CS-2009 Design and Analysis of Algorithms

Assignment # 3 Part A

Roll #: 22L-6552

Section: 4 H

22L-6552 4H

Q1) Minimum Sum Partition Problem

ind \ Target	0	1	2	3 - - - -	Total Sum
0	[redacted]				
1	True				
2	True				
3	True				
⋮	True				
N	True				

Array index starts from 1

↳ An empty subset will give a sum of zero

```
int findMinDiff( Arr[], N)
{
    TotalSum = 0

    for ( i = 1; i <= N; i++) // calculate sum of all elements
    {
        TotalSum += Arr[i]
    }

    Bool R[0 ---- N][0 ---- TotalSum]
    ↳ Boolean 2D array
    Initialize the array with 0s/false
    for (i = 1; i <= N; i++) // Base Case
    {                        Sum 0 is possible with an empty subset
        R[i][0] = true
    }
```

```
if ( Arr[1] <= TotalSum) // Base Case: If
{                           array has only
    R[1][Arr[1]] = true     one element
}                           then it can
                            provide sum
                            equal to
                            itself.

for(i=2 ; i <= N ; i++)
{
    for( sum=1 ; sum<=TotalSum ; sum++)
    {
        exclude = R[i-1][sum]

        include = false

        if(Arr[i] <= sum)
        {
            include = R[i-1][sum- Arr[i]]
        }

        R[i][sum] = exclude || include

    }

}
```

22L-6552 4H

```
minDiff = INT_MAX
for(S1=0; S1<=TotalSum; S1++)
{
    if(R[N][S1] == true)
    {
        S2 = TotalSum - S1
        tempDiff = ABS(S1-S2)
        minDiff = MIN(minDiff, tempDiff)
    }
}
Delete temporary array R
return minDiff
}
```

Time Complexity : O(N* TotalSum)

Q2) Part A: Longest Increasing Sub-sequence

```
int LIS( Arr[], N)
{
    R[1 ---------- N] = { 1 }  // initialize the array with 1s
    maxLIS = 1  // minimum possible sub-sequence can be 1 element

    for (i=1; i<=N; i++)  → Consider the element only if it makes an increasing sub-sequence.
    {
        for (j=1; j<i; j++)
        {
            if (Arr[j] < Arr[i])
            {
// update LIS on index i (ending)   R[i] = max(R[i], R[j] + 1)
// keep track of MaxLIS so far      maxLIS = max(maxLIS, R[i])
            }
        }
    }
    Delete temporary array R
    return maxLIS   // Time Complexity: O(N^2)
}
```

22L-6552 4H

Q3) Palindrome Partitioning

```
bool isPalindrome(i, j, str[])
{
    while(i<j)
    {
        if(str[i] != str[j])
        {
            return false
        }
        i++
        j--
    }

    return true
}
```

```
int palindromePartitioning(str[], N)
{
    R[1 ---------- N+1]  // each index of the array will store the minimum number of partitions ending at that index starting from right
    R[N+1] = 0 // Base Case

    for(i=N; i>=1 ; i--)
    {
        minCost = INT_MAX
        for(j=i; j<=N; j++)
        {
            if(isPalindrome(i, j, str))
            {
                Cost = 1 + R[j+1]
                minCost = min(minCost, cost)
            }
        }
        R[i] = minCost
    }

    return R[1] - 1 // subtract 1 to discard one extra cut
}
```

22L-6552 (4) 4H

Q4) Target Sum Problem

```
int targetSum(Arr[], N, target)
{
    totalSum = 0

    for(i=1; i<=N; i++)
    {
        totalSum += Arr[i]
    }

    if((totalSum-target)<0 || (totalSum-target)%2!=0)
    {
        return 0
    }
    else
    {
        return findWays(Arr, N, (totalSum-target)/2)
    }
}
```

$S1 = totalSum - S2$

$S1 - S2 = target$

$\Rightarrow totalSum - S2 - S2 = target$

$totalSum - 2S2 = target$

$S2 = (totalSum - target)/2$ → We have to find this.

```
int findWays (Arr[], N, target)
{
    R[0 ------ N][0 ------ target] = {0}
    if(Arr[1] == 0)
    {
        R[1][0] = 2 // 2 cases - pick and not pick
    }
    else
    {
        R[1][0] = 1 // 1 case - not pick
    }
    if( Arr[1] != 0 && Arr[1] <= target)
    {
        R[1][Arr[1]] = 1 // 1 case - pick
    }
```

22L-6552 4H

```
for(i=2; i<=N; i++)
{
    for(j=0; j<=target; j++)
    {
        notTaken = R[i-1][j]
        taken = 0
        if(Arr[i]<=j)
        {
            taken = R[i-1][j-Arr[i]]
        }
        R[i][j] = notTaken + taken
    }
}
return R[N][target] // return the number of subsets whose sum is equal to the target value
}
```

(Q5) Part A: Number of Different Optimal Solutions for the Rod Cutting Problem.

```
int RodCut(N, Pr[])
{
    R[0 ---- N]

    R[0] = 0

    for(j = 1; j <= N; j++)
    {
        q = 0

        for(i = 1; i <= j; i++)
        {
            q = max(q, R[j-i] + Pr[i])
        }

        R[j] = q
    }
```

22L-6552 4H

→ Every index contains the maximum profit for the rod length equal to that index.

C[1 ---------- N] = {0} // initialize with zeros.

→ Each index will hold optimal number of solutions for the rod length equal to that index.

C[1] = 1 // Base Case: Only one way to cut a rod whose length is 1

```
for (j=2 ; j<= N; j++)
{
    for (i=1 ; i<= j; i++)
    {
        profit = R[j-i] + Pr[i]
        if (profit == R[j])
        {
            C[j] += C[j-i]
        }
    }
}
return C[N] // return the optimal number of solutions
}
```