Rotate Function

Given an array of integers `A` and let n to be its length.

Assume `B`$_k$ to be an array obtained by rotating the array `A` k positions clock-wise, we define a "rotation function" `F` on `A` as follow:

`F(k) = 0 * B`$_k$`[0] + 1 * B`$_k$`[1] + ... + (n-1) * B`$_k$`[n-1]`.

Calculate the maximum value of `F(0), F(1), ..., F(n-1)`.

Note:
n is guaranteed to be less than 10^5.

Example:

```
A = [4, 3, 2, 6]

F(0) = (0 * 4) + (1 * 3) + (2 * 2) + (3 * 6) = 0 + 3 + 4 + 18 = 25
F(1) = (0 * 6) + (1 * 4) + (2 * 3) + (3 * 2) = 0 + 4 + 6 + 6 = 16
F(2) = (0 * 2) + (1 * 6) + (2 * 4) + (3 * 3) = 0 + 6 + 8 + 9 = 23
F(3) = (0 * 3) + (1 * 2) + (2 * 6) + (3 * 4) = 0 + 2 + 12 + 12 = 26

So the maximum value of F(0), F(1), F(2), F(3) is F(3) = 26.
```

Subscribe to see which companies asked this question

Solution 1

```
F(k) = 0 * Bk[0] + 1 * Bk[1] + ... + (n-1) * Bk[n-1]
F(k-1) = 0 * Bk-1[0] + 1 * Bk-1[1] + ... + (n-1) * Bk-1[n-1]
       = 0 * Bk[1] + 1 * Bk[2] + ... + (n-2) * Bk[n-1] + (n-1) * Bk[0]
```

Then,

```
F(k) - F(k-1) = Bk[1] + Bk[2] + ... + Bk[n-1] + (1-n)Bk[0]
              = (Bk[0] + ... + Bk[n-1]) - nBk[0]
              = sum - nBk[0]
```

Thus,

```
F(k) = F(k-1) + sum - nBk[0]
```

What is Bk[0]?

```
k = 0; B[0] = A[0];
k = 1; B[0] = A[len-1];
k = 2; B[0] = A[len-2];
...
```

```
int allSum = 0;
int len = A.length;
int F = 0;
for (int i = 0; i < len; i++) {
    F += i * A[i];
    allSum += A[i];
}
int max = F;
for (int i = len - 1; i >= 1; i--) {
    F = F + allSum - len * A[i];
    max = Math.max(F, max);
}
return max;
```

written by oreomilkshake original link here

Solution 2

Consider we have 5 coins A,B,C,D,E

According to the problem statement
F(0) = (0*A) + (1*B) + (2*C) + (3*D) + (4*E)
*F(1) = (4*A) + (0*B) + (1*C) + (2*D) + (3*E)
F(2) = (3*A) + (4*B) + (0*C) + (1*D) + (2*E)

This problem at a glance seem like a difficult problem. I am not very strong in mathematics, so this is how I visualize this problem

We can construct F(1) from F(0) by two step:
Step 1. taking away one count of each coin from F(0), this is done by subtracting "sum" from "iteration" in the code below
after step 1 F(0) = (-1*A) + (0*B) + (1*C) + (2*D) + (3*E)

Step 2. Add n times the element which didn't contributed in F(0), which is A. This is done by adding "A[j-1]*len" in the code below.
after step 2 F(0) = (4*A) + (0*B) + (1*C) + (2*D) + (3*E)

At this point F(0) can be considered as F(1) and F(2) to F(4) can be constructed by repeating the above steps.

Hope this explanation helps, cheers!

```
    public int maxRotateFunction(int[] A) {
        if(A.length == 0){
            return 0;
        }

        int sum =0, iteration = 0, len = A.length;

        for(int i=0; i<len; i++){
            sum += A[i];
            iteration += (A[i] * i);
        }

        int max = iteration;
        for(int j=1; j<len; j++){
            // for next iteration lets remove one entry value of each entry and t
he prev 0 * k
            iteration = iteration - sum + A[j-1]*len;
            max = Math.max(max, iteration);
        }

        return max;
    }
```

written by chiranjeeb2 original link here

Solution 3

```
public class Solution {

 public int maxRotateFunction(int[] A) {
  int n = A.length;
  int sum = 0;
  int candidate = 0;

  for (int i = 0; i < n; i++) {
   sum += A[i];
   candidate += A[i] * i;
  }
  int best = candidate;

  for (int i = n - 1; i > 0; i--) {
   candidate = candidate + sum - A[i] * n;
   best = Math.max(best, candidate);
  }
  return best;
 }
}
```

written by lzb700m original link here