CSP2348 Data Structures

Solutions to Workshop Test 1: Algorithm Analysis (Total Marks: 10)

(for your reference only)

Tasks: Attempt all five questions below. Note: all workings must be shown!

Q1. (2 marks)
(i) Using a manual method, apply *floor*( ) and *ceiling*( ) functions to $log_2(996)$.
(ii) Under what condition, $floor(\log_2(x)) = ceiling(\log_2(x))$? Why?

Solutions (for reference only):

Option 1 – (use of inequality):

(i) Since $512 < 996 < 1024$

We have $\log_2(512) < \log_2(996) < \log_2(1024)$,

then $9 = \log_2(2^9) = \log_2(512) < \log_2(996) < \log_2(1024) = \log_2(2^{10}) = 10$

or $9 < \log_2(996) < 10$.

Therefore $floor(\log_2(996)) = 9$ and $ceiling(\log_2(996)) = 10$. [1 mark]

(ii) $floor(\log_2(x)) = ceiling(\log_2(x))$ when x is a power of 2 (or, when $\log_2(x)$ is a whole number). [1 mark]

Option 2: halve the value to reach 1 – the number of times of halving is floor( ).

(i) Since 996/2=498
498/2=249,
249/2 = 124
124/2 = 62
62/2 =31
31/2 = 15,
15/2 = 7,
7/2 = 3,
3/2 = 1,

$floor(\log_2(996)) = 9$ as it takes 9 divisions by 2 to reach 1,

and $ceiling(\log_2(996)) = floor(\log_2(996)) + 1 = 10$

[1 mark]

Marking criteria:	marks
show full working	0.5~1.0
Final result only	0.5 ~ 1
Use of calculator – didn't do it manually	0.5

(ii) *(see Option 1)* [1 mark]

Q2. (1 mark)
Find the Greatest Common Divisor (GCD) of 462 and 105 by manually executing the Euclid GCD algorithm shown on slide 7~9 of lecture01.ppt.

Solution: by running the algorithm (for reference only):

p	***q***	***r*** =(*p* modulo *q*)		**Notes**
462	105			**Step 1**
		42	does *q* exactly divides *p*? - **no**	**Step 2:** r = (*p* modulo *q*)= 462 %105 =42, since 462 ÷ 105 =4 with remainder 42
105	42			**Step 2.1**
		0	does *q* exactly divides *p*? - **no**	**Step 2:** r = (*p* modulo *q*)= 105 %42 =21, since 105÷42 = 2 with remainder 21;
42	21			**Step 2.1**
		0	does *q* exactly divides *p*? - **yes**	**Step 2:** r = (*p* modulo *q*)= 42 %21 =0, since 42÷21 = 2 with remainder 0;
			Answer: q=21	**Step 3** Answer q = 21 (or, GCD of (462, 105) is 21)

Marking criteria:	marks
show full working, i.e., executing the algorithm	0.5~1.0
Final result only	0.4
follow the algorithm but error occur in calculation	0.5 ~.08
Use of math but differs from the algorithm	0.1~ 0.3

Q3. **(3 marks)**

(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*).

(ii) How many comparisons are required in each case?

(i)

(a) Target: **plum** (*left* = 0; *right* = 9)

		l=0	1	2	3	4	5	6	7	8	*r*=9
Initially: (Step 1)	a:	apple	banana	grape	lime	mango	orange	pear	pineapple	plum	strawberry
	l=0; r=9										
		0	1	2	3	4	*l* =5	6	7	8	*r*=9
1st iteration	a:	apple	banana	grape	lime	mango	orange	pear	pineapple	plum	strawberry
	m = 4 (*since (l+r)/2 = (0+9)/2=4), Target '**plum**' was compared with a[4] = '**mango**'* ('**plum**' > *a[4]* = '**mango**', so set *l* to *m*+1=4+1=5)										
		0	1	2	3	4	5	6	7	*l*=8	*r*=9
2nd iteration	a:	apple	banana	grape	lime	mango	orange	pear	pineapple	plum	strawberry
	m= 7 (*since (l+r)/2 = (5+9)/2=7), Target '**plum**' was compared with a[7] = '**pineapple**'* ('**plum**' > *a[7]* = '**pineapple**', so set *l* to *m*+1=7+1=8)										
		0	1	2	3	4	5	6	7	*l*=8	*r*=9
3rd iteration	a:	apple	banana	grape	lime	mango	orange	pear	pineapple	plum	strawberry
	m=8 (*since (l+r)/2 = (8+9)/2=8), Target '**plum**' was compared with a[8] = '**plum**'*, ('**plum**' = *a[8]* = '**plum**', *Target equals to a*[8], terminate with 8 (in Step 2.2)										

Note: array components that were compared with the target are marked red & underlined.

Search is a successful: Target (plum) found in a[8]. [1.2 marks]

(b) Target: **lychee** (*left* = 0; *right* = 9)

