Problem 2.1

1.

```
procedure argmax(L,f):
input: An array, L , of objects to be plugged into the given function, f.
output: The object in the array L that produces the highest value when plugged
into the function f


maxIndex = 0

for x in L.length:
    if f(L[x]) > f(L[maxIndex]):
        maxIndex = x

return L[maxIndex]
```

2.

I believe the Big-O runtime for *argmax* is Big-O(Rn), where R is the worst case runtime for f(x) and n is the number of elements in the array L. Because Big-O is an upper bound, we assume that each element in L is going to experience the worst case run time, which is R. Thus, since we have a for loop iterating through each element in L, our runtime will roughly be equal to the amount of time each of these iterations will take (R), multiplied by the number of iterations (n). Although the for loop has two calls to f(n), the runtime is not 2Rn, as the 2 is a constant.

Problem 2.2

$$f(n) = 10 + 2^3 \log_2 n + 5^{8 \log_5 n} \tag{1}$$

$$f(n) = n^8 + n^3 + 10 \tag{2}$$

Big-Ω(n) = n^8 Proof:

T_A(n) is Big-Ω(T_B(n)) if positive integers c and n_0 exist such that $T_A(n) \geq c * T_B(n)$

Thus,

let $T_A(n) = n^8 + n^3 + 10$

let $T_B(n) = n^8$

$n^8 + n^3 + 10 \geq c * n^8$

let c = 2

$n^8 + n^3 + 10 \geq 2n^8$

$n^3 + 10 \geq n^8$

This inequality is true for n $\leq$ 1.3724

Thus, Big-Ω(n) = n^8

Big-O(n) = n^8 Proof:

T_A(n) is Big-O(T_B(n)) if positive integers c and n_0 exist such that $T_A(n) \leq c * T_B(n)$

Thus,

let $T_A(n) = n^8 + n^3 + 10$

let $T_B(n) = = n^8$

$n^8 + n^3 + 10 \leq c * n^8$

let c = 5

$n^8 + n^3 + 10 \leq 5n^8$

$n^3 + 10 \leq 4n^8$

This inequality is true for n $\geq$ 1.14093

Thus, Big-O(n) = n^8

Big-Θ(n) = n^8 Proof:

T_A(n) is Big-Θ(T_B(n)) if it is Big-O(T_B(n)) and Big-Ω(T_B(n))

Since the Big-O and Big-Ω values for f(n) are both equal to n^8, Big-Θ(n) must also be equal to n^8