Solution

a) Let $opt(j)$ be the number of triangulations of a convex $j-$gon. We know that polygon edge $p_j p_1$ must form a triangle with p_i for some i, where $1 < i < j$. Our base case would be if $j < 4$ then $opt(j) = 1$. We then take i from $2 \rightarrow j-1$ and we have: $opt(j) = \sum_{i=2}^{j-1}$ if $i < 4$ then $opt(j-i+1)$ else if $j-i < 3$ then $opt(i)$ else $opt(j-i+1) + opt(i)$.

```
def opt(j):
        r=0
        if (j<4):
                return 1
        else:
                for i in range(2,j):
                        if i < 4:
                                r+= opt(j-i+1)
                        elif j-i<3:
                                r+= opt(i)
                        else:
                                r += opt(j-i+1)+opt(i)
                return r
```

Time Complexity: All possible $i : O(j)*$ At most j recursive calls per iteration: $O(j) = O(j^2)$

Proof. We will prove that this dynamic programming algorithm works for all $j - gon$.
Base Case: Let $j = 3$. Triangles only have one element in their triangulation set as they have $3 - 3 = 0$ crossing diagonals. Our algorithm computes 1 for $j = 3$, so it holds for this base case.
Inductive Hypothesis: Assume our algorithm computes the right number of triangulations up to a $k - gon$, where $k \geq 3$.
Inductive Step: We must show that our algorithm computes the correct number of triangulations for a $k + 1 - gon$. It is given that polygon edge $p_{k+1}p_1$ must form a triangle with p_i for some i, where $1 < i < k + 1$. Our algorithm goes through every possible value of i. When $i = 2$, we have a diagonal from p_{k+1} to p_2. This leaves us with a triangle $p_{k+1}p_1p_2$ and a $k - gon$. From our inductive hypothesis, we know our algorithm provides the correct number of possible triangulations for a $k - gon$, we only add this number as triangles have one possible triangulation. Similarly, when $i = 3$ we add the triangulations of the $k - 1 - gon$ found by our algorithm based on our IH. A similar thing happens when i approaches $k + 1$. Triangles are made between $p_1p_{k+1}p_i$ and the $k + 1 - gon$ is divided into at least one triangle and a $k - gon$ or $k - 1 - gon$. We use our IH to add the possible triangulations of these two shapes. Finally, the triangle $p_1p_{k+1}p_i$ divides the $k + 1 - gon$ into two shapes of less than $k + 1$ sides. From our IH we take the sum of triangulations of both polygons and ignore the triangle. By splitting our $k + 1 - gon$ into smaller shapes, we are able to use our IH to calculate the number of triangulations of a $k + 1 - gon$. □

b) Let $opt(j, a, z)$ be the weight of a minimum weight triangulation of P. It is obvious that the weight of the minimum triangulation of a triangle is zero, because the triangulation contains no edges. We will use $w_{j,i}$ to denote the weight of the edge between p_jp_i. If said edge is not a part of a triangulation T, then $w_{j,i} = 0$. We will walk through vertices $p_2 \rightarrow p_{j-1}$, adding the weight of the edges of the triangle formed by $p_jp_ip_1$, and any other edge found in the shapes found after this split. We take the minimum weight we find after all possible triangulations. We can build the following recurrence:
$opt(j, a, z) = \min_{a<i<z}\{w_{a,i} + w_{z,i} + opt(j - i + 1, i, z) + opt(i, a, i)\}$ where a and z will be initialized to 1 and j respectively.
Time Complexity: All possible $i : O(j) * j$ recursive calls per iteration = $O(j^2)$.
Space Complexity: Array of size $j : O(j)$.

Proof. We will proof that our dynamic programming algorithm finds the weight of a minimum weight triangulation of P.
Base Case: A triangle has a minimum weight triangulation of zero because its triangulation has no edges. Our algorithm will return zero, so it works for this case.
Inductive Hypothesis: Assume our algorithm returns the weight of a minimum weight triangulation of a convex polygon with up to k sides, where $k \geq 3$.
Inductive Step: We must show our algorithm returns the desired output for a convex polygon P^* with $k + 1$ sides.

As we use i from 2 to k, we add the weight of the edge from p_1 to p_i and from p_{k+1} to p_i to our total weight for this triangulation (Note that edges not in a triangulation of P^* have a weight of zero). These edges divide P^* into at most three polygons of less than $k+1$ edges. Using our Inductive Hypothesis we are able to determine the weight of a minimum weight triangulation of these shapes. Once these minimum weights are added to the weight of the edges connected to i, we increase i until it reaches $k-1$. We were now able to compare all of the weights of all triangulations of P^* and can easily determine the smallest one. Thus, our algorithm can find a correct weight of a minimum weight triangulation of any convex polygon. $\square$

c) An $opt[]$ array can be filled as the algorithm is running so $O(j^2)$ time and $O(j)$ space. Therefore, this modification only adds space complexity. From the array, one can extract a minimum weight triangulation M of P by moving backward through the array: Starting with $i = 2$, $edge_{j,2} \in M$ if $opt[j] = w_{1,2} + w_{j,2} + opt(j-2+1, 2, j) + opt(2, 1, 2)$. Letting $i = 3, 4, ..., j-1$ then $edge_{j,i}$ and $edge_{1,i} \in M$ assuming $opt[j] = w_{1,i} + w_{j,i} + opt(j-i+1, i, j) + opt(i, 1, i)$ and $edge_{j,i-1}, edge_{j,i+1}, edge_{1,i-1}, edge_{1,i+1} \notin M$; otherwise $edge_{j,i}, edge_{1,i} \notin M$. Looking this up will add $O(j)$ to run time but will not affect the asymptotic run time of the algorithm.