Assignment - 4

Rahul Motan

1.a) **Algorithm**

Step 1: Create an array $A′\_{j}$ for each project from $A\_{j}$ such that $A′\_{j}\left[k\right]=A\_{j}\left[i+1\right]−A\_{j}\left[i\right]$ $∀ 0\leq i<m$

Step 2: Create an array $MAT\left[j\right]\left[1\right]$ for each project $: 1\leq j\leq n$

Step 3: Pick the programmer such that $max\left(A′\_{0\leq j\leq n}\left[i\right]∀ 0\leq i\leq m−1\right)$

Step 4: mark $i^{th}$ programmer’s benefit as 0.

Step 5: For the selected  $j \in A′\_{j}$ in Step 3, increment $MAT\left[j\right]\left[1\right] by 1$

Step 6: Repeat Step 3 until $sum\left(MAT\_{j}\right)=m$

Step 7: Return $MAT\_{j}$

1 b) **Greedy Choice:**There is an optimal solution containing k programmers. Assume Y\* is an optimal solution and Y does not contain k programmers. Since Y is a solution, there are j programmers in Y. We can replace j by k if k gives more benefit over j to obtain the claimed optimal solution.

We want to show that optimal solution always contains the greedy choice.

Consider total ‘k’ programmers allocated to ‘Y’ projects $: Y\leq n$

Each ‘j’ project $\in Y$ must contain a certain number of programmers with maximum benefit. If the value picked does not offer maximum local benefit we then replace that number with the number that offers maximum local benefit.

This can continue until all the m programmers are allocated.

If the selected programmers with highest local maximum is more than what can be allocated then select programmers that can be allocated. Thus an optimal solution will always include greedy choice.

**Optimal Substructure:**We want to show that an optimal solution contains within it optimal solutions to subproblems.

Consider the maximum benefit giving ‘m’ programmers to be allocated to a set of $Y$projects such that  $Y\leq n$ such that we get maximum benefit say, $OPT\left(n,m\right)$ which is an optimal solution.

Remove total ‘k’ programmers out of these ‘m’ allocated programmers.

Let the remaining programmers be $k′=m−k$ be allocated to a set of $Y^{\*}$ projects such that $Y^{\*}<Y\leq n$. These must be the programmers who incur maximum benefit over this smaller allocation problem. Let this benefit be $OPT^{\*}\left(k′,n′\right)$.

Let there be another $OPT\left(k′,n′\right): OPT\left(k′,n′\right)<OPT^{\*}\left(k′,n′\right)$

Adding back those k programmers over the entire n projects…

$OPT\left(k′,n′\right)+\left(n−n′\right)k<OPT^{\*}\left(k′, n ′\right)+k\left(n−n′\right)$

$OPT\left(n,m\right)<OPT^{\*}\left(n,m\right)$

This is a contradiction since $OPT\left(n,m\right)$ is the maximum benefit for the optimal allocation of m programmers over n projects. Hence we prove the optimal substructure property.

2. a) The given problem is a problem of finding the smallest-set of unit length closed intervals that contains a given set $\left\{x\_{1},x\_{2},.....,x\_{n}\right\}$ of points on the real line. Here instead of unit length we have the light pannel width and the points correspond to the paintings.

b)  Psuedo-Code:

Create an array to store the point of placing the light of size ‘n’. A[0…n]

Step 1: Sort the given array of points.

Step 2: Initialize ***coverage*** which stores the amount of wall that we have lit. We start with placing the light such that coverage is [x1,x1+m].

Step 3: Increment ***panelCount*** by 1

        ***PlaceLights(Points[ ], panneWidth){***

            $sort\left(Points\right)$ // non-decreasing order …………………………………………………………. $nlgn$

            $coverage \leftarrow Points\left[0\right]+pannelWidth$…………………………………………….. $c\_{1}$

                $A\left[0\right]\leftarrow  Point\left[0\right]+\frac{pannelWidth}{2}$

            $panelCount\leftarrow 1$………………………………………………………………………………………..$c\_{2}$

          $for i\leftarrow 1 to Points.Length−1$:

                    $if Points\left[i\right] > coverage then$………………………………………………………… $c\_{3}$

                                $gap \leftarrow  Points\left[i\right]−coverage$………………………………………………….. $c\_{4}$

                                $coverage\leftarrow coverage+gap+pannelWidth$………………………. $c\_{5}$

                                $A\left[i\right]\leftarrow  Point\left[i\right]+\frac{\left(pannelWidth\right)}{2}$………………………………………………………………………. $c\_{6}$

        $return A$

***Running Time:***

T(n) = $nlgn+c\_{1}+c\_{2}+\left(n−1\right)\left(c\_{3}+c\_{4}+c\_{5}+c\_{6}\right)$

$T\left(n\right) = O\left(nlgn\right)$

***Space-Complexity:  O(n)***

c) **Greedy-Choice Property :**The problem here is to pick the best position of each light panel so that it maximizes coverage over paintings but minimizes total no. of light panels. So as we start at the first painting we place the light panel such that it lights the wall from $x\_{1} to x\_{1}+m$. Our placement of lights does not look over the paintings over the un-lit area of the wall or the remaining paintings over the wall. If there is a painting that does not need a light as it already is covered within our light coverage then we skip that painting. Thus it exhibits greedy choice property.

**Optimal-Substructure:**At each iteration we check if we need to increase the light coverage over the wall, we do this only if the next painting lies outside of our current coverage area. So if we place a light panel, we reduce the unlit paintings over the wall which is our subproblem. Thus it depicts optimal sub-structure.

