Kick Start 2020 - Round E

Analysis: Longest Arithmetic

Consider a subarray of length K which is an arithmetic subarray, and let the elements of arithmetic subarray be B_1, B_2, B_3, ..., B_K. We can say that $B_2 - B_1 = B_{i+1} - B_i$ for $1 \le i < K$, because consecutive elements of arithmetic sequence should have a common difference.

Claim 1: In the given array, consider a subarray starting at index i and ending at index j. Now if this subarray is not arithmetic, there exists some index x such that $i \le x < j$ and $\mathbf{A}_{x+1} - \mathbf{A}_x \ne \mathbf{A}_{i+1} - \mathbf{A}_i$. All subarrays starting at index i and ending at index y such that $x < y \le \mathbf{N}$, are not arithmetic because all such subarrays would contain index x such that $\mathbf{A}_{x+1} - \mathbf{A}_x \ne \mathbf{A}_{i+1} - \mathbf{A}_i$.

Test Set 1

For each element i such that ($1 \le i < \mathbf{N}$), we consider each subarray starting at index i. Consider subarray(i,j) and start with j = i. Increment j while subarray(i,j) is an arithmetic subarray. For a fixed index i, we do not need to increment j after we find the first index such that subarray(i,j) is not an arithmetic subarray. None of the subarrays with i as a starting point and ending point after the index j will be arithmetic subarrays according to Claim 1. Let the maximum j for index i such that subarray(i,j) is an arithmetic subarray be max_j. We can conclude our approach as follows. Initialise the answer as 0. For each index i, find max_j. Update the answer if max_j - i + 1 is greater than the answer. For each index i, we would traverse O(**N**) elements. Hence, the overall complexity of the solution is $O(\mathbf{N}^2)$.

Sample Code (C++)

```
int maxArithmeticSubarray(vector<int> array) {
  int maxLen = 0;
  for(int i = 0; i < array.size() - 1; i++) {
     int j = i;
     int common_difference = array[i+1] - array[i];
     while(j < array.size() - 1 && (array[j + 1] - array[j] == common_difference))
          j++;
     int max_j = j;
     maxLen = max(maxLen, max_j - i + 1);
  }
  return maxLen;
}
```

Test Set 2

Consider an index i. Now consider all the subarrays (i,j) starting at index i and ending at index j. Start with j = i. Let the maximum index j where the subarray(i,j) is an arithmetic subarray be j = x. Let $\mathbf{A}_{i+1} - \mathbf{A}_i = D$. We can say that $\mathbf{A}_{y+1} - \mathbf{A}_y = D$ for all $i \le y < x$. We have 2 cases now.

- Case 1: $x = \mathbf{N}$,
  In this case, subarray(i, **N**) is an arithmetic subarray. All subarrays(p, **N**) such that $i < p \le N$, will have shorter length than subarray(i, **N**). Hence, we can discard all subarrays starting with index p.
- Case 2: $x \ne \mathbf{N}$,
  $\mathbf{A}_{x+1} - \mathbf{A}_x \ne D$. We have already proved that we need not consider j > x for index i as those subarrays will not be arithmetic using Claim 1. Now consider subarrays (k, x + 1) such that ($i+1 \le k < x$). All these subarrays are not arithmetic because $\mathbf{A}_{x+1} - \mathbf{A}_x \ne D$ whereas $A_{k+1} - A_k = D$. Hence, we can discard all the subarray starting with index k. So, we can now shift the starting index to x.

We can conclude our approach as follows. Initialise the answer as 0. We maintain two pointers, left pointer i and right pointer j. For an index i, we try to find the longest arithmetic subarray starting at index i by incrementing j. Let the maximum j for index i such that subarray(i,j) is an arithmetic subarray be max_j. Update answer if max_j - i + 1 is greater than current answer. And then we shift our left pointer i to the current max_j. We can see that both the pointers visit each element at most once. Hence, the complexity of the solution is O(**N**).

Sample Code (C++)

```
int maxArithmeticSubarray(vector<int> array) {
  int maxLen = 0;
  for(int i = 0; i < array.size() - 1;) {
     int j = i;
     int common_difference = array[i+1] - array[i];
     while(j < array.size() - 1 && (array[j + 1] - array[j] == common_difference))
          j++;
     int max_j = j;
     maxLen = max(maxLen, max_j - i + 1);
     i = max(i + 1, j);
  }
  return maxLen;
}
```