sequence(java.util.Scanner input){
        int[] sequence;
        try {
            System.out.println("\nEnter the length of the sequence");
                sequence = new int[input.nextInt()];

            System.out.println("\nEnter the values of the sequence[]");

            for (int i = 0; i < sequence.length; i++){
                sequence[i] = input.nextInt();
            }
            System.out.println("sequence[] = " + java.util.Arrays.toString(sequence));
            return sequence;
        }catch(java.util.InputMismatchException e){
            input.nextLine();
            System.err.print("Integer Values only. Please Try again\n");
            Prompt_Sequence(input);
        }

        return null;
    }


    /**exists to display results without the unit tests and it helped with the table results */
    public static void DisplayResults(){
        int[] seq1 = {4, 9,2,5,3,101,7,18,2,1};
        int[] seq2 = {186, 359, 274, 927, 890, 520, 571, 310, 916, 798, 732, 23, 196, 579,
            426,188, 524, 991, 91, 150, 117, 565, 993, 615, 48, 811, 594, 303, 191,
            505, 724, 818, 536, 416, 179, 485 , 334 , 74, 998, 100, 197, 768, 421,
            114, 739, 636, 356, 908 , 477, 65};
        int[] seq3 = {318 , 536 , 390 , 598 , 602 , 408 , 254 , 868 , 379 , 565 , 206 , 619 , 936 ,
195 ,
            123 , 314 , 729 , 608 , 148 , 540, 256 , 768 , 404 , 190 , 559 , 1000 , 482 , 141 , 26,
            230 , 550 , 881 , 759 , 122 , 878, 350, 756, 82, 562, 897, 508, 853, 317 ,
            380 , 807 , 23 , 506 , 98 , 757 , 247};

        System.out.print("\nMax Decreasing Subsequence:");

        System.out.printf(
            "\nSequence1: " + MaxDecreasingSubsequence(seq1) +
            "\nSequence2: " + MaxDecreasingSubsequence(seq2) +
            "\nSequence3: " + MaxDecreasingSubsequence(seq3)
        );

    }

    //same as Task1 but reversed?
    /** a program that computes the length of a longest decreasing subsequence of a sequence of
integers. */
    public static int MaxDecreasingSubsequence(int[] arr){
```

```
        //basecase
        if(arr == null || arr.length == 0)
            return 0;

        //d[i] corresponds with arr[i]
        int[] d = new int[arr.length]; //keeps track of the COUNTS of the max decreasing subsequence

            d[0] = 1; // d[0] will always be initialized to 1 as will any d[i] as that is the
basecase

        int max_count = 1; //intitialization step is  1
        for (int i = 1; i < arr.length;i++){ //i is always bigger than j
            d[i] = 1;   //intitialized to 1 for the base case of no decreasing subsequences
            for (int j = 0; j < i; j++){
                //if current element is less than a previous element in the array and
                //if d[i] is less than d[j] + 1 then d[i] count will go up
                if (arr[i] < arr[j]){
                    if (d[i] < d[j] + 1)
                        d[i] = d[j] + 1;
                }
            }
            //update max_count if d[i] is bigger
            if (d[i] > max_count)
                max_count = d[i];
        }


        return max_count;


    }


}
```