Group Project 2

Kevin Stine

Section 1: Introduction

The problem we are trying to solve revolves around lockers and tennis balls. There are N number of lockers, and there are random keys placed within those lockers. A copy of the key to each locker are in the adjacent lockers, so the locker to the left and right hold the keys. Inside certain lockers are tennis balls. The problem we need to solve is retrieving all of the tennis ball by opening the least amount of lockers as possible.

Section 2: Algorithm 1 Enumeration (Brute-force algorithm)

2.1: Pseudocode for algorithm 1

```
algorithmOne(n, m, t, M[1...m], T[1...t])
        leastOpenedLockers = infinity
        unopened lockers = 0
        keys[1] = 0
        while k != 1
                unopened lockers = 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]
                ballCount = 0
                while (T[ballCount] < keys[1])
                        ballCount++
                var = 1
                for var to k-1
                        bestEmptySet = 0
                i = keys[var] + 1
                for i to keys[var+1] - 1
                        j = i
                        if (i = T[ballCount])
                                i++
```

```
                    ballCount++
                else
                    while(j+1 != T[bacllCount])
                        j++
                    if (j-i) + 1 > bestEmptySet
                        bestEmptySet = (j-i) + 1
                    i = ballCount
                unopened lockers += bestEmptySet
            if (total - unopened lockers < leastOpenedLockers)
                leastOpenedLockers = total - unopened
        return leastOpenedLockers
```

2.2 Explanation of pseudocode

This pseudocode runs through the brute-force method of calculating the algorithm. It starts by creating a powerset, the set of all subsets, of all the keys. Once the powerset is created, the algorithm then checks each key in the combination and checks the largest number of consecutive empty lockers. The largest number of consecutive empty lockers are then subtracted from the lockers between the keys.

2.3 Analysis of runtime

The process of creating the powerset for all the keys takes $\Theta(2^m)$. Checking the largest number of consecutive empty lockers is done for all of the elements, which is done in $\Theta(n)$ time. Therefore we have an overall running time of $\Theta(n2^m)$.

2.4 Solutions to the corresponding input instances

The enumeration algorithm reads from the file dp_set1.txt which has been modified with spacing so the program can read the file. The results of the enumeration algorithm are:
11, 12, 6, 14, 17, 1, 10, 5.

A few of these results are incorrect. Some of the results are off by one or two lockers, which is probably caused by an error in the implementation.

Section 3: Algorithm 2 Dynamic Programming

3.1 Pseudocode for algorithm 2

```
algoritmTwo(n, m, t, M[1...m], T[1...t])
        D = []
        if t == 1 //Edge case for one tennis ball
                for i in range(0, m)
                        if M[i] == T[0]
                                return 1
        #First Key
        if M[0] <= T[0] //Base case/
                return 0
        else
                return M[0] - T[0] + 1
        #Middle Key
        for i in range (1 to m)
                for j in range(0 to i)
                        return openLeastLockers(M[i], M[j])
        #Second Key
        for T[t-1] >= M[m-1]
                D[m-1] += T[t-1] - M[m-1] + 1
        return D[m-1]

openLeastLockers(self, mi, mj)
        bestUnopenedCount = 0
        if mi - mj == 1
                if mi in t
                        if mj in t
                                return 1
                        else
                                return 0
                else
                        if mj in t
                                return 1
                        else
                                return 0
        else
                for i in range mj+1 to mi
                        j = i
                        if i in t
                                continue
                        else
```

```
            while (j+1) not in t and j < mi - 1
                    j++
            if (j-i) + 1 > bestOpenedCount
                    bestUnopenedCount = (j-i) + 1
    return (mi - (mj+1) + 1) - bestUnopenedCount
```

3.2 Explanation of pseudocode

This algorithm is split up into a couple sections. There is the main algorithmTwo function as well as a helper function called openLeastLockers. The main algorithmTwo function handles D[i] which implements the recursive section of this algorithm. It is split up into a couple sections, finding the first key, the middle key and then the second key. Since this algorithm is recursive, it calls itself $\Theta(M^2)$ times.

The openLeastLockers helper function finds the largest empty consecutive set of lockers and returns the distance between the keys. This helper function runs in $\Theta(N)$ time.

3.3 Analysis of runtime

Since the algorithmTwo runs in $\Theta(M^2)$ time and openLeastLockers runs in $\Theta(N)$ time, which is called each time for every recursive call, we have a runtime of $\Theta(NM^2)$.

3.4 Solutions to the corresponding input instances

The dynamic programming algorithm reads from the file dp_set2.txt which has been modified with spacing so the program can read the file. The results of the dynamic programming algorithm are:
97, 22, 64, 31, 103, 31, 87, 81

Section 5: Conclusion

The algorithms are made up differently, as the dynamic programming algorithm utilizes recursion. Dynamic Programming runs in $\Theta(NM^2)$ while enumeration runs in $\Theta(n2^m)$. Dynamic programming also has a helper function which gets called for each recursive call.