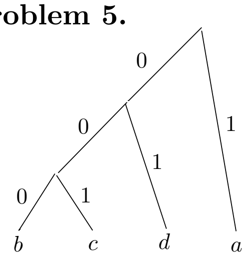
Problem 1. Perform k-selection to find the element e_1 with rank k_1. Perform k-selection again to find the element e_2 with rank k_2. The cost so far is $O(n)$. Then, scan S once again to report every element $e \in S$ between e_1 and e_2. This takes another $O(n)$ time because we only need to spend $O(1)$ on each $e \in S$.

Problem 2. Counterexample: $\mathcal{I} = \{[1,4],[4,5],[5,8]\}$. The algorithm returns only $\{[4,5]\}$ but the optimal solution is $\{[1,4],[5,8]\}$.

Problem 3. Identify any MST T of G. If e is an edge in T, we are done. Otherwise, T must contain a (unique) S-cross edge e'. Replacing e' with e gives another tree T'. As e has the minimum weight among all S-cross edges, the weight of T' cannot be higher than that of T. This means that T' must also be an MST.

Problem 4. $\{b,e\},\{b,c\},\{c,f\},\{c,d\},\{a,d\}$.

Problem 5.



Problem 6.

ℓ	1	2	3	4	5	6	7	8
$opt(\ell)$	3	6	9	12	15	18	21	24

Problem 7.

	s	1	2	3	4
t	0	0	0	0	0
1	0	0	1	1	1
2	0	0	1	2	2
3	0	0	1	2	3
4	0	0	1	2	3

Problem 8.

Lemma 1. *Let I_1 be the first interval selected by the algorithm. There must exist an optimal solution that contains I_1.*

Proof. Consider an arbitrary optimal solution S^*. Identify an arbitrary interval $I \in S^*$ that covers value 0. As I is at least as long as I_1, replacing I with I_1 gives another solution S with the same size as S^*. Therefore, S must be optimal. □

Lemma 2. *Let $I_1, I_2, ..., I_k$ be the first $k \geq 2$ intervals selected by the algorithm (in this order). If $\{I_1, ..., I_{k-1}\}$ exists in some optimal solution, then there must exist an optimal solution that contains all of $I_1, I_2, ..., I_k$.*

Proof. Consider an arbitrary optimal solution S^* that contains $I_1, ..., I_{k-1}$. Suppose that $I_{k-1} = [x, y]$. Thus, after adding I_{k-1} to S, Step 3 sets the value of a to $y + 1$.

Identify an arbitrary interval $I \in S^*$ that covers the value $a = y + 1$. As $I_k \cap [a, U]$ is at least as long as $I \cap [a, U]$, replacing I with I_k gives another solution S with the same size as S^*. Therefore, S must be optimal. $\square$

The algorithm's optimality follows from the above two lemmas.

Problem 9. For each $i \in [0, n]$, define $A[1 : i]$ as the prefix of A containing the first i elements. Given an integer $0 \in [1, n]$, define $opt(i)$ as the maximum sum that can be achieved by picking elements from $A[1 : i]$ under the stated constraint. Clearly, $opt(0) = 0$ and $opt(1) = A[1]$.

Lemma 3. *For $i \geq 2$, it holds that $opt(i) = \max\{opt(i-1), A[i] + opt(i-2)\}$.*

Proof. Consider the best strategy for picking elements from $A[1 : i]$.

- If the strategy does not choose $A[i]$, then the elements chosen also constitute an optimal solution for $A[1 : i-1]$. Hence, $opt(i) = opt(i-1)$.
- If the strategy chooses $A[i]$, the rest of the elements chosen must constitute an optimal solution for $A[1 : i-2]$ (notice that the strategy cannot pick $A[i-1]$ in this case). Hence, $opt(i) = A[i] + opt(i-2)$.

The lemma holds true because there are no other possibilities. $\square$

We can now compute $opt(i)$ in ascending order of i:

```
1.  opt(0) ← 0, opt(1) ← A[1]
2.  for i ← 2 to n
3.      if opt(i − 1) ≤ A[i] + opt(i − 2) then
4.          opt(i) ← A[i] + opt(i − 2)
        else
5.          opt(i) ← opt(i − 1)
```

It is clear that the running time is $O(n)$.