GREEDY FLORIST.

A group of friends want to buy a bouquet of flowers. The florist wants to maximize his number of new customers and the money he makes.

To do this he decides that he'll multiply the price of each flowers by the number customer's previously purchased flowers plus 1.

The first flower will be ORIGINAL PRICE, (0+1)* ORIGINAL PRICE, the next will be (1+1) * ORIGINAL PRICE and so on.

Given the size of the group of the friends, the number of flowers they want to purchase and the original prices of the flowers, determine the minimum cost of purchase all of the flowers. The no. of flowers they want equals the length of the c. array.

For example :→

C = [1 2 3 4]

K = 3

The length of c=4, so they want to buy 4 flowers total. Each will buy one of the flowers priced [2 3 4] at the original price. Having each purchased x=1, the first flower in the list, c[0], will now cost (current purchase + previous purchase) * c[0] = (1+1)*1 = 2

The total cost is 2+3+4+2 = 11.

Input: n: The number of flowers

k: The number of friends

An array of integers c, where c[i] is the price of i-th flowers.

output: The minimum cost of buying all the flowers.

Rules: - • Each friend buy atleast one flower.

• A friend can buy multiple flowers, but the

Approach

To minimize the cost, buy the most expensive flowers first since the cost increases with each purchase.

Use a greedy strategy.

1. Sort the flower prices in DESENDING ORDER.
2. DISTRIBUTE the flowers among friends, keep track of the multipleer for each friend.
3. Minimize the total cost by iterating through the sorted list and assigning flowers to friends sequentially.

ALGO GREEDY FLORISTS (C++)

```
int getMinimumCost(int k, vector<int>&c){
    sort(c.rbegin(), c.rend()); // Sort prices in decreasing order
    int totalCost = 0;
    int multiplier = 0;

    for(int i=0; i<c.size(); i++){
        totalCost += (multiplier + 1) * c[i];
        if((i+1) % k == 0) multiplier++; // Increment multiplier after every k
        flowers
    }
    return totalCost;
}

int main()
{
    ____
    ____
    ____
    vector<int> c(n);
    for(int i=0; i<n; i++){
        cout << "enter cost of flowers";
        cin >> c[i];
    }
```

```
    cout << getMinimumCost(k,c) <<endl;
    return 0;
}
```

TIME COMPLEXITY : $O(n \log n)$, where n is the number of flowers (for sorting).

SPACE COMPLEXITY: $O(1)$

as no extra space is used apart from input storage.