COMP3821 Assignment 2

1.1

Define the subproblem $P(i)$, where $i \in \{1, 2, \ldots, n\}$ represents the number of slices of pizza remaining. The subproblem is now to minimise the amount of topping lost, $L(i)$, after having taken $n - i$ pieces of pizza, so we arrive at i slices remaining. The rules for topping loss and slice choosing remain the same as in task T.

1.2

For the recurrence relation, we have the optimal solution to $P(i)$, let it be $S(i)$, which is the minimum amount of topping lost to arrive at i pieces remaining. Then the minimum amount of topping lost in choosing piece i, denoted by $S(i-1)$, to arrive at $i-1$ pieces left is constructed by choosing the piece with minimum topping loss that is adjacent to the empty space. This is expressed in the relationship below, where subscripts l and r represent the index of the slices on the left and right ends of the empty space.

$$S(i-1) = \min\{S(i) + a_l + b_l \times i, S(i) + a_r + b_r \times i\}$$

The base case is when n slices remain, which is given by

$$S(n) = a_i + b_i \times n$$

where i is the index of the slice we are considering starting from.

1.3

In the pseudocode below, n is a global variable for the size of the pizza, a, and b are global arrays containing all a_i and b_i respectively. l and r are the index of the slices on the left and right of the empty space.

```
procedure LOST(k, s, l, r, lost)
    if k = n then
        return a_s + b_s × n
    end if
    if b_l < b_r then
        add ← a_l + b_l × k
        l ← l − 1  mod n
    else
        add ← a_r + b_r × k
        r ← r + 1  mod n
    end if
    return lost + add + LOST(k + 1, s, l, r, lost)
end procedure
```

```
procedure MinimumLost
    min ← ∞
    for all i in 1...n do
        l ← l − 1  mod n
        r ← r + 1  mod n
        check ← Lost(0, i, l, r, 0)
        if check < min then
            min ← check
        end if
    end for
    return min
end procedure
```

1.4

The worst case complexity is $O(n^2)$.

2.1

The cost of the configuration outlined is $5 + 2 + 4 + 2 + 5 + 1 = 19$.

2.2

The set of legal swaps, $\{(1, 4), (7, 8)\}$, gives the configuration with the smallest cost.

2.3

Define the subproblem $P(i)_j$, where n is the size of the configuration, and $i \in \{0, 1, \ldots, n\}$ represents the i-th black marble in the configuration. Let j be the number of swaps performed with the i black marbles. Thus, $j \in \{0, 1, \ldots, i\}$, The subproblem is now to minimise the distance of the sub-configuration, given by the first i black marbles of the overall configuration, by using only j swaps. When constructing the sub-configuration, only remove the black marbles that are not in the first i black marbles when counting from the left. The same rules apply as defined in the overall problem. Let the optimal solution $S(i)_j$ to $P(i)_j$ have the minimum distance $D(i)_j$.

2.4

Let m_k be the index of the k-th marble, h_k be the index of the matching box for the marble at m_k, and b_i be the index of the i-th black marble. The recurrence relation is given by the following formula.

$$D(i+1)_j = \{\min\{D(i)_j - \Big[|m_k - h_k| - |h_k - b_{i+1}|\Big], D(i)_j\} \ : \ \forall k \in \{1, \ldots, i+1\}, \ \forall j\}$$

The formula is essentially computing the change obtained by swapping a marble and the $(i + 1)$-th black marble, and determining if extending a previous solution is best (as long as the swap is legal), or keeping the previous solution is best.

The base case is given when $i = 0$, that is the subproblem has no black marbles in the sub-configuration. This also forces $j = 0$. Thus $D(0)_0$ is simply the cost of the configuration as it stands, as no swaps are able to be performed.

2.5

The algorithm solves each subroblem and stores in a 2-D array for each solution $D(i)_j$, indexed by (i, j). Thus computing the solution to $P(i)_j$ is performed in at most $O(n^2)$ time, as we must loop through all previous solutions. Overall the algorithm runs in $O(n^3)$ time as there are at most n black marbles. Obviously this is not the $O(n^2)$ time asked for, however I could not think how to reduce this solution down to $O(n^2)$ time.

3.1

Let each vertex $v \in V$ represent a single polygon in the plane. Let an edge $e \in E$ be the unordered pair (v_1, v_2) for $v_1, v_2 \in V$. Each edge connects the vertices whose polygons share an edge in the plane.

3.2

Each vertex $v \in V$ contains the variables b_v and r_v. Each of these variables are binary variables, meaning they can hold the value of either 0 or 1. If v is coloured blue, $b_v = 1$, if v is coloured red, $r_v = 1$, and if v is uncoloured, $b_v = 0$ and $r_v = 0$. Clearly, b_v and r_v both cannot be 1.

3.3

For each edge $(u, v) \in E$, we have the constraints

$$b_u + b_v \leq 1$$
$$r_u + r_v \leq 1$$

For each vertex $v \in V$, we have the constraint

$$b_v + r_v \leq 1$$

3.4

The objective function we wish to maximise is

$$\sum_{v \in V} (b_v + r_v)$$

4.1

It is better to model the problem with Integer Linear Programming, as we wish to determine the number of bags in each of the p stacks, and we cannot have part of a bag in one stack, and another part in another stack.

4.2

Let b_{iq} be a binary variable that takes the values of either 0 or 1. If $b_{iq} = 1$, this indicates that the i-th bag is in the q-th stack. Let $h(q)$ be the height of the q-th stack. It is given that there are n bags, p stacks, and each stack may have a height of h bags. The final variable is the force given by the i-th bag, in the q-th stack, denoted by f_{iq}.

4.3

For each stack of bags, we have the constraint

$$\sum_{i=1}^{n} b_{iq} \leq h$$

For each bag, we have the constraint

$$\sum_{q=1}^{p} b_{iq} = 1$$

4.4

The objective function that we wish to minimise is

$$\sum_{i=1}^{n} \sum_{q=1}^{p} b_{iq} f_{iq}$$