```scheme
(define (A x y)
  (cond ((= y 0) 0)
        ((= x 0) (* 2 y))
        ((= y 1) 2)
        (else (A (- x 1) (A x (- y 1))))))
```

```scheme
(A 1 10)
(A 0 (A 1 9))
(A 0 (A 0 (A 1 8)))
(A 0 (A 0 (A 0 (A 1 7))))
(A 0 (A 0 (A 0 (A 0 (A 1 6)))))
(A 0 (A 0 (A 0 (A 0 (A 0 (A 1 5))))))
(A 0 (A 0 (A 0 (A 0 (A 0 (A 0 (A 1 4)))))))
(A 0 (A 0 (A 0 (A 0 (A 0 (A 0 (A 0 (A 1 3))))))))
(A 0 (A 0 (A 0 (A 0 (A 0 (A 0 (A 0 (A 0 (A 1 2)))))))))
(A 0 (A 0 (A 0 (A 0 (A 0 (A 0 (A 0 (A 0 (A 0 (A 1 1))))))))))
(A 0 (A 0 (A 0 (A 0 (A 0 (A 0 (A 0 (A 0 (A 0 2)))))))))
(A 0 (A 0 (A 0 (A 0 (A 0 (A 0 (A 0 (A 0 4))))))))
(A 0 (A 0 (A 0 (A 0 (A 0 (A 0 (A 0 8)))))))
(A 0 (A 0 (A 0 (A 0 (A 0 (A 0 16))))))
(A 0 (A 0 (A 0 (A 0 (A 0 32)))))
(A 0 (A 0 (A 0 (A 0 64))))
(A 0 (A 0 (A 0 128)))
(A 0 (A 0 256))
(A 0 512)
```

1024

$1024 = 2^{10}$

`(A 1 10)` is reduced 9 times to `(A 1 1)`, and each reduction is nested in an `(A 0` call. `(A 1 1)` evaluates to 2, which is 2^1. Each `(A 0` procedure evaluates to 2 times y, so with 9 `(A 0` calls, the initial y is multiplied by 2 9 times (or multiplied by 2^9, and $2^9 \cdot 2^1 = 2^{10} = 1024$).

To generalize, `(A 1 y)` will reduce to `(A 1 1)` with $y - 1$ enclosing `(A 0` calls. `(A 1 1)` evaluates to 2, which is 2^1, and each `(A 0` call doubles the value of y, which is the equivalent of multiplying 2 by 2, $y - 1$ times, or multipying 2^1 by 2^{y-1} which is equal to 2^y. Therefore, `(A 1 y)` will evaluate to 2^y.

```scheme
(A 2 4)
(A 1 (A 2 3))
(A 1 (A 1 (A 2 2)))
(A 1 (A 1 (A 1 (A 2 1))))
(A 1 (A 1 (A 1 2)))
(A 1 (A 1 (A 0 (A 1 1))))
(A 1 (A 1 (A 0 2)))
(A 1 (A 1 4))
(A 1 (A 0 (A 1 3)))
(A 1 (A 0 (A 0 (A 1 2))))
(A 1 (A 0 (A 0 (A 0 (A 1 1)))))
(A 1 (A 0 (A 0 (A 0 2))))
(A 1 (A 0 (A 0 4)))
(A 1 (A 0 8))
(A 1 16)
```

From above we know that `(A 1 16)` will evaluate to $2^{16} = 65536$.

65536

`(A 2 4)` is reduced 3 times to `(A 2 1)`, and each reduction is nested in a `(A 1` call. `(A 2 1)` evaluates to 2. This results in 3 `(A 1)` calls with the final call being `(A 1 2)` which gets reduced once to `(A 1 1)`, with the reduction resulting in an enclosing `(A 0)` call, behavior seen in the previosu problem.

`(A 1 1)` evaluates to 2, and is multiplied by 2 by the outer `(A 0` call to yield `(A 1 4)` which from above we know evaluates to 2^4. With one final outer `(A 1` call the problem reduces to `(A 1 16)` which we know is 2^{16}.
To generalize, from `(A 2 y)` we know that it will be reduced to `(A 2 1)`, and the number of enclosing `(A 1` calls will be $y - 1$. The final `(A 2` call will reduce to 2 (2^1), and each nested `(A 1` call will double that figure. That means that `(A 2 y)` will always reduce to `(A 1` 2^y`)` since $2^1 \cdot 2^{y-1} = 2^y$. From there we know that `(A 1 y)` will evaluate to 2^y, so we can say that `(A 2 y)` will evaluate to 2^{2^y}.

```
(A 3 3)
(A 2 (A 3 2))
(A 2 (A 2 (A 3 1)))
(A 2 (A 2 2))
(A 2 (A 1 (A 2 1)))
(A 2 (A 1 2))
(A 2 (A 0 (A 1 1)))
(A 2 (A 0 2))
(A 2 4)
```

From above we know that `(A 2 4)` evaluates to 65536.
65536

```
(define (f n) (A 0 n))
(define (g n) (A 1 n))
(define (h n) (A 2 n))
(define (k n) (* 5 n n))
```

`(k n)` computes $5n^2$, `(f n)` computes $2n$, `(g n)` computes 2^n, and `(h n)` computes 2^{2^n}.