1 Array and Hashmaps

Arrays and hashmaps are datastructures used to store a collection of variables. We can access elements in an array by providing an integer based index. On the other hand, we can access elements of a hashmap by providing a unique key which corresponds to an element. The key could be any variable/object.

1.1 Group Anagrams

The key idea to solve this question would be that the template of a map could be yet another map.

```
map<map<char,int>,vector<string>> ans;
```

1. For each element in the given string, we would generate a

   ```
   map<char,int> obj
   ```

   where obj would keep count of the number of characters that have appeared in the given string.

2. Obj would be used as a key to insert the current string in the correct index of the ans map.

3. Lastly, we would just have to transfer the nested vectors in the ans map to the correct type. In this case, it is vector<vector<string».

2 Two Pointers

The main idea to two pointers would be to pick a pair of elements from an array, where the chosen pair of elements would satisfy some condition. More often than not, we would sort the array first.
A common application would be to find a pair of integers whose sum is = to some value.

1. Sort the array of integers

2. We would initialize the left integer iterator to be 0 to refer to the left most index, and the right integer iterator to be the size of the array to refer to the right most index.

3. If the current sum is too low, we would increase the left iterator to achieve a greater sum.

4. If the current sum is too high, we would decrease the right iterator to achieve a lower sum.

5. If the left iterator = right iterator, then we have not found a pair of integers to satisfy the given sum.

2.1 Valid Palindrome

The hardest part to this question would be to convert a arbitary string to an array of alphanumeric characters only. Then, we could apply the two pointers idea to compare the first and the last elements in the array, to see if they match.

```
vector<char> filtered;
for(int i=0;i<s.length();i++){
  if(isalnum(s[i])){
    filtered.push_back(tolower(s[i]));
   }
 }
```

2.2 3Sum

This problem could be reduced to the 2Sum problem, albeit with a couple of modifications.

1. If the given array has less than 3 elements, we can't really do anything. Thus, we should just return an empty array.

2. We would sort the given array at first.

3. Then, we would pick the first element (starting from the left of the array). Starting from index 1 and above, we should check the element to the left to see if we have already checked it. This step is critical to ensure that we do not create duplicate triplets.

4. Then, we would use the two pointers idea to find a pair of elements which would satisfy the criteria of $ele1 + ele2 + ele3 = 0$. Once we have found a suitable pair, we should continue checking, as there could be more than one solution.

5. We could use a set to ensure that we do not generate identical triplets, before copying the contents of the set back to a vector of vector<int>.