

- A. (a) Suppose we have an optimal solution which includes program P_i on the disk but not program P_j . If $s_i > s_j$ then we get another optimal solution by replacing P_i with P_j . Hence this greedy algorithm is correct.
- (b) Suppose we have three programs of size 7, 5, and 4 kilobytes, and the disk has size 10 kilobytes. We could use 9 kilobytes by storing the two smaller programs, but the proposed greedy algorithm would only use 7 kilobytes by storing the largest program on the disk. Hence this greedy algorithm fails in general.

- B. (a) Consider a log of length 10 with cuts at lengths 4, 5 and 6. The optimal strategy for cutting this log would be to first cut the log at 4, and then at 6, and finally at 5; the total cost would be $10 + 6 + 2 = 18$.

However, the proposed greedy strategy would cut the log at 5, and then at 4, and finally at 6. The total cost using this strategy would be $10 + 5 + 5 = 20$.

- (b) If $j = i+1$, then there are no cuts to be made, so the cost is 0.

If $j = i+2$, then there is one cut to be made, at a cost of $(\ell_{i+2} - \ell_i)$.

Assuming that $j > i+2$, the cost of the first cut is equal to the length of the sublog in question, namely $(\ell_j - \ell_i)$, regardless of where the cut is made. This is added to the (optimal) costs of cutting the two resulting sublogs. We wish therefore to minimize the sum of the costs of these two subproblems, considering all possible positions ℓ_k of the first cut.

Summarizing, we get the following recursive definition for $c[i, j]$.

$$c[i, i+1] = 0 \qquad c[i, i+2] = (\ell_{i+2} - \ell_i)$$

$$c[i, j] = (\ell_j - \ell_i) + \min_{i < k < j} (c[i, k] + c[k, j]) \quad (\text{for } j > i+2).$$

- (c) The elements of c are computed diagonal-by-diagonal.

LOGCUTTINGCOST($\ell[0..n+1]$)

```

1  for  $i \leftarrow 0$  to  $n$  do  $c[i, i+1] \leftarrow 0$ 
2  for  $i \leftarrow 0$  to  $n-1$  do  $c[i, i+2] \leftarrow (\ell[i+2] - \ell[i])$ 
3  for  $d \leftarrow 3$  to  $n+1$  do
4    for  $i \leftarrow 0$  to  $n+1-d$  do
5       $m \leftarrow c[i+1, i+d]$ 
6      for  $k \leftarrow i+2$  to  $i+d-1$  do
7        if  $m > c[i, k] + c[k, i+d]$  then  $m \leftarrow c[i, k] + c[k, i+d]$ 
8       $c[i, i+d] \leftarrow (\ell[i+d] - \ell[i]) + m$ 
9  return  $c[0, n+1]$ 

```

- (d) There are $O(n^2)$ values to compute, each of which requiring $O(n)$ time to compute. Thus the space required by this algorithm is $O(n^2)$, and the runtime of the algorithm is $O(n^3)$.