Todays Content.

1. CountSort
2. Max triplet prod
3. Pair<>;
4. Vector<pair<>>;

CountSort:

Given arr[N] all ele in range [2-6] sort arr[] in Inc.

	0	1	2	3	4	5	6	7	8	9
arr[10] = {	3	2	4	6	4	2	3	4	3	6 }
arr[10] = {	2	2	3	3	3	4	4	4	6	6 }

Idea1: Apply BS/IS/SS : O(N^2)
Apply MS : O(NlogN)

Idea2: Store freq of hashmap & use freq to sort arr[]

hashmap
2 : 2
4 : 3
6 : 2
3 : 3

Issue1: Hashmap is not ordered
Resolve: Iterate in range 2..6
For every element get freq:
Store that ele those many times in arr[]:

Dry Run:

hashmap
2 : 2
4 : 3
6 : 2
3 : 3

i : [2..6]	Freq: i	j iterat
i : 2	: Freq: 2	2
i : 3	: Freq: 3	3
i : 4	: Freq: 3	3
i : 5	: Freq: 0	0
i : 6	: Freq: 2	2

Array size = 10

0	1	2	3	4	5	6	7	8	9
~~3~~	~~2~~	~~4~~	~~6~~	~~4~~	~~2~~	~~3~~	~~4~~	~~3~~	~~6~~
2	2	3	3	3	4	4	4	6	6

j → j → j → j → j → j → j → j → j → j → j

Note: j = 0 : Initialized at start
In each iteration we continue j →

Total iterations :
Total Innerloop = N
Total Outer loop = 6-2+1 = 5

#Con: We use freq of each element to sort arr[]
This is considered freqsort / CountSort

```
vector<int> sort(vector<int> &A) {  TC: O(N+6+N) = O(N)
    #Step1:                          SC: O(N) -> if all elements are diff
    unordered_map<int,int> hm;
    for(int i=0; i<A.size(); i++){
        hm[A[i]]++;
    }


    #Step2:
    int j=0;                         Total iterating: O(6+N)
    for(int i=2; i<=6; i++){
        if(hm.find(i)==hm.end()){continue;}
        int f=hm[i]; # if i doesn't exist it will retn 0;
        #Copy i, f times in array.
        for(int l=1; l<=f; l++){
            ar[j]=i;
            j++;
        }
    }


    return ar;
}
```

Given an arr[] where all ele in Range [a..b] sort arr[].

#a = min of arr[] b = max of arr[]

```
vector<int> sort(vector<int> &arr, int a, int b){
    #Step1:
    unordered_map<int,int> hm;
    for(int i=0; i<A.size(); i++){
        hm[A[i]]++;
    }
```

TC: O(N + R + N)
TC: O(N+R)
SC: O(N)

Storing ele in hashmap

outerloop Innerloop

Total iterating = O(b-a+1 + N)

Assume R = b-a+1; # R = Range

Total iterating = O(R+N)

```
    #Step2:
    int j=0;
    for(int i=a; i<=b; i++){
        if( hm.find(i)==hm.end()){continue;}
        int f= hm[i]; # if i doesn't exist it will retn 0;
        #Copy i, f times in array.
        for( int l=1; l<=f; l++){
            arr[j]=i;
            j++;
        }
    }

    return arr;
}
```

When to apply CountSort:

look at constrains: $1 <= ar[i] <= 10^5$

Sort ar[N] elements & assume max - min + 1 = R

Range.

	MergeSort : $O(N \log N)$	CountSrr $O(N+R)$
if ($R \approx N$)	$O(N \log N)$	$O(N+N) \approx O(N)$ ✓
if ($R \approx N \log N$)	$O(N \log N) * 1$ ✓	$O(N+N \log N) \approx O(N \log N) * O(1)$
if ($R > N \log N$)	$O(N \log N)$ ✓	$O(N+R) \approx O(R) > O(N \log N)$

Q Given ar[N] elements calculate min Triplet product

Constraints:

$1 <= N <= 10^6$

$1 <= ar[i] <= 10^9$

	0	1	2	3	4	5	6
Eg: ar[] = {	3	6	7	4	2	9	8 }

ans = 2 * 3 * 4 = 24

Idea1: Sort arr() & multiply 1st 3 elements

TC: O(NlogN + 1) = O(NlogN)

Idea2: Apply selection sort but only 3 times

```
for(int i=0; i<3; i++){   TC: O(3*N) = O(N)   SC: O(1)
    // i: j: [i..N-1] & calculate mini;
    int mini = i;
    for(int j=i; j<N; j++){
        // ar[mini] && ar[j]
        if(ar[j] < ar[mini]){
            mini = j;
        }
    }
    swap ar[i] & ar[mini]
}

return ar[0] * ar[1] * ar[2]
```

Q: Given ar[N] & M : Mostly skip it

Calculate length of longest subset, without a pair (i, j) such that (ar[i] + ar[j]) % M = 0.

Pair: When we want to combine more than 1 value into single variable

#library:

#include <utility>

#Declaration:

pair<Type1, Type2> p1; p1 < > ← default Initialization

pair<int, float> p1; p1 < 0, 0.0 >

pair<Type1, Type2> p2(v1, v2);

pair<String, int> p2("shaw", 24); p2 < shaw, 24 >

pair<Type1, Type2> p3 = make_pair(v1, v2);

pair<int, int> p3 = make_pair(10, 20); → < 10, 20 >

pair<Type1, Type2> p4 = {v1, v2}

pair<int, int> p4 = {10, 20} → <10, 30>

Note: Use make_pair(v1, v2); or {v1, v2} to create pairs.

#Auss:

p {v1, v2} v1 = P.first v2 = p.second

```
pair<Type1, Type2> p4 = {10,20}  #   P4 = <10, 20 50>
print( P4.first); #10
P4.secnd = P4.secnd+30; # 20+30=50
```

Comparism:

When we compare 2 pairs by default below procss followed.

Step1: 1st element in both are compared first

Step2: If 1st element is same 2nd element in both are compared.

```
pair<type1, type2>  p1(10, 20);
pair<type1, type2>  p2(20, 5);
pair<type1, type2>  p3(10, 30);
```

Output

```
print( P1 == P2)   false
print( P2 > P1 )   true
print( P1 > P3 )   false;
print( P3 > P2 )   false;
```

```
pair< pair<int, int>, pair<int, float> > p1;
```

Type1, Type2

P1.first.first P1.first

P1.sec.sec P1.sec

Pair in vector:

vector<type> v; → type can be int/long/string/<Pair>/Object/...

Eg: vector<pair<type1, type2>> v;

Eg: vector<pair<int, int>> v;

```
v.push_back({10, 20});
v.push_back({20, 30});
v.push_back(make_pair(50, 60));
v.push_back({15, 25});
```

v	
0	<10, 20>
1	<20, 30>
2	<50, 60>
3	<15, 25>

Iteration in vector:

#Way1:

```
for(int i=0; i<v.size(); i++){
    # v[i] is pair
    print(v[i].first, v[i].second);
}
```

v	
0	<10, 20>
1	<20, 30>
2	<50, 60>
3	<15, 25>

#Way2: Copy stored in it & it's printed.

```
for(auto it : v){
    # it is pair
    print(it.first, it.second)
}
```

auto it : <1, 2>

Note: updating in it won't effect original, because it is copy.

<1, 2>
<3, 4>
<10, 20>
<5, 6>

#Way3

Pass by reference, it & pairs share same memory, so if we update in it, it will update in original

```
for(auto &it : vec){
    # it is pair
    print(it.first, it.second)
}
```