**Exam II**

Ramsey Numbers Final Exam

# Problem 1:

State the asymptotic relationship between the functions and , as in , where may be , , . You must justify your answer by showing your work using the

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  | *formula* |
|  | *substitute values* |
|  | *remove smaller order terms* |
|  | *cancel like terms* |
|  | *solve* |

Since , the statement is true.

Since , the statement is true.

Since , the statement is true.

|  |  |
| --- | --- |
|  | *formula* |
|  | *substitute values* |
|  | *cancel like terms* |
|  | *limit rule* |
|  | *limit rule* |
|  | *limit rule* |
|  | *limit rule* |
|  | *solve* |

Since , the statement is true.

Since , the statement is false.

Since , the statement is false.

|  |  |
| --- | --- |
|  | *formula* |
|  | *substitute values* |
|  | *base conversion* |
|  | *log rule* |
|  | *cancel like terms* |
|  | *limit rule* |

Since , the statement is true.

Since , the statement is true.

Since , the statement is true.

|  |  |
| --- | --- |
|  | *formula* |
|  | *substitute values* |
|  | *factor* |
|  | *cancel like terms* |
|  | *exponent rule* |
|  | *exponent rule* |
|  | *subtraction* |
|  | *limit rule* |
|  | *limit rule* |
|  | *limit rule* |

Since , the statement is false.

Since , the statement is true.

Since , the statement is false.

# Problem 2:

Using the master theorem for solving recurrences, state the Big-O value for the following recurrences. If it is inappropriate to use the master method, then state this fact instead.

|  |  |
| --- | --- |
|  | |
| for some constant |  |
|  |  |
| for some constant  for some constant  and all sufficiently large |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

|  |
| --- |
| Case 3 |
| is polynomially greater than |
| : The work completed in the sub-roots, which are not leaves, of the recursion tree is the primary determinant of the computational complexity. As is polynomially greater than , our regularity condition is met. |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

|  |
| --- |
| Case 2 |
| and are polynomially equivalent |
| : The work is evenly distributed between the leaves and sub-roots of the recursion tree. |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

|  |
| --- |
| No Master Theorem Case Applicable |
| We can not compare theandasis *undefined*. As such, the master theorem can not be applied here. |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

|  |
| --- |
| Case 3 |
| is polynomially greater than |
| : The work completed in the sub-roots, which are not leaves, of the recursion tree is the primary determinant of the computational complexity. As is polynomially greater than , our regularity condition is met. |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

|  |
| --- |
| Case 1 |
| is polynomially greater than |
| : The work completed in the sub-roots, which are not leaves, of the recursion tree is the primary determinant of the computational complexity. As is polynomially greater than , our regularity condition is met. |

# Problem 3:

For the following algorithm, (3A) give a precise line count for every numbered line of code in terms of . Use indicator functions if/when needed. When you've completed deriving the line frequencies, (3B) give the efficiency of the algorithm in terms of Big-O.

## **Code**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | void sort(int A[], int n) { | | | | | | | | |
| *1* |  | |  | int i **=** 0**,** j**,** s**;** | | | | | | |
| *2* |  | |  | **while(**i **<** n**-**1**){** | | | | | | |
| *3* |  | |  | |  | s = i**;** | | | | |
| *4* |  | |  | |  | j = i+1**;** | | | | |
| *5* |  | |  | |  | **while(**j **<** n**){** | | | | |
| *6* |  | |  | |  | |  | **if(**A**[**j**]** **<** A**[**s**]){** | | |
| *7* |  | |  | |  | |  | |  | s = j**;** |
|  |  | |  | |  | |  | } | | |
| *8* |  | |  | |  | |  | j **+=** 1**;** | | |
|  |  | |  | |  | } | | | | |
| *9* |  | |  | |  | A**[**i**]** **=** A**[**s**];** | | | | |
| *10* |  | |  | |  | i **+=** 1**;** | | | | |
|  |  | |  | } | | | | | | |
|  |  | } | | | | | | | | |

## **Frequency Counts and Big O Notation**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Line | Code | | | | | Count |  |
|  | void sort(int A[], int n) { | | | | | ----- | ----- |
| *1* |  | int i **=** 0**,** j**,** s**;** | | | | 1 |  |
| *2* |  | **while(**i **<** n**-**1**){** | | | |  |  |
| *3* |  |  | s = i**;** | | |  |  |
| *4* |  |  | j = i+1**;** | | |  |  |
| *5* |  |  | **while(**j **<** n**){** | | |  |  |
| *6* |  |  |  | **if(**A**[**j**]** **<** A**[**s**]){** | |  |  |
| *7* |  |  |  |  | s = j**;** |  |  |
|  |  |  |  | } | | ----- | ----- |
| *8* |  |  |  | j **+=** 1**;** | |  |  |
|  |  |  | } | | | ----- | ----- |
| *9* |  |  | A**[**i**]** **=** A**[**s**];** | | |  |  |
| *10* |  |  | i **+=** 1**;** | | |  |  |
|  |  | } | | | | ----- | ----- |
|  | } | | | | | ----- | ----- |

### Line 1: int i **=** 0**,** j**,** s**;**

#### Count:

* + This line initializes three integer variables i, j, and s and is invoked only once during program execution; therefore, the count is .
  + The time complexity is constant, , because this line has a fixed execution time.

