CS 58000\_01 Algorithm Design, Analysis & Implementation(3 cr.)

Assignment As\_01

Student Name: Truc Huynh

I: [80 points]: each of I.1 through I.8 is 10 points.

**(I.1) Give the input and output specifications for :**

1. the algorithm a?
2. the algorithm b?

Answer:

|  |  |  |
| --- | --- | --- |
|  | algorithm a | algorithm b |
| Input | Two n-bit integer x and y, where y ≥ 1 | Two n-bit integer x and y, where y ≥ 1 |
| Output | The quotient and reminder of x divided by y | The quotient and reminder of x divided by y |

**(I.2) What is the input size for:**

1. the algorithm a?
2. the algorithm b?

Answer:

|  |  |
| --- | --- |
|  | algorithm a & algorithm b |
| Input Size | Algorithm a & algorithm b will have the same input size as they take x and y for division  The input x or y will be an integer n. The input size for each of them will be└ log2 n ┘ + 1 bits  Define the input size as the number of symbols (in this case bits) used for encoding a positive integer n. |

**(I.3) What is the basic operation for:**

1. the algorithm a?
2. the algorithm b?

Answer:

|  |  |  |
| --- | --- | --- |
|  | algorithm a | algorithm b |
| Basic Operation | Right shift (bit additions) | Left shift (bit multiplication)  Right shift (bit additions) |

**(I.4) In algorithm b, what is the functionality of the following segment of statements?** (i.e., give the reasons for writing each of the statements. Why double the q and r? Why increase r by one when x is odd? Why reduce r by y and increase q by one when r ≥ y?)

q := 2 \* q; r := 2 \* r;

if (x is odd) then r := r + 1;

if (r ≥ y) then

{ r := r – y; q := q + 1};

Answer:

|  |  |
| --- | --- |
| Why double the q and r? | In the recursive call, we divide x by 2. There we must double q and r to make it even. It can be done by shift left one bit |
| Why increase r by one when x is odd? | Because we want to get the top division (not the floor division) when using integer division. We want to change it to an even number that can divide by 2. Ex: 7/2 = 3 (floor division) but that is not what we want to get. Therefore, we add (7+1)/2 = 4 |
| Why reduce r by y and increase q by one when r ≥ y?) | * Reduce r by y to get the remaining value of x * Adding 1 to q because we just subtract y from (reduce) the value of r. Thus, we prove that previous r (before the subtraction) can divide one more y -> Increase q by 1 |

**(I.5) Analyze and derive the algorithm’s time and space efficiency for Algorithm a**.

(Hint: express time efficiency in terms of summation )

Given algorithm a,

if x = 0, then return (q, r) := (0, 0);

q := 0; r := x;

while (r ≥ y) do // takes n iterations for the worse case.

{ q := q + 1;

r := r – y}; // O(n) for each r – y, where y is n bits long.

return (q, r);

Answer:

|  |  |
| --- | --- |
|  | algorithm a |
| Time Efficiency | Takes 2n iterations for the worse case, which says x/1 where x is n bits long with a max value 2n – 1. Since this takes linear time O(n) for each iteration. The worse case would be O(n2n) |
| Space Efficiency | Take 4 spaces: x, y, q, r |

**(I.6) Analyze and derive the algorithm’s time and space efficiency for algorithm b.**

(Hint: for time efficiency, use recurrence relation and solve for the recurrence relation system)

Given algorithm b,

if (x = 0) then return (q, r):=(0, 0);

(q, r) := divide(└x/2┘, y ) //requires n-bits right shift

q := 2 \* q, r := 2 \* r; // shift left one bit.

if (x is odd) then r := r + 1; // needs c\*n-bits

if (r ≥ y) then // additions

{ r := r – y; q := q + 1};

return (q, r);

Answer:

|  |  |
| --- | --- |
|  | algorithm a |
| Time Efficiency | * Takes n call, so the worse case would be O(n2) |
| Space Efficiency | * Take 4 spaces: x, y, q, r |

**(I.7) Can these two algorithms a and b be improved? Justify your answer.**

1. for the algorithm a?
2. for the algorithm b?

Answer:

|  |  |  |
| --- | --- | --- |
|  | algorithm a | algorithm b |
| Improvement | * No improvement if the use of iteration is applied. * Can only improve by using recursive (algorithm b) because time complexity is huge compared to algorithm b (O(n2n) > O(n2)) * x must be ≥ 0 for the input (precondition). | * Pretty optimal already (in terms of time and space complexity). * x must be ≥ 0 for the input (precondition). |

**(I.8) Compare these two algorithms a and b:**

Which is a better algorithm in terms of time and space efficiency? Justify your answer.

Answer:

|  |  |  |
| --- | --- | --- |
|  | algorithm a | algorithm b |
| Time efficiency | * O(n2n) is slower algorithm b | * O(n2) is faster than algorithm a |
| Space efficiency | * Take 4 spaces: x, y, q, r. The calculation between them is stored in the computer register (the number of registers depends on the computer). Thus, the same efficiency as algorithm b. | * Take 4 spaces: x, y, q, r. Thus, the same efficiency as algorithm a |

Reference:

Purdue slides Ch000\_02\_IntroFoundation\_ProgCorrectionLec.pdf from CS 58000\_01 Algorithm Design, Analysis & Implementation