Let $OPT\left(j\right)$ be the optimal solution that lights $n$ paintings with the minimum no. of light panels.

Let $OPT\left(j−1\right)$ be the optimal solution that lights $n−1 paintings$ with  the minimum no. of light panels which is a

sub-problem to the given problem.

Let there exist another solution $OPT^{\*}\left(j−1\right)$ for the above problem which gives lesser no. of light panels.

To light the $n^{th}$ painting…

$OPT^{\*}\left(j−1\right)+panel\left(x\_{n}\right)<OPT\left(j−1\right)+panel\left(x\_{n}\right)$

$OPT^{\*}\left(j−1\right)+panel\left(x\_{n}\right)<OPT\left(j\right)$

This is a contradiction as OPT(j) is an optimal solution which lights the n paintings with minimum no. of lights. Hence we prove the optimal sub-structure property.

3a.)

This solution is optimal as it uses dynamic programming which calculates each sub-problem optimally. We need to create milk quantity using minimum no. of vials. Each smaller quantity created optimally will help create bigger quantity optimally. For eg if we had used greedy instead of DP we’d be making a quantity using the greatest value always which is not optimal in this case we might have to use more smaller vials for the remaining portion than if we had opted for the optimal choice.

**Algorithm:**

We have an array that contains vials with some quantity $v\_{i}$,$VIALS\left\{v\_{1},v\_{2},....v\_{n}\right\}$ of length V and their corresponding amount is stored in an array say, $COUNT\left\{c\_{1},c\_{2},c\_{3},....,c\_{n}\right\}$, To dispense the $m$ amount using minimum vials we’ll follow this algorithm.

Sort the $VIALS$ array in descending order of quantity$\leftarrow $ $nlgn$

$Create a memo table M\left[0. . .m+1\right]$$\leftarrow $ $c\_{1}$

$Create Inventory Memo table IT\left[0..m+1\right]\left[VIALS.Length\right]$ . This table saves the state of inventory for each coin and amount.

$M\left[0\right]\leftarrow  0$                                                                        $\leftarrow c\_{2}$

$Q\_{dispensed} \leftarrow  0$                                                            $\leftarrow c\_{3}$

$M\left[1....m+1\right]\leftarrow \infty $                                                $\leftarrow c\_{4}m$

$for i\leftarrow 1 to m$

                $M\left[i\right]\leftarrow \infty $

            $for j\leftarrow 0 to VIALS.Length−1$

                $if i\geq VIALS\left[j\right]$    $AND IT\left[i−VIALS\left[j\right]\right]\left[j\right]<COUNT\left[j\right]$                  $\leftarrow  c\_{6}$

                        $amount \leftarrow  M\left[i−VIALS\left[j\right]\right]$       $\leftarrow  c\_{7}$

                        $if amount != \infty  AND amount+1<M\left[i\right]$      $\leftarrow c\_{8}$

                                    $M\left[i\right]\leftarrow  amount+1$                                                         $\leftarrow c\_{9}$

                                    $IT\left[i\right] = IT\left[i−VIALS\left[j\right]\right]\left[j\right]$                                                                     $$

                                    $IT\left[i\right]\left[j\right]\leftarrow IT\left[i\right]\left[j\right]+1$                                                           $\leftarrow  c\_{11}$

$if M\left[m\right] == \infty $

        return  $Not Possible$

$else$

        return $M\left[m\right]$

**b) Running Time**: $O\left(m\right)$ as V is predefined constant.

**Space Complexity** : $O\left(m\right)$

**Correctness of Algorithm**

**Recurrence Relation:**

                                                    $OPT\left(j\right)=0     for j=0$

                                                    $OPT\left(j\right) = min\_{i:VIALS\_{i}\leq j}\left\{OPT\left(j−VIALS\_{i}\right)+1\right\}      for j>0$

**Overlapping Sub-problems:**We can solve the above problem using recursion by taking care of the base cases but every recursive calls computes the sub-problem several times. Thus this has overlapping sub-problems. In our solution we solve all the sub-problems once and only look back to compute bigger sub-problem.

**Optimal Sub-structure:**Let $OPT\left(n\right)$ be the optimal solution which gives the minimum no. of vials ‘n’ for a given milk quantity ‘V’.

                Let $OPT\left(n−1\right)$ be the optimal solution which gives the minimum no. of vials ‘n-1’ for a given milk quantity $V′: V′<V$

Let there exist another solution $OPT^{\*}\left(n−1\right)$ that gives the solution to the problem with lesser use of vials.

To make the given quantity $V$

$OPT^{\*}\left(n−1\right)+VIAL\left\{v\_{n}\right\}<OPT\left(n−1\right)+VIAL\left\{v\_{n}\right\}$

$OPT^{\*}\left(n−1\right)<OPT\left(n\right)$

This is a contradiction since $OPT\left(n\right)$ is the optimal solution that gives the minimum no of vials for the given quantity $V$. Hence we prove the optimal substructure property i.e an optimally created quantity contains within it, the optimally created sub-quantities .

c)We can determine whether we can dispense a given amount by checking these conditions:-

    a)  if the initially set value in the memo table remains unchanged, i.e it stays infinity we know that we cannot dispense that amount             via the given quantities of the vials

    b) If the required quantity cannot be created by using all of the vial i.e it is so much greater than the total quantity we have using all            the vials.