CS 58000\_01/02I Algorithm Design Analysis & Implementation(3 cr.)

Assignment As\_01

Student Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Due no later than Monday, September 11, 2023, midnight before Tuesday, submit to purdue.brightspace.com. No late assignment is acceptable.

Only a Word file (.doc or .docx file) is accepted. Will not accept any PDF file or other file (or 10 points off if you use a PDF file). This allows me to give comments on your work. It would be easier to grade your assignments.

Your file name should be such as Name\_CS580\_As01. (Name could be specified as such JNg for John Ng). Deduct 5 points without specifying correctly your filename, such as JNg\_CS486\_As01.

Total number of points for this Assignment As\_01 is 150 points.

*You need to provide the details of your work which shows how you obtained the solution for each of the problems.*

**Problem I [80 points] Algorithms Comparison**

Given two algorithms “function divide(x, y)” in my lecture note Ch 00\_02\_IntroFoundation\_ProgCorrectionLec.pptx, (posted in the Purdue.brightspace.com), the first one is iterative and the second one is recursive. The operators used are: “shift right one bit”, “shift left one bit”, copy, add (+), compare two contents of given variables (such as ), assign (:=), and call statement (such as a recursive call statement).

**Algorithm Ia:**

**function divide(x, y)**

Input: Two n-bit integers x and y, where y ≥ 1.

Output: The quotient and remainder of x divided by y.

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

q := 0; r := x;

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

{ q := q + 1;

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

return (q, r);

**Algorithm Ib:**

**function divide(x, y)**

Input: Two n-bit integers x and y, where y ≥ 1.

Output: The quotient and remainder of x divided by y.

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);

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

(I.a) for the algorithm Ia?

(I.b) for the algorithm Ib?

(I.cd) What is the input size for:

(I.c) for the algorithm Ia?

(I.d) for the algorithm Ib?

(I.ef) What is the basic operation for :

(I.e) for the algorithm Ia?

(I.f) for the algorithm Ib?

(I.g) In algorithm Ib, state the functionality for the following statements?

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};

(I.h) Analyze and derive the algorithm’s time and space efficiency for Algorithm Ia.

(Hint: express time efficiency in terms of summation )

Given algorithm Ia,

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

q := 0; r := x;

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

{ q := q + 1;

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

return (q, r);

(I.i) Analyze and derive the algorithm’s time and space efficiency for algorithm Ib.

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

Given algorithm Ib,

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);

(I.j) Can these two algorithms Ia and Ib be improved? Justify your answer.

1. for the algorithm Ia?
2. for the algorithm Ib?

(I.k) Compare these two algorithms Ia and Ib:

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

**Problem II [30 points]: Polynomial function**

Given the following modular exponentiation called function modexp(x, y, N)

function modexp(x, y, N)

//Compute xy mod N

Input: Two n-bits integers x and N, an integer exponent y.

Output: xy mod N.

if (y == 0) then return 1;

z = modexp(x, └y/2┘, N); // z = x└ y/2 ┘ mod N

if (y is even) then return z2 mod N;

else return x \* z2 mod N;

(II.1) Apply function modexp(x, y, N) to compute mod 13. Show step by step. Complete the following table to show the execution of the algorithm modexp(15, 7, 13).

|  |  |  |
| --- | --- | --- |
| x, y, N | z = modexp(x, └y/2┘, N); | if (y is even) then return z2 mod N;  else return x \* z2 mod N; |
| 15, 7,13 | z = modexp(15, 7, 13). |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

(II.2) How many times for execute the recursive calls modexp(x, └y/2┘, N)?

(II.3) Use the slide of the example to compute mod 13.

(Hint: see Example: Compute when k is a power of 2.

Find mod 713.)

**Problem III(40 points) : Cutting a rope:**

A rope n inches long needs to be cut into n pieces 1 inch long.

(III.a) Outline/design an algorithm unitCut (do not give me any program code) that performs this task with the minimum number of cuts if several pieces of the rope can be cut at the same time.

(III.b) Also, give a formula for the minimum number of cuts.

(III.c) If n is 50000 inches, for getting 50000 1-inch pieces, where is the first cut? And how many cuts are to be performed.

(III.d) What is the input size and time efficiency of your algorithm in (I.a)?

**Note: If you provide your answer in your handwriting, good handwriting is required.**

**Proper numbering your answer to each problem is strictly required. The problem’s**

**solution must be orderly given. (10 points off if not)**