CSE 321 ALGORITHM HW5

YAKUP TALHA YOLCU 1801042609

1. cluster

```python
def sum_rest(stations2,start_index2,sums2,sums_counter2):
    if(start_index2==len(stations2)):
        return sums2,sums_counter2
    else:
        sums2.append(sums2[sums_counter2-1]+stations2[start_index2])
        sums_counter2+=1
        return sum_rest(stations2,start_index2+1,sums2,sums_counter2)

def find_cluster_rec(stations,start_index,sums,sums_counter):
    if (start_index==len(stations)):
        return sums
    else:
        sums.append(stations[start_index])
        sums_counter+=1
        sums,sums_counter=sum_rest(stations,start_index+1,sums,sums_counter)
        return find_cluster_rec(stations,start_index+1,sums,sums_counter)



def find_cluster2(stations):
    sums=[]
    sums_counter=0
    sums=find_cluster_rec(stations,0,sums,sums_counter)
    return max(sums)
```

In this method I start from start index and go until the end of the array. I sum the value with the previous value of the sum array and I add the current value from stations array. I increment the counter and I go until the end of the station array. This method takes theta(n) time. I will have recurrence relation at analysis part.

In this method I start from start index and go until the end of the array. Firstly I put the current integer value to the sum array. Then I increment the counter of the sum array. Then I start from one more index from the current index. It is explained in sum_rest method. Because I go until the end of array, I call this method n times. But in this method, sum_rest method takes theta(n) time, so overall complexity is theta(n^2)

find_cluster_rec => base case if i==n

append => $\Theta(1)$ to the end

sum_rest => to be calculated

return find_cluster (___ s+1) => $T(n) = T(n-1) + ?$

sum_rest => base case => if k==n

append to the end => $\Theta(1)$

return sum_rest => (_ s+1) => $T(n) = T(n-1) + 1$

Recurrence relation of the sum_rest

$T(n) = T(n-1) + 1$

$T(n-1) = T(n-2) + 1$

$T(n-2) = T(n-3) + 1$

$T(n) = T(n-2) + 2$

$T(n) = T(n-3) + 3$

$T(n) = T(n-k) + k$

$T(n) = T(1) + k$

$T(1) = 1 + n - 1$

$T(1) = 1$

$n - k = 1$ $n = k + 1$

$T(n) = T(n-k) + n$ $k = n - 1$

$T(0) = T(1) = 1$

$T(1) = 1$ => $T(n) = T(1) + k$

$T(1) = T(1) + n - 1$

Induction ① $T(1) = 1$ ✓

② $T(n-1) = T(1) + n - 2$ Assume true

③ try for n $T(n) = T(1) + n - 1$ =? $T(n) = T(n-1) + 1$

$1 + n - 1 \stackrel{?}{=} 1 + n - 2 + 1$

$n = n$ ✓ Proved

This was for sum_rest. We have same recurrence relation for find_cluster_rec. But instead constant we have linear as T(n) because of sum_rest

$T(n) = T(n-1) + n$

$T(n-1) = T(n-2) + n - 1$

$T(n-2) = T(n-3) + n - 2$

$n - k = 1$ $n = k + 1$ $k = n - 1$

$T(n) = T(n-3) + n - 2 + n - 1 + n$

$T(n) = T(n-k) + \frac{n(n-1)}{2}$ $T(n) = T(1) + \frac{n(n-1)}{2}$

$T(n) = T(1) + \frac{n(n-1)}{2}$

① Try for n=1; $T(1) = T(1) + \frac{n(n-1)}{2}$

$T(1) = T(1) + 0$ ✓

② Assume true for n-1 $T(n-1) = T(1) + \frac{(n-1)(n-2)}{2}$

③ Try for n $T(n) = T(1) + \frac{n(n-1)}{2} \stackrel{?}{=} T(n-1) + n$

$T(1) + \frac{n(n-1)}{2} \stackrel{?}{=} T(1) + \frac{(n-1)(n-2)}{2} + n$

$\cancel{1} + \frac{n^2-n}{2} \stackrel{?}{=} \cancel{1} + \frac{n^2-3n+2}{2} + n$

$\frac{n^2-n}{2} = \frac{n^2-n}{2}$ ✓ true for n

So $T(n) = T(1) + \frac{n(n-1)}{2}$

$T(n) = 1 + \frac{n(n-1)}{2}$

$T(n) = n^2 \in \Theta(n^2)$

Firstly I take the stations as an argument. I call the find cluster2 method. Then I call other recursive methods.

