1 Inductive Proofs

Prove each of the following claims by induction

Claim 1. *The sum of the first* n *even numbers is* n^2+n. *That is,* $\sum_{i=1}^{n} 2i = n^2+n$.

Base Case:
When $n = 1$, we want to show $P(n)$ true, where $P(1)$ is the sum of the first even integer. So, we have that $P(1) => \sum_{i=1}^{1} 2(1) = 2$. Now, we want to show that $n^2 + n$ is equal to 2 when $n = 1$. $(1)^2 + (1) = 2$. Thus, the base case has been proven.

Inductive Step:
Assume $P(k)$ is true for some fixed $k \geq n$. We want to show that $P(k + 1)$ is true. By the induction hypothesis, we assume $\sum_{i=1}^{k} 2(i) = k^2 + k$ is true. So, we want to show that $\sum_{i=1}^{k+1} 2(i) = (k + 1)^2 + (k + 1)$. So, we have

$$\sum_{i=1}^{k+1} 2(i) = (k + 1)^2 + (k + 1)$$

$$\sum_{i=1}^{k} 2(i) + \sum_{i=k+1}^{k+1} 2(i) = k^2 + 2k + 1 + (k + 1)$$

By the induction hypothesis, we can substitute as such:

$$k^2 + k + \sum_{i=k+1}^{k+1} 2(i) = k^2 + 2k + 1 + (k + 1)$$

Now, $\sum_{i=k+1}^{k+1} 2(i)$ is equal to the value $2(k + 1) => 2k + 2$, so now let us substitute and simplify.

$$k^2 + k + 2k + 2 = k^2 + 2k + 1 + (k + 1)$$

$$k^2 + 3k + 2 = k^2 + 3k + 2$$

So, we have shown that $P(k + 1)$ is true, and thus have proven the given statement true.

Claim 2. $\sum_{i=1}^{n} 3^i = \frac{3}{2}(3^n - 1)$

Base Case:
When $n = 1$, we want to show $P(n)$ true, where $P(1) => \sum_{i=1}^{1} 3^i = 3$. We want to show that $P(n) = 3/2(3^n - 1)$. Therefore, we have that $P(1) = 3/2(3^1 - 1)$, where we want $P(1)$ to equal 3. By algebra, we get that $3/2(3^1 - 1) = 3$. Thus, we proven the base case true.

Inductive Step:
Assume $P(k)$ is true for some fixed $k \geq n$. We want to show that $P(k+1)$ is true. By the induction hypothesis, we assume $\sum_{i=1}^{k} 3^i = 3/2(3^k - 1)$ is true. So, we want to show that $\sum_{i=1}^{k+1} 3^i = 3/2(3^{k+1} - 1)$. So, we have

$$\sum_{i=1}^{k+1} 3^i = 3/2(3^{k+1} - 1)$$

$$\sum_{i=1}^{k} 3^i + \sum_{i=k+1}^{k+1} 3^i = 3/2(3^{k+1} - 1)$$

By the induction hypothesis, we can substitute as such:

$$3/2(3^k - 1) + \sum_{i=k+1}^{k+1} 3^i = 3/2(3^{k+1} - 1)$$

Now, $\sum_{i=k+1}^{k+1} 3^i$ is equal to the value 3^{k+1}, so now let us substitute and simplify.

$$3/2(3^k - 1) + 3^{k+1} = 3/2(3^{k+1} - 1)$$

By algebra, we get

$$3/2(3^{k+1} - 1) = 3/2(3^{k+1} - 1)$$

So, we have shown that $P(k+1)$ is true, and thus have proven the given statement true.

Claim 3. *For any integer $n \geq 1$, $5^n - 1$ is divisible by 4. In other words, for every positive integer n there exists some constant z_n such that $5^n - 1 = 4z_n$. (Note that z_n denotes a different z for each power of 5; that is, $5^1 - 1 = 4z_1$, $5^2 - 1 = 4z_2$, and so on for a series of z_n values.) You may write your proof in general terms of divisibility by four or in specific terms by solving for z_n in the inductive case.*

Base Case:
When $n = 0$, we want to show that $P(n)$ true, such that $P(0) = 5^0 - 1$ is divisible by 4. So, we have that $5^0 - 1 = 0$, where 0 is divisible by $4 => 0/4 = 0$. So, by showing that $P(0) = 5^0 - 1$ is divisible by 4, we have proved the base case true.

Inductive Step:
Assume $P(k)$ is true for some fixed $k \geq n$. We want to show that $P(k+1)$ is true. By the induction hypothesis, we assume $5^k - 1$ is divisible by 4. We want to show that $5^{k+1} - 1$ is divisible by 4.
First,

$$5^{k+1} - 1 = 5 * 5^k - 1$$

Then,

$$5 * 5^k - 1 = (5 * 5^k - 5) + 5 - 1$$

This statement holds true since by adding 5 and subtracting 5 we are changing the right side by a net of 0, thus not changing it at all.
Next we simplify,

$$(5 * 5^k - 5) + 5 - 1 = 5(5^k - 1) + 4$$

Since we assume that $5^k - 1$ is divisible by 4, then we can see that the left side of the final form $5(5^k - 1) + 4$ is divisible by 4. And, 4 is divisible by 4 as well. So, we have shown that $P(k + 1) = 5^{k+1} - 1$ is divisible by 4.

2 Recursive Invariants

The function `maxOdd`, given below in pseudocode, takes as input an array A of size n of numbers. It returns the largest *odd* number in the array. If no odd numbers appear in the array, it returns negative infinity ($-\infty$). Using induction, prove that the `maxOdd` function works correctly. Clearly state your recursive invariant at the beginning of your proof.

```
Function maxOdd(A,n)
  If n = 0 Then
    Return -
  Else
    Set best To maxOdd(A,n-1)
    If A[n-1] > best And A[n-1] is odd Then
      Set best To A[n-1]
    EndIf
    Return best
  EndIf
EndFunction
```

Recursive Invariant:
$P(n)$ = "Returns $-\infty$ if there are no odd numbers, and returns the largest odd number otherwise."

<u>Base Case:</u>
If $n = 0$, that is that there are no values in the array passed to the function, then maxOdd(A, 0) returns $-\infty$. This is because when $n = 0$, the condition of the first If statement is true. So, the If statement evaluates "Return $-\infty$", which returns the value negative infinity, which we want.

<u>Inductive Step:</u>
Assume $k \geq n$. We want to show that maxOdd(A, k+1) returns the largest odd number that appears in the array. Since $(k + 1) > 0$, the first If condition is

false, so we go to the corresponding else statement and evaluate the code line by line. We set the variable "best" equal to the recursive call of maxOdd(A, (k+1) - 1). So, we call the function again, setting "best" equal to another recursive call of maxOdd where the integer argument is continuously subtracted by 1. This will continue until the integer argument, n, equals 0. Finally, when $n = 0$, the condition for the first If statement will be true, so we return $-\infty$. Since we have called maxOdd recursively k times, we will continue the code after the recursive call for every time that a recursive call was made. If the value at A[n-1], where n is equal to the value passed into maxOdd from the recursive call, is both greater than "best" and an odd number, then the if condition is true. Since, the if condition is true, then we will set "best" to the value at A[n-1]. The function will then return value of "best", which will get set to "best" in the maxOdd function that called it. This will continue until all the recursively called functions have ran. If there were no odd numbers in the array, then at the end of the recursive calls, "best" will get set to $-\infty$, which is the value that we want. Otherwise, it will return the largest odd number in the array.