1.

(a)

	Day 1	Day 2	Day 3	Day 4	Day 5
x	20	11	15	12	16
s	10	4	3	2	1

Rebooting on day 2 and day 4 is the optimal solution in that we would process a total of 30 terabytes (10 + 0 + 10 + 0 + 10).

(b)

OPT(i, j) is the max terabytes processed for the subset {x_1, …, x_i} j days after the reboot.

Algorithm:
Initialize a 2D Array that is OPT[x][x]
OPT[x][j] = min(x_n, s_j) for j = 1, …, n
For i = n-1, …, 1
    For j = 1, …, i
        Set OPT[i][j] to max(OPT(1+i, 1), min(x_i, s_j) + OPT(1+i, 1+j))
Return OPT[1][1] //final solution

3 Different Cases:
OPT(i, j) = {Last day(i = n): min(x_n, s_j)
    Reboot Days: OPT(i+1, 1)
    Other days: OPT(i+1, j+1) + min(x_i, s_j)}

Time Complexity:
Going through and filling in the OPT matrix is done in O(n^2). The functions for max and min are all constant time (O(1)), so the total time complexity is O(n^2).

2.

OPT(i, j) = minimum insertions to create a palindrome from index i to j in a string

Algorithm:
Initialize 2D-array OPT[i][j] = 0 for all i and j from 1 to n
For x = 1, … , n
For j = x, … , n
    Set i = j - x
    If String[i] is the same as string[j]
        Set OPT[i][j] to OPT[i+1][j-1]
    Else
        Set OPT[i][j] to min(OPT(i-1, j) + 1, OPT(i, j-1)+1)
Return OPT[1][n] //final solution

Cases:
OPT(i, j) = {No insertion: OPT(i+1, j-1)
        Insertion: min(OPT(i-1, j)+1, OPT(i, j-1)+1)

Time Complexity:
Going through and filling in the OPT matrix is done in O(n^2). All the other functions are all constant time (O(1)), so the total time complexity is O(n^2).

3.

Idea:
The goal is to get the minimum wasted space for any division. Between cutting vertically vs horizontally from each of the dimensions of the rectangle, we are going to split the triangle into different subproblems. We store a matrix that contains the least amount of waste in order to keep track of the minimum amount. And this matrix is for a rectangle of the current size as we build up to the complete rectangle.

Algorithm:
Initialize Matrix A[L][W] = 0
For c = 0, … , L
    For d = 0, … , W
        If ∃ no rectangle where a = c, b = d
            A[c][d] = xy
Waste(x, y) = minimum(minimum(for all d < y) (Waste(d, y)+Waste(x-d, y)), minimum (for all c < x (Waste(c, y) + Waste(x-c, y)), Waste(x,y))

Comments:
The vertical axis is x and the horizontal axis is y. In the case that there exists no rectangle such that a = c and b = d in it, then that would be the worst case scenario (A[c][d] = xy)

4.

With the Hitting Set Problem, we want to try and find a set H with max size k with an element in each subset B_i. For the Vertex Cover Problem, it finds a set S with >= 1 vertex in each e in E. So, elements in H are similar to vertices in S and a subset B_i will be an edge e in E. Therefore, if we can solve the Hitting Set Problem, we can also solve the Vertex Cover Problem for a max size k. And finally, we can say that the Vertex Cover Problem is able to be polynomially time reduced to the Hitting Set Problem and the vertex cover problem $\leq_p$ the hitting set problem