**Kick Start 2021 - Round G**

# Analysis: Banana Bunches

Formally the problem can be written as: given an array of size $\mathbf{N}$, select at most two non-overlapping subarrays, such that, sum of all the selected elements is equal to $\mathbf{K}$. We need to minimise the sum of lengths of these two subarrays. We can calculate the prefix sum of the given array in $O(\mathbf{N})$. $Pre_i$ denotes the sum of all the elements from $1$ to $i$. The sum of elements in a particular $subarray(i, j)$ is given by $Pre_j - Pre_{i-1}$ and can be calculated in $O(1)$ time.

One special case is when there is an element equal to $\mathbf{K}$ in the given array. In this case, we can return $1$ as the answer and can be done in a $O(\mathbf{N})$ traversal. For all the other cases, we can get the answer by selecting two non-overlapping subarrays.

### Test set 1

For this test set, we can consider all possible pair of subarrays as constraints on $\mathbf{N}$ are small. The total number of pair of subarrays are $O(\mathbf{N}^4)$. For each pair of non-overlapping subarrays, one can check whether the sum of elements in these subarrays is equal to $\mathbf{K}$. Formally, consider all $(i, j, x, y)$ such that sum of $subarray(i, j)$ and $subarray(x, y)$ is equal to $\mathbf{K}$ and $i \leq j < x \leq y$. Among all such pairs of subarrays, the optimal answer is the pair with the minimum sum of length of the subarrays. Formally, we need to minimise $j - i + 1 + y - x + 1$ over all such pairs. For each pair, one can calculate the sum and length of these subarrays in $O(1)$. Hence, the overall complexity of the solution is $O(\mathbf{N}^4)$.

### Test set 2

For this test set, we cannot consider all possible pair of subarrays as this solution would time out with the given constraints. We can iterate on all possible $(i, j, x)$ and find an optimal index $y$ such that sum of $subarray(i, j)$ and $subarray(x, y)$ is equal to $\mathbf{K}$ and $i \leq j < x \leq y$. For a particular $(i, j, x)$ we need to find smallest index $y$ such that $Pre_j - Pre_{i-1} + Pre_y - Prex - 1 = \mathbf{K}$. This can be done by performing a binary search on indexes from $x$ to $N$. If there is no such index $y$, we can ignore this triplet. Among all such possible $(i, j, x, y)$ quadruples, we need to minimise $j - i + 1 + y - x + 1$. We are able to find an optimal index $l$ in $O(\log \mathbf{N})$ time complexity for a particular triplet $(i, j, x)$. There are $O(\mathbf{N}^3)$ such triplets. Hence, the overall complexity of the solution is $O(\mathbf{N}^3 \log \mathbf{N})$.

We can further reduce the complexity by using two pointers instead of binary search. We can consider all possible $(j, x)$ and then use two pointer approach to find optimal $y$ for each $i$. For each pair $(j, x)$, we perform $O(\mathbf{N})$ operations. Hence, the overall complexity of the solution is $O(\mathbf{N}^3)$.

### Test set 3

For this test set, we can store the optimal second subarray length for each sum and iterate over each possible first subarray.

Consider the following pseudocode:

```
for i = 0 to K:
  Best[i] = inf
ans = inf
for i = N to 1:
  currSum = 0
  for j = i to 1:
    // |currSum| denotes sum of subarray(j, i).
    currSum += B[j]
    if currSum <= K:
      // Best[K - currSum] denotes the minimum length of subarray starting after index i which has sum equ
      ans = min(ans, j - i + 1 + Best[K - currSum])

  currPostSum = 0
  for x = i to N:
    // |currPostSum| denotes sum of subarray(i, x).
    currPostSum += B[x]
    if currPostSum <= K:
      // Update the minimum length of subarray with sum equal to |currPostSum|.
      Best[currPostSum] = min(Best[currPostSum], x - i + 1)
```

When we are iterating from $\mathbf{N}$ to $1$ and are at index $i$ currently, $Best_S$ stores the minimum length of subarray starting at any index greater than $i$ and having sum $S$. Basically, this would denote the optimal length of second subarray with sum $S$ if we choose first subarray ending at index $i$. We consider all indexes $j$ from $1$ to $i$. If sum of subarray $(j, i)$ is $currSum$ (such that $currSum \leq \mathbf{K}$), we update the answer. Then, we update the minimum length for each sum for the subarrays starting at index $i$. We do this by iterating for $x$ from $i$ to $\mathbf{N}$. If sum of $subarray(i, x)$ is $currPostSum$, then update $Best_{currPostSum} = min(Best_{currPostSum}, x - i + 1)$. As $K \leq 10^6$, we can maintain an array for storing $Best$. This would allow updating and looking up the sum in $O(1)$ time. For each index we do $O(\mathbf{N})$ operations, hence the overall complexity is $O(\mathbf{N}^2 + \mathbf{K})$.