**1. Problem Discussion :**

You are given an array “arr” of N integers. Where, the only operation allowed is to rotate(clockwise or counter clockwise) the array any number of times. Your task is to find the maximum value of sum of arr[i]\*i, where i = 0,1,2,...,n-1.

Example) For N = 4, arr[]={8,3,1,2}

The output is 29.

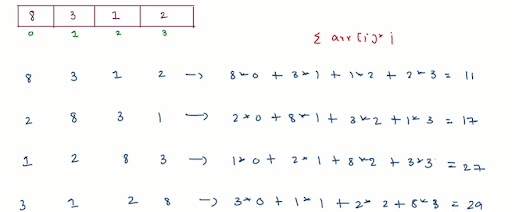
If we rotate the array by 3 as {3,1,2,8}.

Sum of arr[i]\*i = 3\*0 + 1\*1 + 2\*2 + 8\*3 = 29, which is maximum.

**2. Approach :**

Naive Approach :

One basic approach to solve this problem is to generate all possible rotations of given array and calculate the value of sum of arr[i]\*i for each rotation and then return the maximum value.

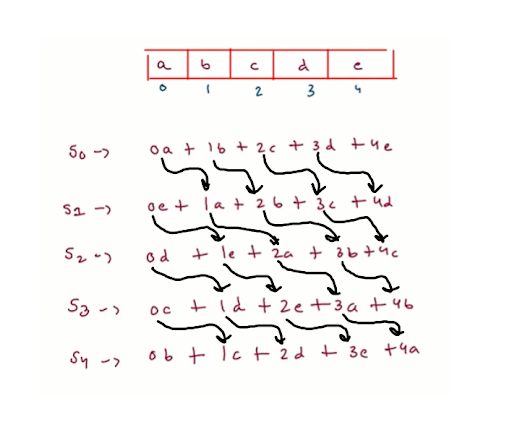
****

But using this approach will consume too much time and will have time complexity of O(n^2).

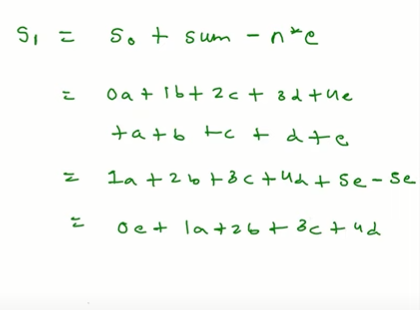
Optimized Approach :

Our optimized approach will improve over the naive approach. In the naive approach sum of arr[i]\*i was calculated for every rotation. If we are able to calculate sum of arr[i]\*i in constant time then we will be able to reduce the time complexity of naive approach.

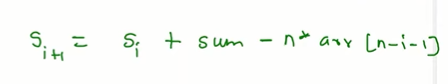
So, our basic idea is to somehow calculate the sum of arr[i]\*i of new rotation using the sum of arr[i]\*i of previous rotation. If we observe the coefficients of all elements in the sum of arr[i]\*i , only the coefficients of the first and last element change drastically and the coefficient of every other element increases or decreases by 1.

****

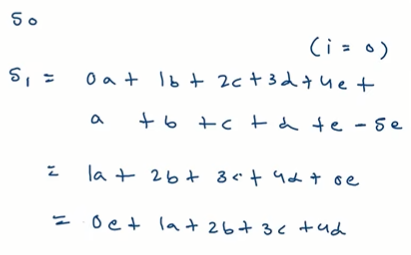
Now, we will try to find sum of arr[i]\*i for 1st rotation in terms of 0 rotations.

****

Now, we are able to see how we can calculate the sum of arr[i]\*i of new rotation using the sum of arr[i]\*i of previous rotation.

****

Using this formula we can calculate the sum of arr[i]\*i of new rotation using the sum of arr[i]\*i of previous rotation.

****

Now, we just need to calculate the sum of arr[i]\*i for all rotations using the derived formula and then return the maximum value.

**3. Code :**

ConsoleCpp

#include<bits/stdc++.h>

using namespace std;

int maximise(vector<int>& arr)

{

int n = arr.size();

int sum = 0;

int S0 = 0;

for(int i=0; i < n;i++) {

sum += arr[i];

S0 += arr[i]\*i;

}

int max = S0;

int Si = S0;

for(int i=0; i < n-1 ;i++) {

int temp = Si + sum - n\*arr[n-i-1];

Si = temp;

if(temp > max) {

max = temp;

}

}

return max;

}

int main()

{

int n;

cin>>n;

vector<int> arr(n,0);

for(int i =0;i<n;i++)

{

cin>>arr[i];

}

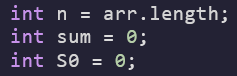
int ans = maximise(arr);

cout<<ans;

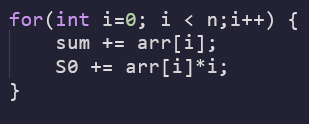
}

**4. Code Discussion**

1• First we need variables to store sum of all elements of array, the sum of arr[i]\*i for 0 rotation, also we stored the length of array for easier access.

****

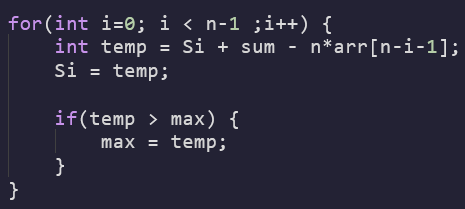
2• Here, we calculate the sum of all elements of array and the sum of arr[i]\*i for 0 rotation as a prerequisite for the formula we derived.

****

3• Max variable is declared for storing maximum sum of arr[i]\*i and Si variable is declared for storing the sum of arr[i]\*i for i number of rotation.

****

4• Here, we are calculating the sum of arr[i]\*i for each number of rotation and checking if have maximum sum of arr[i]\*i.

****

5• Now, we just need to return the maximum sum of arr[i]\*i that we calculated.

****

**5. Analysis :**

Time Complexity : O(n)

We only need to check the sum for all the rotations and find the maximum sum which takes O(n) time as the number of rotations for any array is from 0 to n-1 and since, the sum of the present rotation is calculated from the previous rotations in constant(O(1)) time, the overall time complexity of the function becomes O(n).

SPACE COMPLEXITY : O(1)

Since, no extra spaces are used to store numbers, therefore the space complexity is constant.