Name: Alvaro Austin
NPM: 2106752120
Class: 6
Asdos: VAC

1) Algo A: $8n \lg n$ We can find n needed by simplifying A is efficient than B so

B: $2n^2$

$$\frac{B}{A} = \frac{2n^2}{4n \lg n} \rightarrow \frac{n}{2 \lg n} : 1$$

$n = 4 \lg n$

$$\frac{n}{\lg n} : 4 \rightarrow \frac{n}{2 \log n} : 4$$

$$\frac{n}{4} : 2 \log n$$

$a^c = b$

$2^{\frac{n}{4}} = (n)^4$

$2^n = n^4$

$2 : n^{\frac{4}{n}}$

We can get $n : 16$, to get

$2 = 16^{\frac{4}{16}}$

$2 : 16^{\frac{1}{4}}$

$2 : 2^{4 \cdot \frac{1}{4}}$

$2 = 2$

So n_0 is 16 because $16 \geq 16$ Hence, n_0 is 16

↓ we can define

2) Slower → Faster

- 2^{10}
- $2^{\log n}$
- $3n + 100 \log n$
- $4n$
- $n \log n$
- $4n \log n + 2n$
- $n^2 + 10n$
- n^3
- 2^n

3) Asymptotic notation $\Theta(F(n))$:

- $4n \log n + 2n : \Theta(n \log n)$ $C_0 = 4$
- $3n + 100 \log n : \Omega(n)$ $C_0 = 3$
- $n^2 + 10n : \Omega(n^2)$ $C_0 = 1$
- $2^{10} : \Theta(1)$ $C_0 = 2^{10}$
- $4n : \Omega(n)$ $C_0 = 4$
- $n^3 : \Omega(n^3)$ $C_0 = 1$
- $2^{\log n} : \Omega(2^{\log n})$ $C_0 = 1$
- $2^n : \Theta(2^n)$ $C_0 = 1$
- $n \log n : \Theta(n \log n)$ $C_0 = 1$

④ a) Is $2^{n+1} = O(2^n)$

Yes. $2^{n+1} = 2 \cdot 2^n$. Based on big O definition. $F(N) = O(g(N))$ if there exists positive constants C, N_0 such that $F(N) \leq C \cdot g(N)$ for all $N \geq N_0$

We could see here that $F(N) = 2^{n+1} = 2 \cdot 2^n$. So based on the definition $F(N) = O(2^n)$ is correct because 2 is a positive constant C. Hence Proved that $2^N \leq 2 \cdot 2^N$

$2^N \leq 2^{n+1}$

Hence, Its true that $2^{n+1} = O(2^n)$

b) Is $2^{2n} = O(2^n)$

Based on Big O notation definition, $F(n)$ is said to be $O(g(n))$ if there exist positive constants c, n_0 such that $F(n) \leq c \cdot g(n)$ for all $n \geq n_0$

The question states that $2^{2n} = O(2^n)$. We shall prove
$F(n)$ $g(n)$

$$2^{2n} \leq c \cdot 2^n$$

$$\ln 2 \cdot 2n \leq \ln c + \ln 2 \cdot n$$

$$2n \leq \ln c + n$$

$n \leq \ln c$, which is wrong because no such constant fit every case on $n \geq n_0$ all

Hence $2^{2n} \neq O(2^n)$ //

(5) Loop invariant for euclid algorithm → Guard: $y \neq 0$

Initialization : We have to check / show that loop invariants works at first iteration

yang ingin dipenuhi invariant:
$x = y \cdot k + x \% y$ formula
k = konstant

gcd(12,15)
a=12 b=15

First loop
x = 12
y = 15
While(y ≠ 0) → guard

variabel penampung — temp = 15
y = 12 mod 15 = 12
x = 15

Ingin mencari yg terbesar pada iterasi pertama untuk x lalu y untuk yg terkecil (pelemen)

Sehingga invariant loop benar pada saat iterasi pertama ini, karena $y \neq 0$ dan belum ada nilai gcd, Hal ini juga memenuhi $x = y \cdot k + x \% y$.

Namun apabila kondisi telah terpenuhi (x sudah terbesar dan y sudah terkecil) maka loop ini benar juga, karena kita hanya menentukan nilai x dan y. Memenuhi $x = y \cdot u + x \% y$

Maintenance :

- Pada setiap loop akan dilakukan perhitungan nilai x dan y. Dalam setiap perhitungan
- Selalu memenuhi syarat bahwa $x = y \cdot u + x \% y$ akan selalu terpenuhi. Hal ini karena kita
- Selalu menghitung berdasarkan sifat $GCD(a,b) = GCD(b, a \bmod b)$. Maka pada tahap ini berlaku invariant loop

Termination : Untuk yg terakhir dimana x sudah menyatakan nilai gcd dan negasi dari guard terpenuhi yaitu $y = 0$ maka kondisi untuk loop invariant juga terpenuhi berdasarkan syarat diatas yaitu $x = y \cdot k +$ dimana k, dan z adalah konstanta

dimana artinya $x = gcd(a,b)$ karena $x \% y = 0 = y$ dari sifat $gcd(a,0) = a$. Sebelumnya terjadi $gcd(x, x \% y)$, $x \% y = 0$

Sehingga dapat dikatakan bahwa code euclid dan tsb benar berdasarkan invariant loop //