Q 2.1)

They could group up employees then calculating profitability of the group without n^{th} employee when $n \in [1, k]$.

Then keep the group of employees with the highest profitability and choose to remove the one that is not included in the group.

By using this method, it would cover all possible profitability of any set of employees which always include the nephew and be able to choose the best set of employees and the one to fire to maximize the profitability.

Q 2.2)

Initializing an empty array to store a set of k employees that maximize profitability. By iterating through all bankers (since the given array is already been sorted in descending in term of performance), then putting first k bankers into the created array and calculate for profitability of the team. Kicking the least investments among them and picking next banker to check for new profitability then keep tracking of the team with highest profitability so far (replace with a new one whenever seeing a higher profitability).

Kick and pick a new banker until we reach the last element of the given array. Since the performance of the next banker from the array would always be used as the multiplier when calculating profitability (the next element's performance is at most the performance of the previous one), so that we wouldn't miss any possible greater profitability (because we kick a banker with the least investment which when grouping him with the next banker the profitability would always be less than when grouping the next banker with any other bankers with bigger investments because we would always use the next banker's performance as a multiplier).

Given $A > B$, $(A + C + D) > (B + C + D)$. In the end, we could use the set of k employees that we tracked as a result for the team with highest profitability.

Overall time complexity takes $O(nk)$ sine we iterate through all bankers (n) and we calculate profitability of team which we need to iterate through every member in the team (k) to sum them in term of investment and get the least performance among them (do this n^{th} times). Which means it is $O(n) * O(k) = O(nk)$.

Q 2.3)

Similar approach from the previous question but this time we would lessen time complexity when calculating for profitability from $O(k)$ into $O(\log k)$ with help using min heap.

By having min heap to store the investment of each banker in the team of k size, both inserting a new banker and deleting the lowest investment banker would take only $O(\log k)$. In order to calculate profitability, we then keep tracking of the current sum of investment of the team, whenever we take an element out from the heap, we subtract that amount of investment from the team's investment and add new banker's investment into it whenever we pick a new banker to the team. It would take constant time to adjust the sum of investment of the current team. And we would use the new upcoming banker's performance as a multiplier (as explained in previous question). Doing it this way takes $O(1)$ to calculate for profitability and $O(\log k)$ for inserting and deleting an element from the heap. So, it is $O(\log k) + O(1) = O(\log k)$.

Any other details as described in the previous question.

Overall time complexity takes $O(n \log k)$ since we are going through all bankers (n) but this time, by using min heap, this could take $O(\log k)$ to get profitability. So, $O(n) * O(\log k) = O(n \log k)$.