

Today's Content.

1. Count Sort
2. Max triplet prod
3. Pair $\< \>$;
4. Vector & pair $\< \>$;

CountSort:

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

arr[10] = { 3 2 4 6 4 2 3 4 3 6 }

arr[10] = { 2 2 3 3 3 4 4 4 6 6 }

Idea: Apply BS/IS/SS : $O(N^2)$

Apply MS : $O(N \log N)$

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

HashMap Issue: HashMap is not ordered

2 : 2 Resolve: Iterate in range 2..6

4 : 3

6 : 2

3 : 3

For every element get freq:

Store that use those many times in arr[]:

Dry Run:

	i: [2..6]	Freq: i	j iterat
<u>HashMap</u> 2 : 2 4 : 3 6 : 2 3 : 3	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	X	6	X	2	3	X	6	6
2	2	3	3	3	4	4	4	6	6
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 Inner loop = N

Total Outer loop = 6 - 2 + 1 = 5

Con: we use freq of each element to sort arr[]

This is considered freq sort / Count Sort

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) { TC: $O(N + R + N)$

Step 1:

unordered_map<int, int> hm;

for (int i = 0; i < A.size(); i++) {

 hm[A[i]]++;

}

TC: $O(N + R)$

SC: $O(N)$

Storing ele in hashmap

outerloop innerloop

Total iterations = $O(b - a + 1 + N)$

Step 2:

Assume $R = b - a + 1$; # $R = \text{Range}$

int j = 0;

Total iterations = $O(R + N)$

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 return 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 constraints: $1 \leq \text{arr}[i] \leq 10^5$

Sort $\text{arr}[N]$ elements & assume $\text{max} - \text{min} + 1 = R$

MergeSort: $O(N \log N)$

CountSort $O(N + R)$ ^{Range.}

if $R \approx N$

$O(N \log N)$

$O(N + N) \approx O(N) \checkmark$

if $R \approx N \log N$

$O(N \log N) * 1 \checkmark$

$O(N + N \log N) \approx O(N \log N) * O(1)$

if $R > N \log N$

$O(N \log N) \checkmark$

$O(N + R) \approx O(R) > O(N \log N)$

Q Given $arr[N]$ elements calculate min Triplet product

Constraints:

$$1 \leq N \leq 10^6$$

$$1 \leq arr[i] \leq 10^9$$

Ex: $arr[] = \{ \overset{0}{\underline{3}} \overset{1}{6} \overset{2}{7} \overset{3}{\underline{4}} \overset{4}{\underline{2}} \overset{5}{9} \overset{6}{8} \}$ ans = $2 * 3 * 4 = 24$

Idea1: Sort $arr[]$ & multiply 1^{st} 3 elements

$$TC: O(N \log N + 1) = O(N \log N)$$

Idea2: Apply solution for 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 min i;
    int mini = i;
    for(int j=i; j<N; j++) {
        # arr[mini] & arr[j]
        if(arr[j] < arr[mini]) {
            mini = j;
        }
    }
    swap(arr[i], arr[mini]);
}
```

return $arr[0] * arr[1] * arr[2]$

Q: Given $arr[N]$ & M : Mostly skip it

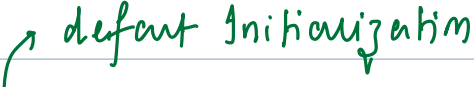
Calculate length of longest subset, without a pair (i, j)
such that $(arr[i] + arr[j]) \% M = 0$.

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


#library:


#include <utility>

#Declaration:

pair<Type1, Type2> p1; p1 < >
 pair<int, float> p1; p1 < 0, 0.0 >
 

pair<Type1, Type2> p2(v1, v2);
 pair<String, int> p2("sham", 24); p2 < sham, 24 >

pair<Type1, Type2> p3 = make_pair(v1, v2);
 pair<int, int> p3 = make_pair(10, 20);
 

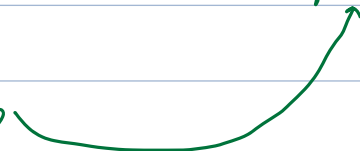
pair<Type1, Type2> p4 = {v1, v2}
 pair<int, int> p4 = {10, 20}
 

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

#Access:

$p\{v_1, v_2\}$ $v_1 = p.first$ $v_2 = p.second$

```
pair<Type1, Type2> p1 = {10, 20} # p1 = {10, 20 50}
print(p1.first); #10 ✓
p1.second = p1.second + 30; # 20 + 30 = 50
```



Comparison:

When we compare 2 pairs by default below process 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
```

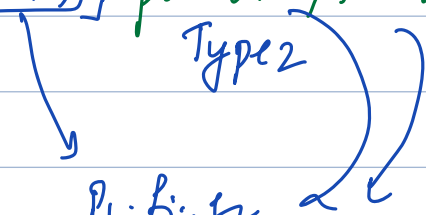
```
print(p2 > p1) true
```

```
print(p1 > p3) false;
```

```
print(p3 > p2) false;
```

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

Type1, Type2



$p1.first$ for $p1.first$
 $p1.second$ for $p1.second$

Pair in vector:

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

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

Ex: 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}

#way 3

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)
```

3

auto it:

