Problem 3

(a) The way that we can kind the minimum value is by differentiating, since f(θ) is a parabola and therefore has a minimum value at the vertex, given that w_i are positive. Doing so yields:

$$f'(\theta) = \frac{1}{2}\sum_{i-1}^{n} w_i * 2(\theta - x_i)$$

By equaling to zero and solving for theta, we achieve the minimum value of theta:

$$\theta = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$$

If some of the wi's are negative, we might not actually have a single minimum value for the parabola and therefore might not be able to minimize θ.

(b) We can solve this by writing some equivalences. Note:

$$f(x) = \sum_{i=1}^{n} |xi|$$

$$g(x) = \left|\sum_{i=1}^{n} xi\right|$$

Referencing the triangular inequality, we can see that

$$\sum_{i=1}^{n} |xi| \geq \left|\sum_{i=1}^{n} xi\right|$$

and therefore:

$$f(x) \geq g(x)$$

(c) Let's say the expected number of points is S. If in the first throw the number is 6, *b* points are added, and when the number is 2, *a* points are subtracted. If the outcome is anything else other than 1, we know that we will continue to roll. The recursive relation can be written as:

$$S = \{Pr(6) * b\} - \{Pr(2) * a\} + \{[1 - Pr(1)] * S\} \ldots (until\ we\ get\ a\ 1)$$

$$S = \left\{\frac{1}{6} * b\right\} - \left\{\frac{1}{6} * a\right\} + \left\{\left[1 - \frac{1}{6}\right] * S\right.$$

$$S = \frac{b}{6} - \frac{a}{6} + \frac{5S}{6}$$

$$S = b - a$$

(d) Using logarithmic rules:

$$\log(p) = 4\log(p) + 3\log(1 - p)$$

Taking the derivative with respect to p,

$$\frac{\partial \log\big(L(p)\big)}{\partial p} = \frac{4}{p} - \frac{3}{1 - p}$$

Solving that for 0, $p = \frac{4}{7}$, which is the maximum value for L(p).

(e) After much calculation and scratch work on paper, the gradient of f(w) can be represented as:

$$\nabla f(w) = 2\sum_{i=1}^{n}\sum_{j=1}^{n}(a_i^T + b_j^T)^2 w + \lambda w$$

Problem 4

(a) I think that since we probably have to be processing each rectangle in the image that is n*n pixels, the total amount of rectangles in the square image is $n(n+1)n(n+1)$, according to the amount of body parts. From this we can clearly notice that the complexity of this is $O(n^4)$.

(b) We can represent the recurrence relation in terms of the costs when we are moving. Since we can only move either down or right:

$$v(i,j) = c(i,j) + \min\left(v(i-1,j), v(i,j-1)\right)$$

As stated in the problem, we see that there are repeated computations, which we can avoid. We do this by creating a memo table with all values initialized to -1, so that when that value is not -1 we know we have already computed that subproblem and we can just return the value already stored in the table. This means that we are only computing each subproblem once and reducing the complexity to $O(n^2)$. In pseudocode, it would look something like:

```
Create table with -1 values
v(I, j):
        if table[i][j] is not -1: return table[i][j]
        else:
                table[i][j] = c(i, j) + min(v(i-1, j), v(i, j-1)
return table[i][j]
```

(c) To solve this, let f(n) describe the different ways we can reach the top step. We see that:

$$f(1) = 1$$
$$f(2) = 2$$
$$f(3) = 4$$
$$f(4) = 8$$

We can see that there is a pattern for the amount of ways we can reach the top, which is given by:

$$f(n) = 2^{n-1}$$

(d) What we can do it first find the magnitude of the vector (column) sum of w, which would be the preprocessing. After that, we can compute f(w) in $O(d^2)$ time.