**GROUP ASSIGNMENT 2**

**ALGORITHM 1: ENUMERATION**

ALGORITHM\_ONE(n, m, t, M[1..m], T[1..t])  
 leastOpenedLockers <- oo  
 unopened <- 0  
 keys[1] <- 0  
  
 while k != 1  
 unopened <- 0  
 if keys[k] < m  
 keys[k + 1] <- keys[k] + 1  
 k++  
 else  
 keys[k-1]++  
 k--  
  
 if(T[1] < M[1])  
 unopened += T[1] - 1   
 if(T[t] > M[m])  
 unopened += n - T[t]   
  
 ballCounter <- 0  
 while(T[ballCounter] < keys[1])  
 ballCounter++  
  
 z <- 1  
 for z up to k-1   
 bestE <- 0  
 i <- keys[z] + 1  
 for i up to keys[z+1] - 1   
 j <- i  
 if(i = T[ballCounter])  
 i++  
 ballCounter++  
 continue  
 else  
 while(j+1 != T[ballCounter])  
 j++  
 if ((j-i)+1) > bestE  
 bestE <- ((j-i)+1)  
 i = ballCounter   
 unopened += bestE  
 if(total - unopened < leastOpenedLockers)  
 leastOpenedLockers = total - unopened  
 return leastOpenedLockers

The algorithm begins by creating a powerset of all the keys. This process takes Θ(2^m) time. Then, for each key in each combination, we check for the largest number of consecutive, empty lockers, and we subtract them from the lockers between the keys. This process is done in Θ(n) time, and it is done for each key combination, so we end up with Θ(n2^m) time.

When we implemented this in Python, we got the following results:

3, 4, 5, 11, -70.

It’s most likely an implementation error as to why we didn’t get the correct solutions for the last sample sets.

**ALGORITHM 2: RECURSIVE**

ALGORITHM\_TWO\_RECURSIVE(n, m, t, M[1..m], T[1..t])  
 part = d(n)  
 if T[t] > K[k]  
 return part + (T[t] - K[k])  
 else  
 return part

d(i)  
 //BASE CASE  
 if M[0] < T[0]:  
 return 0  
 else:  
 return M[0] - T[0]  
  
 return min{d(j) + LEAST\_OPENED(M[i], M[j])} for all j < i  
  
LEAST\_OPENED(mi, mj)  
 best <- 0  
 for i in [mi..mj]  
 j <- i  
 if i in T  
 continue  
 else  
 while (j+1) not in T  
 j++  
 if (j-i) + 1 > best  
 best <- (j-i) + 1  
 return (mj - mi) - best

We split this algorithm into helper functions, so d(i) is actually the ‘recursive’ algorithm in this function. That way we can handle the right-most key after we handle d(i).

The function itself will be calling d(n), which will recursively call itself Θ(2^m) times. The helper function LEAST\_OPENED goes just finds the largest empty consecutive set of lockers, and returns the distance between the keys without that length, essentially counting the least lockers that need to be opened. Least opened is done in Θ(n) time, and it is done in every recursive call, so we get a total running time of Θ(n2^m), just like our enumeration one.

**ALGORITHM 2: DYNAMIC**

ALGORITHM\_TWO\_DYNAMIC(n, m, t, M, T):  
 D <- []  
  
 for i <- 0 up to m  
 D[i] <- infinity

if M[0] <= T[0]:  
 D[0] = 0  
 else:  
 D[0] = M[0] - T[0] + 1  
  
 for i <- 1 up to m  
 for j <- 0 up to i  
 leastOpened = LEAST\_OPENED(M[i], M[j])  
 if D[j] + leastOpened < D[i]  
 D[i] = D[j] + leastOpened  
  
 #-----second key  
 if T[t-1] >= M[m-1]:  
 D[m-1] += (T[t-1] - M[m-1]) + 1  
  
 return D[m-1]  
  
LEAST\_OPENED(mi, mj):  
 bestUnopenedCount = 0  
 if mi - mj == 1  
 if mi in self.\_tennisBalls  
 if mj in self.\_tennisBalls  
 return 1  
 else  
 return 0  
 else  
 if mj in self.\_tennisBalls  
 return 1  
 else  
 return 0  
 else  
 for i <- mj up to mi  
 j <- i  
 if i in T  
 continue  
 else  
 while (j+1) not in T and j < mi-1:  
 j += 1  
 if (j-i) + 1 > bestUnopenedCount  
 bestUnopenedCount = (j-i) + 1  
 return (mi - (mj+1) + 1) - bestUnopenedCount

This algorithm reverses the recursion from the previous algorithm by building up a table D, from the bottom up. The base case is the first key in the table. In the main function, we are looping through all of the keys twice to try to find the minimum number of keys needed to open D[i]. This is where the Θ(m^2) comes from. Inside of this, we have to look through every locker between keys to find the best way to open the lockers. This takes Θ(n) time.

The runtime analysis of this problem is Θ(nm^2):

When we implemented this in Python, we got the following answers:

97, 22, 64, 31, 103, 31, 87