(a) 设 J 是最优子集。根据定义，J 中的所有作业都可以按时完成。现在考虑调度问题以最小化最大延迟，但只考虑 J 中的作业。我们知道，根据 J 的定义，最小延迟为 0 (即，所有作业都可以按时调度)，并且在课堂上我们已经证明了按截止日期顺序调度作业的贪婪算法对于最小化最大延迟是最佳的。因此，按截止日期顺序调度 J 中的作业会为这组作业生成一个可行的调度。

(b) 这个问题与子集和问题相似。我们将有与该问题类似的子问题。第一个想法是考虑使用作业子集 $\{1,\ldots,m\}$ 的子问题。我们将像往常一样按截止日期递增的顺序排序作业，并且我们假设它们是这样编号的，即 $d_1 \leq \cdots \leq d_n = D$。为了解决原始问题，我们考虑两种情况：要么作业 n 被接受，要么被拒绝。如果作业 n 被拒绝，则问题简化为只使用前 $n-1$ 个作业的子问题。现在考虑作业 n 被接受的情况。根据 (a) 部分我们知道作业 n 应该最后调度。为了确保机器能在作业 n 的截止日期 D 前完成作业 n，所有其他被调度接受的作业应该在时间 $D - t_n$ 之前完成。我们将定义子问题，使这个问题是我们的子问题之一。

对于时间 $0 \leq d \leq D$ 和 $m = 0,\ldots,n$，令 $OPT(d,m)$ 表示在集合 $\{1,\ldots,m\}$ 中可以满足截止日期 d 的请求的最大子集。我们在这个子问题中意味着机器在时间 d 之后不再可用，因此所有请求要么必须在截止日期 d 之前完成，要么被拒绝 (即使作业 i 的截止日期 $d_i > d$)。现在我们有以下陈述。

(1)

- 如果作业 m 不在最优解 $OPT(d,m)$ 中，则 $OPT(d,m) = OPT(m-1,d)$。
- 如果作业 m 在最优解 $OPT(d,m)$ 中，则 $OPT(d,m) = OPT(m-1,d-t_m) + 1$。

这暗示了构建子问题值的方法如下。

Algorithm 1 Select-Jobs(n,D)

Array $M[0\ldots n, 0\ldots D]$
Array $S[0\ldots n, 0\ldots D]$
for $d = 0,\ldots,D$ **do**
    $M[0,d] = 0$
    $S[0,d] = \phi$
end for
for $m = 1,\ldots,n$ **do**
    for $d = 0,\ldots,D$ **do**
        if $M[m-1,d] \geq M[m-1,d-t_m] + 1$ **then**
            $M[m,d] = M[m-1,d]$
            $S[m,d] = S[m-1,d]$
        else
            $M[m,d] = M[m-1,d-t_m] + 1$
            $S[m,d] = S[m-1,d-t_m] \cup \{m\}$
        end if
    end for
end for
Return $M[n,D]$ and $S[n,D]$

正确性直接由陈述 (1) 得出。运行时间为 $O(n^2D)$ 也直接由问题中的 for 循环得出，有两个嵌套的 for 循环用于 m 和一个用于 d。这意味着循环的内部部分被调用 $O(nD)$ 次。这个 for 循环的

内部部分需要 $O(n)$ 时间，因为我们明确维护最优解。通过不维护 S 数组，并且仅在知道值之后恢复解决方案，运行时间可以改进到 $O(nD)$。