Subtask 1

In the first subtask, the limits are very low, and almost any brute force solution passes. One solution is as follows. Consider a fixed permutation of lanterns. Note that we can always buy the first one by definition.

Assume that we have already bought the first j lanterns. We can easily find the range of peaks that are reachable from the starting position using these lanterns. If all peaks are reachable, we are done and the answer for this starting point and this permutation of lanterns equals the total cost of the first j lanterns. Otherwise, check that the $j+1$-th lantern is reachable. If it isn't, or we have already bought all the lanterns, this particular permutation doesn't yield any solution. Otherwise, repeat the same process.

The complexity of this approach is $\mathcal{O}(k!(k+n))$.

Subtask 2

Let L be a set of purchased lanterns. The set of reachable altitudes is the union of all $[a_j, b_j]$ for all lanterns $j \in L$.

For the subtasks, we need the following simple, yet important observation:

There is always an optimal solution in which at any point in time the set of reachable altitudes forms a contiguous interval.

Indeed, this is clearly satisfied when the first lantern is purchased, and all purchases that would violate the constraint can be deferred to a later time. This is because we can never leave the interval of reachable altitudes that we're currently in.

This suggests that dynamic programming approaches might work for this task.

Indeed, we can define our state as

$$D[l][h][cur] = d$$

where l is the lowest reachable altitude, h is the highest reachable altitude, cur is the current peak, and d is the minumum cost needed to reach all the peaks from this situation.

Initially, $D[1][n][x] = 0$ for all x from 1 to n.

To compute all the DP transitions, simply note that we can find the range $[c, d]$ of all the reachable peaks trivially in time $\mathcal{O}(n)$, and then for each lantern j that satisfies the following conditions:

- it can be purchased in one of peaks $[c, d]$
- its operational range is not disjoint from $[l, h]$
- its operational range is not a subrange of $[l, h]$ (i.e., buying it enlarges the intervals of reachable altitudes)

we set

$$D[l][h][cur] = \min(D[l][h][cur], D[\min(a_j, l)][\max(b_j, h)] + c_j)$$

For a lantern j (if $a_j \leq h_{p_j} \leq b_j$), the answer is

$$D[a_j][b_j][p_j] + c_j$$

There are $\mathcal{O}(k^2 \cdot n)$ states, and each of them can be processed in $\mathcal{O}(\max(k, n))$ time. The total time complexity is $\mathcal{O}(k^2 \cdot n \cdot \max(k, n))$ or $\mathcal{O}(n^4)$ with the assumption $n \approx k$.

The solution can be implemented as a recursive function with memoisation. Alternatively, one can see that the states can be processed in the order of increasing l and decreasing h.

Subtask 3-4

To improve the above solution, observe that we don't need the value of *cur* exactly – it is sufficient to just know the range of peaks we currently are. Thus, instead of remembering the lowest and highest reachable altitudes, we remember $[id_l, id_h]$: the ID of the lanterns that achieve the lowest and highest reachable altitude, respectively. Clearly, $l = a_{id_l}$ and $h = b_{id_h}$.

What makes this solution more efficient than the previous one is that we can set $cur = p_{id_l}$. This reduces the number of states to $\mathcal{O}(k^2)$.

The initial conditions, the solution reconstruction and the transitions can be performed the same way. As a result, the total time complexity is reduced to $\mathcal{O}(k^2 \max(k, n)$, or $\mathcal{O}(n^3)$ under the assumption that $n \approx k$.

Subtask 5

The full solution optimises the above approach even further.

Assume that the extremal lanterns are i, j and we buy the k-th lantern. Assume that the k-th lantern is reachable, and changes the lowest reachable altitude (but not the highest). The act of buying the k-th lantern is equivalent to setting, for each applicable i, j:

$$D[i][j] = \min(D[i][j], D[k][j] + c_k)$$

Let us fix k and j. We can express the above assignments for all valid choices of i as follows

$$D[i][j] = \min(D[i][j], \min_{k \in F_j^+(i))} D[k][j] + c_k)\,,$$

where $F_j^+(i)$ is the set of all lanterns k that satisfy the following two conditions:

- $a_i \geq a_k$ (otherwise purchasing lantern k doesn't improve the range)
- Peak p_k is reachable when lanterns i and j are purchased.

The set $F_j(i)$ can have $\mathcal{O}(k)$ elements, so we cannot afford to compute it explicitly. Instead, we do the following.

We process the lanterns in decreasing order of a_i. This way, the first condition is satisfied automatically.

We store values of $D[i][j]+c_i$ in a segment tree that returns the minimum value in a range. In this segment tree, the lanterns are sorted by p_i. This way, the set of lanterns purchasable when i and j are the lanterns achieving the lowest and highest altitude, respectively, form an interval in this segment tree.

There are multiple ways of obtaining this interval. For a fixed pair of lanterns i, j, we look for the largest interval whose elevation is in the range $[a_i, b_j]$. We can find this interval using binary search on its endpoints, starting from p_i. In order to do that fast enough, we need to be able to answer range min and max queries on h. We can precompute these minimums and maximums in $\mathcal{O}(n^2)$ time.

To summarise:

- Precompute $lo_h(a, b) = \min_{i \in [a,b]} h(i)$ and $hi_h(a, b) = \max_{i \in [a,b]} h(i)$ for all $1 \leq a \leq b \leq n$ in $\mathcal{O}(n^2)$.
- Precompute $left(i, j)$, $right(i, j)$ – the range of reachable lanterns when lanterns i and j are purchased, using binary search and lo_h and hi_h.
- For a fixed j, store $D[i][j] + c_i$ in a segment tree, ordered by p_i.
- For each k, in order of increasing a_k, set $D[i][j] = \min(D[i][j], segtree.getmin(left(i, j), right(i, j)))$.

This handles the purchases of lanterns that decrease the lowest reachable altitude. We can create a similar set of segment trees for lanterns that increase the highest reachable altitude. Note that in this case, we have to process the lanterns in order of **decreasing** b_k.

Another complication is that there are also lanterns that decrease the lowest reachable altitude and increase the highest reachable altitude at the same time. We can use another segment tree to process these. The details of this last step are left as an exercise to the reader.

In total this solution uses $\mathcal{O}(\max(n,k)^2 \log \max(n,k))$ time.