1. Scheduling with weights

a) Not optimal. Example:
If there are three emails to write are 1,2, and 3 with $t_1=1$, $t_2=2$, $t_3=3$ and $w_1=1$, $w_2=2$, $w_3=9$. With the greedy algorithm by smallest time t1 first. Then $C_1=1$, $C_2=3$, $C_3=6$. $C_1w_1+C_2w_2+C_3w_3=1+3+54=58$. But if starts with t_3, we have $C_1'=3$, $C_2'=5$, $C_3'=6$, $C_1'w_3+C_2'w_2+C_3'w_1=9+10+6=25<58$.

b) Not optimal. Example:
If there are three emails to write are 1,2, and 3 with $t_1=10$, $t_2=2$, $t_3=1$ and $w_1=15$, $w_2=14$, $w_3=13$. With the greedy algorithm by largest weight w1 first. Then $C_1=10$, $C_2=12$, $C_3=13$. $C_1w_1+C_2w_2+C_3w_3=150+168+169=487$. But if starts with w_3, we have $C_1'=_1$, $C_2'=3$, $C_3'=13$, $C_1'w_3+C_2'w_2+C_3'w_1=250<487$.

c) Optimal. Proof:

(1) All placement with no inversions have the same total value.
If two different solutions have neither inversions nor idle time, then they might not write emails exactly in the same order, but they can only differ in the order in which emails with the same ratio. Consider such a ratio r. In both solutions, the emails with ratio r are all wrote back consecutively (after all emails with smaller ratios and before all emails with larger ratios). Among the emails with ratio r, the last one makes the maximum sum of w_iC_i, and this value does not depend on the order of the emails.

(2) There is an optimal solution that has no inversions.

(a) If O has an inversion, then there is a pair of emails i and j such that i is placed immediately before i and has ratio(i)<ratio(j).
Indeed, consider an inversion in which an email a is placed sometime before an email b, and ratio(a)<ratio(b). If we advanced in the placement order of emails from a to b one at a time, there has to come a point at which the total sum we see decreases for the first time. This corresponds to a pair of consecutive emails that form an inversion.

(b) After swapping i and j we get a solution with one less inversion.

(c) The new swapped solution has a total sum of w_iC_i no larger than that of O.
Assume all emails before email I take time C. Before swapping, $C_iw_i+C_jw_j = (C+t_i)*w_i+(C+t_i+t_j)*w_j = C(w_i+w_j)+w_it_i+w_jt_j+t_iw_j$. After swapping, $C_iw_i+C_jw_j = (C+t_j)*w_j+(C+t_i+t_j)*w_i = C(w_i+w_j)+w_it_i+w_jt_j+t_jw_i$. For $w_i/t_i<w_j/t_j$, we have $w_it_j<w_jt_i$. Hence $C_iw_i+C_jw_j$ becomes smaller after swapping.

(3) The solution A produced by the greedy algorithm has optimal maximum value.
Statement (2) proves that an optimal solution with no intervals exists. Statement (1) proves that all solutions with no intervals have the same maximum value, and so the solution obtained by the greedy algorithm is optimal.

2. Divide and Conquer

a) Two indices: 4 and 7
Corresponding optimal sum:32

b)

1. Assume each integer in the table has an index. Indexes from left to right are 1 to n. Divide the table into two parts at the integer[(n+1)/2].

2. If the subarray of maximum sum is entirely in the left part: Calculate all the sums of subarrays recursive calling this function in the left part (from integer [1] to integer[(n+1)/2]) of table. Set the maximum sum SumLeftMax.

3. If the subarray of maximum sum is entirely in the right part: Calculate all the sums of subarrays recursive calling this function in the right part (from integer[(n+1)/2] to integer[n]) of table. Set the maximum sum SumRightMax.

4. If the subarray is across left and right parts: Find the maximum sum SumL of the subarray which starts from integer[(n+1)/2] to left part of the table. Find the maximum sum SumR of the subarray which starts from integer[(n+1)/2] to right part of the table. The maximum sum of subarray across left and right parts is SumL+SumR.

5. Get the maximum value in SumLeftMax, SumRightMax and (SumL+SumR). This is the subarray of maximum sum.

3 Master Theorem

a) a=49, b=7, d=2. $\log_b a$=2=d. T(n)= $\Theta(n^2 \log n)$

b) a=1/4. a<1, there cannot be less than one subproblem. So master theorem is not applicable.

c) F(n), which is the combination time, cannot be negative. T(n) is not monotone. Master theorem is not applicable.

d) a=2, b=4, d=0.6. $\log_b a$=0.5<d. T(n)= $\Theta(n^{0.6})$

e) a=3, b=2, d=0. $\log_b a$>d. T(n)= $\Theta(n^{\log_2 3})$

4 Dynamic Programming

a) Prove the problem has optimal substructure:

MaxPlausibility(y) = 0 if k = 0

For $0 \leq j \leq k$:

MaxPlausibility ($y_1 \ldots, y_j$)

$= \max_{1 \leq n \leq j}$ {MaxPlausibility(n-1) + plausibility ($y_n \ldots, y_j$)}

We use the optimal solution to find max plausibility in each prefix and the rest part of the original part will be compute max plausibility recursively. Then we can get the max plausibility of the whole original string with optimal solution. Thus, this problem can be divided into several overlapping subproblems and we will end up solving the same subproblem over and over again.

b) Base case: MaxPlausibility(y) = 0 if k = 0.

Recursive expression:

For $0 \leq j \leq k$:

MaxPlausibility (j)

$= \max_{1 \le n \le j}$ {MaxPlausibility(n-1) + plausibility (y_n…, y_j)}

Goal: Find the maximum total plausibility of the segmentation of a string.

c) Recursive algorithm with memorization:

MaxPlausibility (0) = 0
For j from 1 to k
  M[j] = empty
Compute MaxPlausibility(k)

MaxPlausibility(j) {
If M[j] = empty
  M[j] = $\max_{1 \le n \le j}$ {MaxPlausibility(n-1) + plausibility (y_n…, y_j)}
Return M[j]
}

Time complexity: $O(k^2)$. There are k loops when computing max plausibility of each prefix. In each prefix there are j recursive calls.
Space complexity: O(k). Create a list which contains k elements to store the results of all the prefix of the original string.

d) Bottom Up:

```
MaxPlausibility (0) = 0
For j from 1 to k {
  value = - ∞
  For n from 1 to j {
      If value < MaxPlausibility(n-1) + plausibility (yn…, yj)
         value = MaxPlausibility(n-1) + plausibility (yn…, yj)
      }
  MaxPlausibility(j) = value
 }
```