CSC301 HW1

Alex Zhang

Januray 2023

1 Question 1

1.1 (a)

Based on the formula, $g(n) = \sum_{k=0}^{n} r^k = \frac{1-r^{n+1}}{1-r}$, for $r < 1$. Then

$$g(n) = \frac{1 - r^{n+1}}{1 - r} \leq \frac{1}{1 - r}$$

since $r^{n+1} \geq 0$ when $r \geq 0$. We can then assume that $\exists c$ which $c = \frac{1}{1-r}$ becuase r is a constant and $\exists N$ where $N = 1$. So,

$$g(n) \leq c \cdot 1$$

for all $n \geq N$, then

$$g(n) = O(1)$$

1.2 (b)

When $r = 1$, $g(n)$ will be $1 + 1 + 1^2 + \cdots + 1^n$

$$g(n) = 1 \cdot n + 1 = n + 1$$

So, there exists $c = 10$, $N = 1$ such that

$$g(n) \leq 10 \cdot n$$

for all $n \geq N$, then

$$g(n) = O(10n)$$

The time complexity for $g(n)$ when $r = 1$ will be $O(n)$.

1.3 (c)

When $r > 1$, the formula for finite geometric series sum will be $\frac{r^{n+1}-1}{r-1}$. Then,

$$g(n) = \frac{r^{n+1}-1}{r-1} \leq \frac{r^{n+1}}{r-1}$$

We can assume that there exists a $c = \frac{1}{r-1}$, becuase r is a constant so $1/r-1$ will also be a constant and exists $N = 1$. This satisfies that

$$g(n) \leq c \cdot r^{n+1}$$

for all $n \geq N$, then

$$g(n) = O(r^{n+1})$$

The time complexity for $g(n)$ when $r > 1$ will be $O(r^n)$.

2 Question 2

2.1 (a)

```
public static void main(String[] args) {
    int[] test = new int[2];
    test = divide(0b1011, 0b10);
    System.out.println("the quotient will be: " + test[0]);
    System.out.println("the remainder will be: " + test[1]);
}

static int[] divide(int x, int y){
    int[] ans = new int[2];
    if(x == 0)
        return ans;
    ans = divide((int)Math.floor(x / 2), y);
    ans[0] *= 2;
    ans[1] *= 2;

    if(x % 2 == 1)
        ans[1] += 1;
    if(ans[1] >= y){
        ans[1] -= y;
        ans[0] += 1;
    }
    return ans;
}
```

My input for x is 1011 and y is 10 in binary.

For line

```
if(x == 0)
    return ans;
```

This happens in the last recusion call when $x = 0, y = 10$, and it will return the array.
For lines

```
ans = divide((int)Math.floor(x / 2), y);
ans[0] *= 2;
ans[1] *= 2;
```

The first line is doing the recusion and the rest will execute in every recursion.
This condition

```
if(x % 2 == 1)
    ans[1] += 1;
```

will be true when $x = 1, 101, 1011$ since they are all odd numbers.
The last condition

```
if(ans[1] >= y){
    ans[1] -= y;
    ans[0] += 1;
}
```

will execute when $x = 10$ and 1011 which the remainder will be 2 and 3 respectively.

2.2 (b)

The base case will be $x = 0$ and y is an abitary number. Base case is true since the algorithm will return $q = 0, r = 0$, which satisfies

$$0 = 0 \cdot y + 0 \text{ with } 0 \leq r < y$$

For induction case, assume that $x = n$ and given an abitary y, which satisfies

$$n = q \cdot y + r \text{ with } 0 \leq r < y$$

Then

$$\lfloor n/2 \rfloor = q' \cdot y + r'$$

Case 1: n is even Then $\lfloor n/2 \rfloor = n/2$.

$$\begin{aligned} n/2 &= q' \cdot y + r' \\ n &= 2q' \cdot y + 2r' \end{aligned}$$

If $2r' < y$, then $2q' = q$, $2r' = r$.

$$n = q \cdot y + r \text{ with } 0 \leq r < y$$

which is true.
If $2r' \geq y$, for the algorithm,

$$n = (2q' + 1)y + (2r' - y)$$

So $2q' + 1 = q$ and $2r' - y = r$ which is also true since

$$0 \leq r' < y$$

$$0 \leq 2r' < 2y$$

$$0 \leq 2r' - y < y$$

We can get

$$n = q \cdot y + r \text{ with } 0 \leq r < y$$

Case 2: n is odd Then $\lfloor n/2 \rfloor = (n-1)/2$.

$$\begin{aligned} (n-1)/2 &= q' \cdot y + r' \\ n &= 2q' \cdot y + 2r' + 1 \end{aligned}$$

For $2r' + 1 < y$ and $y \leq 2r' + 1 < 2y$, without loss of generality, the proving is the same in case 1, which is true.
For $2r' + 1 = 2y$, it is impossible since $2r' + 1$ will always be an odd number but $2y$ will always be an even number.
Overall, the indcution shows that this algorithm is correct.■

2.3 (c)

Assume that input x, y are n-bits numbers, then this algorithm takes n times of recursion calls becuase each time the number of bit decrease by one. For each recursion call, for the worst case, every call will go into

```
if(ans[1] >= y){
    ans[1] -= y;
    ans[0] += 1;
}
```

condition, which addition is performed. This required linear time complexity.
Overall, there are n times of recursion calls for n-bits number, and for each call, there is going to be an addition performing linear complexity. So the total time complexity for this algorithm is $O(n^2)$.

3 Question 3

By definition, $x \equiv y \bmod N$ if and only if N divides $x - y$. In question since $x \equiv x' \bmod N$, and $y \equiv y' \bmod N$, equivalently,

$$x - x' = N \cdot k$$

$$y - y' = N \cdot l$$

for k, $l \in \mathbb{Z}$
Then $x = N \cdot k + x'$, $y = N \cdot l + y'$, and

$$\begin{aligned} xy &= (N \cdot k + x')(N \cdot l + y') \\ xy &= N^2kl + Nky' + Nlx' + x'y' \\ xy &= N(Nkl + ky' + lx') + x'y' \\ xy - x'y' &= N(Nkl + ky' + lx') \end{aligned}$$

By definition, let $m \in \mathbb{Z}$ and $m = (Nkl + ky' + lx')$ since integer is closed under addition.

$$xy - x'y' = N \cdot m$$

which by definition again,

$$xy \equiv x'y' \bmod N$$

The substitution rule for modular multiplication is true.$\blacksquare$