Problem 1:

Since the maximum element contributes to two maximums so the two maximum elements must be equal.
You can just print one maximum and two smallest elements.

```
while (t--)
	{
		vector<int> v(3);
		cin >> v[0] >> v[1] >> v[2];
		sort(v.begin(), v.end());
		if (v[2] == v[1])
		{
			cout << "YES" << endl;
			cout << v[2]<<" " << v[0]<<" " << v[0] << endl;
		}
		else
			cout << "NO" << endl;
	}
```

Problem 2 :
Just take the first occurrence of each element

```
while (t--)
	{
		cin >> n;
		set<int> s;
		vector<int> v;

		for (int i = 0; i < 2 * n; ++i)
		{
			cin >> n0;
			if (s.find(n0) == s.end())
			{
				v.push_back(n0);
				s.insert(n0);
			}
		}
		for (int i = 0; i < n; ++i)
			cout << v[i]<<" ";
		cout << endl;
	}
```

Problem 3:

```
while (t--)
	{
		cin >> n;
		vector<int> v(n);
 //just iterate from the last
//either all the elements must be in non-increasing order from the last
//or they must be non-decreasing and then non-increasing
		for (int i = 0; i < n; ++i)
		{
			cin >> v[i];

		}
//skip  all the non-increasing elements first
		int i = n - 1;
		while (i > 0)
		{
			if (v[i] <= v[i - 1])
				--i;
			else
				break;
		}
//after  that skip all the non-decreasing elements
		while (i > 0)
		{
			if (v[i] >= v[i - 1])
				--i;
			else
				break;
		}
//finally these prefix elements must be removed
		cout << (i) << endl;
	}
```

Problem 4:
// I have used the brute force approach here
//Recursion is used
//either you need to fill first half with a or the last half

```
ll ans;
void make(string s, char a,ll count)
{
//this is to  handle the base case
//if the size of string is 1 then we have completed
// so count will give the ans and take the minimum of those
	if (s.length() == 1)
	{
		if (s[0] != a)
			++count;
		ans = min(ans, count);
		return;
	}
//saving the count for  next half
	ll savc = count;
//changing the first half
	for (int i = 0; i < s.length()/2; ++i)
		if (s[i] != a)
			++count;
//since we have changed the first half
//we need to make the last half c+1
	make(s.substr(s.length() / 2, s.length() / 2), a + 1, count);
//changing the second half
	for (int i = s.length()/2; i < s.length(); ++i)
		if (s[i] != a)
			++savc;
//since you have changed the second half
//make the first half c+1
	make(s.substr(0, s.length() / 2), a + 1, savc);
}

int main()
{
	IOS;
	ll n, c0, c1, c2, t, sum;
	cin >> t;
	string s;
	while (t--)
	{
		cin >> n;
		cin >> s;
		ans = INT_MAX;
```

```
            make(s, 'a', 0);
            cout << ans << '\n';
        }
        return 0;
}
```

Problem 5:
First you need know about topological sort to understand my solution
//mycode for topological sort using indegree

```
vector<int> topological_sort(vector<vector<int>>& graph,
int n,vector<int>& indegree)
{
    vector<int> ans;
    stack<int> s;
    for (int i = 0; i < n; ++i)
        if (indegree[i] == 0)
            s.push(i);
    while (!s.empty())
    {
        int x = s.top();
        s.pop();
        ans.push_back(x);
        for (auto& adj : graph[x])
            if (--indegree[adj] == 0)
                s.push(adj);
    }
    return ans;
}
```

//Then you just need to get the topological sort using the given directed edges
//if we can't get the topological sort including all the vertices he answer is NO
//otherwise the answer is yes and direct the undirected edges in the order of topological sort

```
int main()
{
	IOS;
	int t, n,m,c,x,y;
	cin >> t;
	while (t--)
	{
		cin >> n>>m;
//to store he indegree of vertices
		vector<int> indegree(n,0);
//to store the adjacency list of graph
		vector<vector<int>> graph(n);
//to store the undirected edges
		vector<vector<int>> undirected_edges;
		for (int i = 0; i < m; ++i)
		{
			cin >> c >> x >> y;
//converting 1 to n into 0 to n-1
			--x; --y;
//if directed edge include it in graph
			if (c)
			{
				graph[x].push_back(y);
				++indegree[y];
			}
//store undirected edges for future reference
			else
				undirected_edges.push_back({ x,y });
		}
//get the topological sort of the graph
		vector<int> ans = topological_sort(graph, n,
indegree);
//if the topological sort doesn't include all vertices that
means the graph has a cycle
		if (ans.size() != n)
		{
			cout << "NO" << endl;
			continue;
		}
```

```
//else the answer is yes
cout << "YES" << endl;
//print the directed edges
          for (int i = 0; i < n; ++i)
               for (int j = 0; j < graph[i].size(); ++j)
                    cout << i + 1 << " " << graph[i][j] + 1
<< endl;
//for checking the order of vertices in topological sort
          vector<int> check(n);
          for (int i = 0; i < n; ++i)
               check[ans[i]] = i;
          for (auto& i : undirected_edges)
          {
//if i[1] comes before i[0] in topological sort
               if (check[i[0]] > check[i[1]])
                    cout << i[1] + 1 << " " << i[0] + 1 <<
endl;
//if i[0] comes before i[1]
               else
                    cout << i[0] + 1 << " " << i[1] + 1 <<
endl;
          }
     }
     return 0;
}
```

// If anyone has any confusion comment down I will try to help

We can discuss in this discord server as well
https://discord.gg/en2AK26
https://discord.gg/N8SbEqh

//I will post the link to the whole source code
1.https://codeforces.com/contest/1385/submission/87083896
2.https://codeforces.com/contest/1385/submission/87092072
3.https://codeforces.com/contest/1385/submission/87103588
4.https://codeforces.com/contest/1385/submission/87136548

5.https://codeforces.com/contest/1385/submission/87239820