Ugly Number II

Here numbers should not have any other prime factor other than 2,3,5. So we will divide the number by 2/3/5 and if it’s not divisible by these numbers then it’s not ugly.

Example:
  4 is divisible by 2
  6 is divisible by 3
  14 is divisible by 2 but it will require 7 also for fully divisible. (Not ugly)
Though you can use this method for basic understanding it would give tle.
So we know that number should be in the multiple of 2,3,5 only so we will try to generate multiple of these numbers only.

We will take three-pointers t1,t2,t3 for generating the multiples of 2,3,5 respectively and take the minimum in each iteration

C++
```
class Solution {
public:
    int nthUglyNumber(int n) {
       vector<int>res(n);
       int i;
      res[0] = 1;
      int t1 = 0 ,t2 = 0 ,t3 = 0;
      for(i = 1;i < n ; i++)
      {
          res[i] = min(res[t1]*2,min(res[t2]*3,res[t3]*5));
          if(res[i] == res[t1]*2)t1++;
          if(res[i] == res[t2]*3)t2++;
          if(res[i] == res[t3]*5)t3++;
      }
        return res[n-1];
    }
};
```

Java

```
class Solution {
    public int nthUglyNumber(int n) {
      int res[] = new int[n];
        int i;
      res[0] = 1;
      int t1 = 0 ,t2 = 0 ,t3 = 0;
      for(i = 1;i < n ; i++)
      {
          res[i] = Math.min(res[t1]*2,Math.min(res[t2]*3,res[t3]*5));
          if(res[i] == res[t1]*2)t1++;
          if(res[i] == res[t2]*3)t2++;
          if(res[i] == res[t3]*5)t3++;
      }
        return res[n-1];

    }
}
```

Python
```
class Solution:
   def nthUglyNumber(self, n: int) -> int:
        ans, ptr2, ptr3, ptr5 = [1], 0, 0, 0
        for i in range(n-1):
            curr_ele = min(2*ans[ptr2], 3*ans[ptr3], 5*ans[ptr5])
            if curr_ele == 2*ans[ptr2]:
                ptr2 += 1
            if curr_ele == 3*ans[ptr3]:
                ptr3 += 1
            if curr_ele == 5*ans[ptr5]:
                ptr5 += 1
            ans.append(curr_ele)
        return ans[-1]
```

Subarray Sum Equals K

Though you can solve this problem by generating all subarrays and then adding them to find if it equals K or not.
But we would use a map here and find the prefix sum and then take a hashmap to check if (current sum - k) exists in map or not

Watch this video if you have trouble understanding.

Learn about hashmap in:
Java
C++

C++

```
class Solution {
 public:
     int subarraySum(vector<int>&nums, int k) {
        int n = nums.size();
        vector<int> prefixSum(n,0);
        int i , j;
        prefixSum[0] = nums[0];
        for(i = 1; i < n; i++)
            prefixSum[i] = prefixSum[i-1] + nums[i];
        map<int,int>m1;
        m1[0] = 1;
        int cnt = 0;
        for(i = 0; i < n; i++)
        {
            if(m1[prefixSum[i] - k])
            {
                cnt += m1[prefixSum[i] - k];
            }
            m1[prefixSum[i]]++;
        }
        return cnt;
    }
};
```

Java

```java
class Solution {
    public int subarraySum(int[] nums, int k) {
        int n = nums.length;
        int prefixSum[] = new int[n];
        int i , j;
        prefixSum[0] = nums[0];
        for(i = 1; i < n; i++)
            prefixSum[i] = prefixSum[i-1] + nums[i];
        HashMap<Integer,Integer> m1 = new HashMap<Integer,Integer>();
        m1.put(0 ,1);
        int cnt = 0;
        for(i = 0; i < n; i++)
        {
            if(m1.containsKey(prefixSum[i] - k))
            {
                cnt += m1.get(prefixSum[i] - k);
            }
            m1.put(prefixSum[i],m1.getOrDefault(prefixSum[i],0) + 1);
        }
        return cnt;
    }
}
```

Python

```
class Solution(object):
   def subarraySum(self, A, K):
    count = collections.Counter()
    count[0] = 1
    ans = su = 0
    for x in A:
        su += x
        ans += count[su-K]
        count[su] += 1
    return ans
```

Two Sum II - Input array is sorted

C++

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        int i = 0,j = nums.size()-1;
        while(i < j){
            int sum = nums[i] + nums[j];
            if(sum == target) return {i+1,j+1};
            else if(sum > target) j--;
            else i++;
        }
        return {};
    }
};
```

Java

```
class Solution {
   public int[] twoSum(int[] numbers, int target) {
        int start = 0, end = numbers.length - 1;
        while(start < end){
            if(numbers[start] + numbers[end] == target) break;
            if(numbers[start] + numbers[end] < target) start++;
            else end--;
        }
        return new int[]{start + 1, end + 1};
    }
}
```

Python

```python
class Solution(object):
    def twoSum(self, numbers, target):
        i = 0
        j = len(numbers)-1
        while i<j:
            if numbers[i] + numbers[j] == target:
                return [i+1,j+1]
            elif numbers[i] + numbers[j] > target:
                j-= 1
            else:
                i += 1
        return []
```

K-diff Pairs in an Array

Put all the numbers in hashmap and search for nums[i] + k if it is found we got a pair.

Possible edge cases while dealing with the map:
Check if there is any 0 required
Can number duplicate in the array
Can value be negative?

C++

```
class Solution {
public:
    int findPairs(vector<int>& nums, int k) {
      map<int,int>m1;
      int n = nums.size();
      if(n == 0 or k < 0)return 0;
      int i;
      for(i = 0; i < n; i++)
          m1[nums[i]]++;
      int res = 0;
      for(auto it:m1)
      {
          if(k==0)
          {
               if(m1[it.first] >= 2)
                   res++;
          }
          else{
            if(m1.find(it.first + k)!=m1.end())
            {
                res++;
            }
          }

      }
      return res;
   }
};
```

Java

```
class Solution {
    public int findPairs(int[] nums, int k) {
        if( k < 0  || nums.length == 0)
        return 0;
        int i,j;
        HashMap<Integer,Integer>m1 = new HashMap<>();
        for(i = 0; i < nums.length; i++)
          m1.put(nums[i],m1.getOrDefault(nums[i],0) + 1);
        int res = 0;
        for(int key:m1.keySet())
        {
          if(k == 0)
          {
              if(m1.get(key)>=2)
              res++;
          }
          else{
              if(m1.containsKey(key + k))
              res++;
          }
        }
        return res;
    }
}
```

Python
```
class Solution:
    def findPairs(self, nums: List[int], k: int) -> int:
        unique_num = Counter(nums)
        count = 0

        for x in unique_num:

            if k and x + k in unique_num:
                count += 1
            elif not k and unique_num.get(x+k, 0) >= 2:
                count += 1

        return count
```

Solve this problem from gfg