Test outputs:

```
[3, -5, 2, 11, -8, 9, -5]
MAX OF PROFIT:  14
------------
[3, 5, 2, 11, -8, 9, -5]
MAX OF PROFIT:  22
------------
```

At hw3, I had this algorithm =>

Its complexity was theta (n^2)

My new algorithm has also theta(n^2). So nothing is changed. At both algorithm, I iterate through n^2 times

```python
def most_profitable(stations):
    left_index=0
    right_index=0
    profit_sum=-999
    current_max=0

    for i in range(len(stations)):
        current_max=stations[i]
        for j in range(i+1,len(stations)):
            current_max+=stations[j]
            if(profit_sum<current_max):
                profit_sum=current_max
                left_index=i
                right_index=j

    return stations[left_index:right_index+1],left_index,right_index
```

2. candy

```python
def getmaxobtainable(price,length):
    values = []
    for i in range(length+1):
        values.append(0)

    values=getmaxobtainable_rec(price,length,1,INT_MIN_VALUE,values)
    return values[length]

def determine_max(price,start_index,end_index,max_val,values):
    if(start_index==end_index):
        return max_val
    else:
        max_val = max(max_val, price[start_index] + values[end_index - start_index - 1])
        return determine_max(price,start_index+1,end_index,max_val,values)

def getmaxobtainable_rec(price,length,start_index,max_val,values):
    if(start_index==length+1):
        return values
    else:
        max_val=INT_MIN_VALUE
        max_val=determine_max(price,0,start_index,max_val,values)
        values[start_index]=max_val
        return getmaxobtainable_rec(price,length,start_index+1,max_val,values)
```

Firstly I take the prices of the candies. Then I fill the hold array with zeros because at each iteration I will have a maximum value. In recursive getmaxobtainable method I take price, start index, maxvalue, and values. If we reached the end of the array, we return the current hold array. Otherwise at each iteration of the second recursive function(determine max) we partitonate the value. For instance, if price[start] index is 4, We start from 1 and 3 then continue 2 and 2. We are doing the same operation to the all of the values of the price array.

getmax oblowable => filling with 0 => $\sum_{i=0}^{n} 1 = n$

get_max_obtaiable_rec=>

determine_max => Goes until end of array.

$\sum_{i=0}^{n} = n$ find-max op=> constant time

Determine max is called. It has $\Theta(n)$ complexity

This function goes until end of array. So

overall comp is $\sum_{i=0}^{n} n = n^2 \in \Theta(n^2)$

T(n)=T(n-1)+theta(n)

It is same as first question. Complexity becomes theta(n^2)

Test Outputs:

```
[1, 5, 8, 9, 10, 17, 17, 20]
Maximum Obtainable Value is  22
[1, 5, 8, 9, 10]
Maximum Obtainable Value is  13
```

3. cheese

```python
def getMaxValue(weights, prices, capacity):
        values = []
        for i in range(len(weights)):
            values.append([prices[i]//weights[i],weights[i],prices[i]])

        values.sort(reverse=True)

        totalValue = 0
        for i in values:
            currentWeight = int(i[1])
            currentPrice = int(i[2])
            if capacity - currentWeight >= 0:
                capacity -= currentWeight
                totalValue += currentPrice
            else:
                fraction = capacity / currentWeight
                totalValue += currentPrice * fraction
                capacity = int(capacity - (currentWeight * fraction))
                if(capacity<=0):
                    break
        return totalValue
```

At this function, I take weights array, prices array and capacity of the box. Then I create an array that holds the price/weight ratio of the cheese, weight of the cheese, price of the cheese. Then I sort the entire list depend on the price/weight ratio. Because the one that have most price/weight ratio should be put firstly. In the for loop, I take the current weight and current price of the cheese. Then I check the capacity if full or not. If we have enough capacity, then we put the entire cheese. If there is not enough capacity, we put a portion of the the cheese. We take fraction of the remained capacity. Then we put the cheese portion. At each put, we add price to get entire price of the box

Second for loop iterates through n element so worst case is theta(n). But before this we have sorting algorithm which has larger and O(nlogn) complexity, Overall complexity is O(nlogn)

Test Outputs:

$\sum_{i=0} 1 = n \Rightarrow$ appending operation $\Theta(n)$

Sorting op $\Rightarrow$ merge sort $\Rightarrow$ nlogn $\Theta(n\log n)$

worst case $\sum_{i=0}^{n} 1 = n$ if capacity runs out it could have less complexity

for loop But capacity is constant value. We don't need to take into account.

Overall complexity is $\Theta(n\log n)$

```
[10, 40, 20, 30]
[60, 40, 100, 120]
Maximum value = 240.0
[10, 40, 20, 30]
[60, 40, 100, 120]
Maximum value = 230.0
```

4. Courses

```python
def maxcourse(all_courses,n):
    max_counter=1
    cur_course=0
    for i in range(n):
        if(all_courses[cur_course][1]<=all_courses[i][0]):
            cur_course=i
            max_counter+=1
    return max_counter

start_time = [1, 3, 0, 5, 8, 5]
finish_time = [2, 4, 6, 7, 9, 9]
courses=[]
for i in range(len(start_time)):
    courses.append([start_time[i],finish_time[i]])
print(courses)
print(start_time)
print(finish_time)
print("Max number of course:",maxcourse(courses,len(start_time)))
print("------------")
```

I start from first course and look for the next course is overlapping or not, then I take it and incremented the counter

Test Outputs:

```
[[1, 2], [3, 4], [0, 6], [5, 7], [8, 9], [5, 9]]
[1, 3, 0, 5, 8, 5]
[2, 4, 6, 7, 9, 9]
Max number of course: 4
------------
```

$$\sum_{i=0}^{n-1} 1 = n \Rightarrow T(n) = \Theta(n)$$