CSP2348 Data Structures

Workshop Test 1: 20MAR15 1600-1800

Martin Ponce: Student 10371381

1:

i:

> Using a manual method, apply floor() and ceiling() functions to $\log_2(996)$.

1. 512 ≤ 996 < 1024
2. $\log_2(512) \leq \log_2(996) < \log_2(1024)$
3. $\log_2(512) = 9 \leq \log_2(996) < \log_2(1024) = 10$
4. Therefore:
   - floor($\log_2(996)$) = 9
   - ceiling($\log_2(996)$) = 10

ii:

> Under what condition/s, floor($\log_2(x)$) = ceiling($\log_2(x)$)? Why?

- floor($\log_2(x)$) = ceiling($\log_2(x)$) if x is a power of two
- Because the answer will be an integer
  - ceiling($\log_2(x)$) = floor($\log_2(x)$) if x is an integer, else ceiling($\log_2(x)$) = floor($\log_2(x)$) + 1

2:

> Find the Greatest Common Divisor (GCD) of 462 and 105 by manually executing the Euclid GCD algorithm shown on slides 7~9 of Lecture1.ppt.
>
> 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

1. p = 462, q = 105
2. 462 % 105 != 0
   1. p = 105
   2. q = 462 % 105
      - q = 42
3. 105 % 42 != 0
   1. p = 42
   2. q = 105 % 42
      - q = 21
4. 42 % 21 == 0
5. Answer = 21

3:

i:

> Consider the binary search algorithm given in slides 24, Lecture03.ppt (also see the algorithm in page No. 409 of the textbook, or the Algorithm in page No.43 of the Java Collections). Hand test this algorithm with the following array of words:
>
> apple, banana, grape, lime, mango, orange, pear, pineapple, plum, strawberry
>
> with each of the following target values: **plum**; **Lychee**.
> (Hint: refer to the examples in slides 26 & 27, Lecture03.ppt, and track changes of l , r and m).

apple	banana	grape	lime	mango	orange	pear	pineapple	plum	strawberry
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

Target "plum":

1. l = 0, r = 9
2. l ≤ r
    1. m = floor((0 + 9) / 2)
        - m = 4
    2. target > a[4]
        - l = m + 1
        - l = 5
3. l ≤ r
    1. m = floor((5 + 9) / 2)
        - m = 7
    2. target > a[7]
        - l = m + 1
        - l = 8
4. l ≤ r
    1. m = floor((8 + 9) / 2)
        - m = 8
    2. target == a[8]
5. Terminate with answer m
    - m = 8

6 comparisons were made until answer was found, assuming Steps 2.2 - 2.4 are one comparison (if-then-else).

Target "Lychee":

1. l = 0, r = 9
2. l ≤ r
    1. m = floor((0 + 9) / 2)
        - m = 4
    2. target < a[4]
        - r = m - 1
        - r = 3
3. l ≤ r
    1. m = floor((0 + 3) / 2)
        - m = 1
    2. target > a[1]
        - l = m + 1
        - l = 2
4. l ≤ r
    1. m = floor((2 + 3) / 2)
        - m = 2
    2. target > a[2]
        - l = m + 1
        - l = 3
5. l ≤ r

    1. m = (floor((3 + 3) / 2)
        - m = 3
    2. target > a[3]
        - l = m + 1
        - l = 4

6. l !≤ r
7. Terminate with answer none

9 comparisons were made until algorithm terminated, assuming Steps 2.2 - 2.4 are one comparison (if-then-else).

ii:

> How many comparisons are required in each case?

- Target "plum"
    - 6 comparisons were made until answer was found, assuming Steps 2.2 - 2.4 are one comparison (if-then-else).
- Target "Lychee"
    - 9 comparisons were made until algorithm terminated, assuming Steps 2.2 - 2.4 are one comparison (if-then-else).
- Binary search is O(log n)

4:

> Assume that the following expression is the function of a given algorithm (n is input size).
>
> $(7n + 8)^4 - 72 * (n + 1)^9 / (n - 3)^4 + n * (\log_2(n))^{12} + 1029$
>
> Determine the time complexity of the algorithm in O-notation.

$$
\begin{aligned}
&(7n+8)^4 - 72 \times (n+1)^9/(n-3)^4 + n \times (log_2(n))^{12} + 1029 \\
&= \frac{(7^4n^4+8^4) - 72n^9 + 1^9}{(n^4-3^4) + n \times (log_2(n^{12})) + 1029} \\
&= \frac{n^4+n^9}{n^4 + n \times log_2(n^{12})} \\
&= n^4+n^9-n^4+n \times log_2(n^{12}) \\
&= n^9 - n \times log_2(n^{12}) \\
&= n \times log_2(n^{-3})
\end{aligned}
$$

5:

i:

> An array A[0...n-1] is in descending order if A[i-1] ≥ A[i] holds for all i (0 < i < n).
>
> Assume an array **A**[0...n-1], contains distinct integers. Write an algorithm, **descending(A,n)**, that determines whether or not the components of **A** are in descending order (n ≥ 0). If they are in descending order, the algorithm returns n, otherwise it returns an integer m, which is the minumum index value of **A**, such that m ≤ n and **A**[m-1] < **A**[m].
>
> For instance, let **A**[] = {62, 50, 40, 22, 21, 20, 11} and **B**[] = {62, 50, 40, 20, 21, 22, 11}. Then **descending(A, 7)** would return 7 and **descending(B, 7)** would return 4.

```
int descending(int[] array) {
    if(array == null) {
        return -1;
    }
    int n = array.length;
    for(i = 1; i < n; i++) {
        if(array[i + 1] > array[i]) {
            return i;
        }
    }
    return n;
}
```

ii:

> Determine the time complexity of your algorithm, using O-notation.
> (Hint: refer to examples in slides 33, Lecture03.ppt)

- Loop control = 1,2,3, ..., n-1
- Number of executions = O(n)
- Maximum cost = O(n)