CSP2348 Data Structures

Workshop Test 1: 20MAR15 1600-1800

Martin Ponce: Student 10371381

1:

Using a manual method, apply floor() and ceiling() functions to $\log_2(1050)$.

Answer:

1. $1024 \leq 1050 < 2048$
2. $\log_2(1024) \leq \log_2(1050) < \log_2(2048)$
3. $\log_2(1024) = 10 \leq \log_2(1050) < \log_2(2048) = 11$
4. Therefore:
    - $floor(\log_2(1050)) = 10$
    - $ceiling(\log_2(1050)) = 11$

2:

Find the Greatest Common Divisor (GCD) of 1050 and 588 by manually executing the Euclid GCD algorithm shown on page 3 of the textbook or 'lecture01.ppt' slide 7-9.

> To find the GCD of positive integers m and n:
>
> 1. Set p = m, set q = n
> 2. Until q exactly divides p, repeat:
>   1. Set p = q, set q = (p mod q)
> 3. Terminate with answer q

Answer:

1. p = 1050, q = 588
2. 1050 % 588 != 0
    1. p = 588
    2. q = 1050 % 588
        - q = 462
3. 588 % 462 != 0
    1. p = 462
    2. q = 588 % 462
        - q = 126
4. 462 % 126 != 0
    1. p = 126
    2. q = 462 % 126
        - q = 84
5. 126 % 84 != 0
    1. p = 84
    2. q = 126 % 84
        - q = 42
6. 84 % 42 == 0
7. q = 0

3:

Suppose that the following expression is the sum of the characteristic operations of an algorithm.

> $776 * n^2 * \log_2(n) + 3.1 * n^3 + 8 * n^2 + 30 * n^{2/3} + 850$

Answer:

$O(776 * n^2 * \log_2(n) + 3.1 * n^3 + 8 * n^2 + 30 * n^{2/3} + 850)$

$\Rightarrow \max\{O(776 * n^2 * \log_2(n)), O(3.1 * n^3), O(8 * n^2), O(30 * n^{2/3}), O(850)\}$

$\Rightarrow \max\{O(n^2 * \log_2(n)), O(n^3), O(n^2), O(n^{2/3}), O(1)\}$

$= O(n^3)$

Therefore, time complexity for algorithim is $O(n^3)$.

4:

Determine the time complexity of the following method, using O-notation.

```
int example(int[] array) {          // 01 // O(1)
    if(array == null) {             // 02 // O(1)
        return 0;                   // 03 // O(1)
    }                               // 04 // --
    int n = array.length;           // 05 // O(1)
    if(n == 0) {                    // 06 // O(1)
        return 0;                   // 07 // O(1)
    }                               // 08 // --
    int maximum = array[0];         // 09 // O(1)
    int minimum = array[0];         // 10 // O(1)
    for(int i = 1; i < n; i++) {    // 11 // O(1) * #e
        if(array[i] > maximum) {    // 12 // O(1)
            maximum = array[i];     // 13 // O(1)
        }                           // 14 // --
        if(array[i] < minimum) {    // 15 // O(1)
            minimum = array[i];     // 16 // O(1)
        }                           // 17 // --
    }                               // 18 // --
    return n * maximum * minimum;   // 19 // O(1)
}                                   // 20 // --

// Loop-control: i = 1, ..., n-1
// #e = Number of executions = O(n)
// Maximum cost: O(n)
```

Answer:

Time complexity of method is $O(n)$.

Alternative answer:

Let n be the number of the elements of the array, ie. n = array.length.

Lines 02 - 10 and line 19 can be done in constant time $O(1)$.

The `for` loop at line 11 is executed n times.

Inside the body of the loop, from lines 12 - 14 conduct a comparison. Lines 15 - 17 conduct a separate comparison.

Therefore, the total number of comparisons of the `for` loop within lines 11 - 18 is $O(2 * n)$.

Therefore, the time complexity of the algorithm is:

$O(1) + O(2 * n)$
$\Rightarrow \max\{O(1), O(2 * n)\}$
$\Rightarrow \max\{O(1), O(n)\}$
$= O(n)$

5:

Suppose that we have a file **F** that contains **n^2** distinctive integer values, which are in ascending order. Consider the following process:

> - Consecutively, read the values from the file **F** and store them in the same order in an appropriately-sized array **A**
> - Search the array **A** for a specific target value **T**
> - Terminate with either:
>   - **+index** if **T** is found in cell **A[index]**
>   - **-1** if **T** is not found in array **A**

1. Determine which search algorithm is best and state its steps
2. Analyse the time complexity of the entire process in terms of O-notation

Answer:

5.1:

The best search algorithm is binary search because it has the best time complexity for sorted arrays. Array A meets sorted array criteria.

> To find which (if any) component of the sorted (sub)array a[left ... right] equals target:
>
> 1. Set l = left, r = right
> 2. While l ≤ r, repeat:
>   1. Let m be an integer about halfway between l and r
>   2. If target equals a[m], terminate with answer m
>   3. If target is less than a[m], set r = m - 1
>   4. If target is greater than a[m], set l = m + 1
> 3. Terminate with answer none

5.2:

Analysing the time complexity of the entire process:

1. Reading each value from **F** then storing in **A** is a constant time O(1)
   - Since array is of n * n, time to iterate over all values would be $O(n^2)$
2. Using binary search and assuming Steps 2.2 to 2.4 perform a single comparison:
   - If search is unsuccessful, steps are repeated until n^2 is halved to 0
     - Number of comparisons = $floor(\log_2 n^2 + 1)$
       - = $floor(2 * \log_2 n) + 1$
   - If search is successful, steps are repeated at most that many times
     - Max number of comparisons = $floor(\log_2 n^2 + 1)$
       - = $floor(2 * \log_2 n) + 1$
   - Combining both processes
     - $O(n^2) + O(\log_2 n)$
     - $\Rightarrow \max\{O(n^2), O(\log_2 n)\}$
     - $= O(n^2)$