There are 2 questions for a total of 10 points.

1. (3 points) Recall the Longest Increasing Subsequence problem discussed in class. As in the lecture, for the following sequence of numbers in array $A[1...16] = [14, 8, 2, 7, 4, 10, 6, 0, 1, 9, 5, 13, 11, 3, 15, 12]$, let $L[i]$ denote the length of the longest increasing subsequence that ends with $A[i]$. Give the values of $L[1], L[2], ..., L[16]$.

Solution: 1, 1, 1, 2, 2, 3, 3, 1, 2, 4, 3, 5, 5, 3, 6, 6

2. (7 points) You are a manager at a shipping company. On a particular day you have to ship nnn boxes with weights $w[1...n]$ that are positive integers between 1 and 100. You have four trucks, each with weight limit D that is a positive integer. Design an algorithm to determine if it is possible to pack all the n boxes into the four trucks such that the for every truck, the total weight of boxes in the truck is at most D (i.e., weight limit is not exceeded). Your algorithm should output "yes" if it is possible to pack and "no" otherwise. Argue correctness and giving running time analysis.

Solution:

Subproblems: $P(i, j, k, l)$ = problem which asks if there is an assignment of boxes $1..i$ into four trucks where the total weights on the 1st, 2nd, 3rd trucks are j, k and l respectively (the weight on the last truck equals (sum of weights on boxes $1..i$) - $(j + k + l)$), where j, k, l are restricted to the domain where each truck has boxes on it weighing at most D.

$S(i, j, k, l)$ is the answer to the problem $P(i, j, k, l)$ (either True or False).

Base Case: The base case is when $i = 0$, i.e., the case when we assign no box to any truck. In that case, $S(0, 0, 0, 0) = True$, and for all other j, k, l, we have $S(0, j, k, l) = False$.

Recursive step:

Consider the problem $P(i, j, k, l)$.

Either we assign the box #i to the truck $1, 2, 3$ or 4. In any case, we can assign a truck if that truck doesn't overflow. In the case of truck 1, it is possible only if $j \geq w[i]$, and the total weights previously must have been $j - w[i]$, k, l respectively, and similarly for the other trucks. For truck 4, the weights must have been j, k, l on the first three trucks.

Hence we have:

$$S(i, j, k, l) = S'(i-1, j-w[i], k, l) \text{ OR } S'(i-1, j, k-w[i], l) \text{OR } S'(i-1, j, k, l-w[i]) \text{ OR } S'(i-1, j, k, l)$$

where $S'(i, j, k, l)$ is defined as $S(i, j, k, l)$ if $0 \leq j, k, l$, (sum of $w[1..i]$ - $(j + k + l)$) $\leq D$ and False otherwise.

Note that this takes care of overflow as well as that of negative total weight in a subproblem (which can arise due to subtraction, as in cases like where $j < w[i]$). It is easy to see how the assignment can be computed from these values, and hence this condition is necessary and sufficient.

Order in which problems are solved: We solve the problems in increasing order of i, then j, then k, then l (first we compare i, then we compare j if there is a tie, and so on). This is consistent, since for each subproblem we need, the expression on the right (in the recursive step) is computed before the current subproblem in this ordering (since at least one of i, j, k, l decreases by at least 1).

Output (answer to the final problem): If D is at least $100n$, then we can put all our boxes in the same truck, and hence the answer is true. Otherwise, the answer to the given problem will be the logical OR of all $S(n, j, k, l)$ where $j, k, l \leq D$ and $0 \leq$ (sum of weights of boxes $1..n) - (j+k+l) \leq D$.

Pseudocode:

```
function PossiblePacking($w[1..n], D$)
    if $D \geq 100n$ then
        return True
    end if
    let $s :=$ boolean array of dimension $(n+1) \times D \times D \times D$ initialized to all False
    $s[0,0,0,0] := True$
    $sum := 0$
    for $i \in \{1..n\}$ do
        $sum += w[i]$                                         ▷ $sum$ is now sum of $w[1..i]$
        for $j \in \{0..D\}$ do
            for $k \in \{0..D\}$ do
                for $l \in \{0..D\}$ do
                    if $j \geq w[i]$ then
                        $s[i,j,k,l] = s[i,j,k,l] \vee s[i-1, j-w[i], k, l]$
                    end if
                    if $k \geq w[i]$ then
                        $s[i,j,k,l] = s[i,j,k,l] \vee s[i-1, j, k-w[i], l]$
                    end if
                    if $l \geq w[i]$ then
                        $s[i,j,k,l] = s[i,j,k,l] \vee s[i-1, j, k, l-w[i]]$
                    end if
                    if $D \geq sum - j - k - l \geq w[i]$ then
                        $s[i,j,k,l] = s[i,j,k,l] \vee s[i-1, j, k, l]$
                    end if
                end for
            end for
        end for
    end for
    let $answer :=$ False                                    ▷ $sum$ is now sum of $w[1..n]$
    for $j \in \{0..D\}$ do
        for $k \in \{0..D\}$ do
            for $l \in \{0..D\}$ do
                if $0 \leq sum - j - k - l \leq D$ then
                    $answer = answer \vee s[n,j,k,l]$
                end if
            end for
        end for
    end for
    return $answer$
end function
```

Runtime analysis: Note that if $D \geq 100n$, then the algorithm takes $O(1)$ time. Now suppose this is not the case. Then the declaration of the boolean array takes $O(nD^3)$ time. Then in the loop over l, the time taken per iteration is $O(1)$, assuming memory access, branch statements, comparison, and boolean and arithmetic operators take $O(1)$ time. Then since there are D such iterations, the loop takes $O(D)$ time. So the loop over k takes $O(D^2)$ time, and the loop over j takes $O(D^3)$ time. The loop over i thus takes $O(nD^3)$ time, since adding $w[i]$ to sum takes $O(1)$ time and we do $O(D^3)$ work apart from this in each iteration. Now for the final answer computation, we have 3 nested loops, and by a similar argument as above, it takes $O(D^3)$ time for that.

Hence overall the algorithm runs in $O(nD^3)$ time if $D < 100n$ and $O(1)$ otherwise. The first case can be simplified to $O(n^4)$ as well (but that isn't the tightest bound we can get on the time complexity).