### **Outer While Loop**

#### Line 2: **while(**i **<** n**-**1**)** **{**

##### Count:

###### *Outer While Loop Code Analysis*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **while(**i **<** n**-**1**){** | | | | | |
|  | s = i**;** | | | | |
|  | j = i+1**;** | | | | |
|  | **while(**j **<** n**){** | | | | |
|  | |  | **if(**A**[**j**]** **<** A**[**s**]){** | | |
|  | |  | |  | s = j**;** |
|  | |  | } | | |
|  | |  | j **+=** 1**;** | | |
|  | } | | | | |
|  | A**[**i**]** **=** A**[**s**];** | | | | |
|  | i **+=** 1**;** | | | | |
| } | | | | | |

* + We know  *(line 1),* assume

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Iteration |  | Condition | | Result |
| 1 |  |  | true | loop executes |
| 2 |  |  | true | loop executes |
| 3 |  |  | true | loop executes |
| 4 |  |  | false | loop terminates |

* + When , the while loop condition is executed for 4 iterations, or . This is because the loop condition must perform an additional check for when the condition fails:

However, the inside of the while loop is executed for 3 iterations, , because the loop body is only executed when the condition is true. The exact count value varies depending on the value of .

* + The time complexity is equivalent to the count of , and represents the linear relationship between the input size and the execution time.

### **Middle While Loop**

#### Line 5: **while(**j **<** n**)** **{**

##### Count:

###### *Middle While Loop Code Analysis*

|  |  |  |  |
| --- | --- | --- | --- |
| **while(**j **<** n**){** | | | |
|  | **if(**A**[**j**]** **<** A**[**s**]){** | | |
|  | |  | s = j**;** |
|  | } | | |
|  | j **+=** 1**;** | | |
| } | | | |

* + We know (*line 4*), assume and

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Iteration |  | Condition | | Result |
| 1 |  |  | true | loop executes |
| 2 |  |  | true | loop executes |
| 3 |  |  | false | loop terminates |

* + When and , the middle while loop is executed for 3 iterations, . This includes an additional loop check for when the condition fails. However, the inside of the while loop is executed for 2 iterations () because the loop body is only executed when the condition is true. The exact count value varies depending on the value of .
  + Since this loop resides within the outer while loop, and increments by 1, we can use the formula above to solve for sigma to find the count

###### *Outer While Loop Code Analysis*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **while(**i **<** n**-**1**){** | | | | | |
|  | s = i**;** | | | | |
|  | j = i+1**;** | | | | |
|  | **while(**j **<** n**){** | | | | |
|  | |  | **if(**A**[**j**]** **<** A**[**s**]){** | | |
|  | |  | |  | s = j**;** |
|  | |  | } | | |
|  | |  | j **+=** 1**;** | | |
|  | } | | | | |
|  | A**[**i**]** **=** A**[**s**];** | | | | |
|  | i **+=** 1**;** | | | | |
| } | | | | | |

* + Forto
    - We know that from line 1, and the outer loop is executed as long as , or
  + Table Q3-A: Outer Loop Breakdown

|  |  |
| --- | --- |
|  | *The outer loop is equal to the sum of the middle loops for to .* |
|  | *Since we already found the middle count formula, we can insert it directly into sigma* |
|  | *Breakdown Sigma* |
| *table Q1 continued below* | |

* + Observe first sigma
    - Since is a constant, and in this summation is the only thing incrementing, the value remains the same and repeats for the entire series.

|  |  |
| --- | --- |
|  | Value |
|  |  |
|  |  |
|  |  |

* + - Therefore, to find the count for this summation, we simply need to multiply the value by the size of the series

|  |  |
| --- | --- |
|  | *Formula* |
|  | *Input known value* |
|  | *Determine Series Size  A series has terms* |
|  | *Input Series Size* |
|  | *multiply* |

* + Observe Second sigma
    - For this sigma, the terms exhibit a consistent increment of 1, indicating an arithmetic pattern. As each term is incremented by 1 compared to the previous term, we can intuitively recognize that this is an arithmetic series and is equivalent to the following formula:
    - Convert  *to*

|  |  |
| --- | --- |
|  | *sigma formula* |
|  | *since our series starts at 0 and ends at n-2, we need to replace n-1 for n in formula\** |
|  | *simplify* |

*\* A series has terms, therefore, our series has n-1 terms*

* + Table Q3-B: Outer Loop Breakdown

|  |  |
| --- | --- |
| *table Q1 continued from above* | |
|  | *sigma notation* |
|  | *replace with formulas found above* |
|  | *combine fractions* |
|  | *subtract* |
|  | *factor* |

* + The time complexity of is equivalent to the count of . In Big O notation, the focus is on the dominant term as it has the greatest impact on the overall time complexity. In this case, the dominant term of is , and the is less significant, especially as grows larger. Therefore, the time complexity is simplified to .

#### Line 8: and j **+=** 1**;**

##### Count:

* + This line increments the value of the variable *j* by 1 and executes once for each iteration of the Middle While Loop.
  + Although the middle while condition (*line 5*) executes times due an additional iteration when the loop condition fails, it is important to note that the statements *inside* the loop execute only when the condition passes. Therefore, the count for this line is.
  + The time complexity of is equivalent to the count of . In Big O notation, the focus is on the dominant term as it has the greatest impact on the overall time complexity. In this case, the dominant term of is , and the is less significant, especially as grows larger. Therefore, the time complexity is simplified to .

Next: Inner If statement pg 3 of HW Notes Week 3