		l=0	1	2	3	4	5	6	7	8	*r*=9
Initially: (Step 1)	a:	apple	banana	grape	lime	mango	orange	pear	pineapple	plum	strawberry
	l=0; r=9										
		l=0	1	2	*r*=3	4	5	6	7	8	9
1st iteration	a:	apple	banana	grape	lime	mango	orange	pear	pineapple	plum	strawberry
	m= 4 (*since (l+r)/2 = (0+9)/2=4), Target '**lychee**' was compared with a[4] = '**mango**'* (' **lychee**' < *a[4]* = '**mango**', so set *r* to *m*-1=4-1=3)										
		0	1	*l*=2	*r*=3	4	5	6	7	8	9
2nd iteration	a:	apple	banana	grape	lime	mango	orange	pear	pineapple	plum	strawberry
	m=1 (*since (l+r)/2 = (0+3)/2=1), Target '**lychee**' was compared with a[1] = '**banana**'* (' **lychee**' > *a[1]* = '**banana**', so set *l* to *m*+1=1+1=2)										
		0	1	2	*l*=*r*=3	4	5	6	7	8	9
3rd iteration	a:	apple	banana	grape	lime	mango	orange	pear	pineapple	plum	strawberry
	m=2 (*since (l+r)/2 = (2+3)/2=2), Target '**lychee**' was compared with a[2] = '**grape**'* (' **lychee**' > *a[2]* = '**grape**', so set *l* to *m*+1=2+1=3)										
		0	1	2	*r*=3	*l*=4	5	6	7	8	9
4th iteration	a:	apple	banana	grape	lime	mango	orange	pear	pineapple	plum	strawberry
	m=3 (*since (l+r)/2 = (3+3)/2=3), Target '**lychee**' was compared with a[2] = '**lime**'* (' **lychee**' > *a[3]* = '**lime**', so set *l* to *m*+1=3+1=4 , i.e., *l* = 4) now that *l* > *r*, loop ended; *Step 3:* Terminate with *None*										

Search is unsuccessful: answer *None* (or Target, lychee, was not found). [1.2 marks]

(ii) From the above tracking, we conclude that

In case (a): the target is compared with a[4], a[7], a[8] – a total of 3 comparisons; [0.3 mark]

In case (b): the target is compared with a[4], a[1], a[2], and a[3] – 4 comparisons. [0.3 mark]

Q4: (2 marks)

Assume that the following expression is the sum of the time characteristic operations of a given algorithm.

$$(7n+8)^4 - 72 \times (n+1)^9 / (n-3)^4 + n \times (\log_2(n))^{12} + 1029$$

By applying the *O*-natation operations, determine the time complexity of the algorithm.

Solution (reference only):

Since $O((7n+8)^3) = O((7n+8)*(7n+8)*(7n+8)*(7n+8))$
$= O(7n+8)*O(7n+8)*O(7n+8)*O(7n+8)$ (using rule: O(g(n)*h(n))= O(g(n)*Oh(n))
$= O(n)*O(n)*O(n)*O(n))$
$=O(n*n*n*n)$ (using rule: O(g(n)*h(n))= O(g(n)*Oh(n))
$=O(n^4)$;

and

$O(-72*(n+1)^9) /(n-3)^4) = O((n+1)^9) /(n-3)^4)$ (by neglecting constant factor -10219)
$= O((n+1)^9) /O((n-3)^{4)}$ (using rule: O(g(n)/h(n))= O(g(n)/Oh(n))
$= \ldots$ (similar to the case of $O((3n+2)^3$ above))
$=O(n^9)/O(n^4))$
$=O(n^9/n^4)$ (using rule: O(g(n)/h(n))= O(g(n)/Oh(n))
$=O(n^5)$;

Therefore

$$O((7n+8)^4 - 72 \times (n+1)^9 / (n-3)^4 + n \times (\log_2(n))^{12} + 1029)$$

$\Rightarrow \max \{O((7n+8)^4)\}, O(-72\times(n+1)^9)/(n-3)^4), O(n\times(\log_2 n)^{12}), O(1029)\}$,
$\Rightarrow \max \{O(n^4), O(n^5), O(n\times(\log_2 n)^{12}), O(1)\}$,
$= O(n^5)$

The time complexity of the algorithm is $O(n^5)$.

(Note: for any constant integer k>0, $O(\log_2(n)^k) < O(n)$. Therefore, $O(n*\log_2(n)^{12} < O(n^2))$.

Marking criteria:	marks
Shows full working	1.0~2.0
Final result (correct) only	1
No use of O-notation	0.5 or less
Other issues: OK if the rules to be used were not shown. many accepted any variations	

Q5 (2 marks)

(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], contain distinct integers. Write an algorithm, ***descending*(A, *n*)**, that determines whether or not the components of **A** are in descending order ($n \geq 0$). If they are in descending order, the algorithm returns n. Otherwise it returns an integer m, which is the minimum index value of **A**, such that $m \leq n$ 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**.**

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

Solutions **(Note: Reference only - there are many version of algorithms)**

Method 1: (using for loop):

(i)
```
descending (A, n)
// determine if integer array, A[0, ..., n-1], is in descending order or not
Step 1:   if (n<=0) return 0
Step 2:   for i = 1,..., n-1, repeat
 Step 2.1   if A[i-1] < A[i] return i;
Step 2:   return n;
```

(ii) Analysis: (by counting the number of comparisons)
There are n array components. Inside the body of the *for* loop, Step 2.1 makes one comparison. This step will be repeated for a maximum of n times (i.e., *for* loop in Step 2). So Step 2 needs O(n) time. Step 1 and Step 3 each needs O(1) time. Therefore, the complexity of the algorithm will be O(1) + O(n)+O(1) = O(n).

==

Method 2: (counting No. statements to be executed, refer to lecture slides 33 in Lecture3):

code	##	cost
int descending(int[] array, int n) {	01	-
if (n<=0) return 0;	02	O(1)
for (i=1; i<n; i++)	03	O(1) *#e
if (A[i-1]<A[i]) return i;	04	O(1) *#e
return n;	05	O(1)
}	06	-
loop-control = 1,2,3,…,n-1 #e = number of executions = O(n) maximum cost = O(n)		

Marking criteria:	marks
Algorithm Design: Shows full working (step by step)	1.0
Algorithm analysis: Final result (correct) only	0.1 only
Should have shown that the loop body would run n times or O(n) All other lines: should be O(1)	0.5 ~1.0
Overall: reach to result of O(n)	0.2+

END of THE SOLUTIONS