CLUB OF DATA ENGINEERS
SCHOOL OF INFORMATION SCIENCES

Post Contest

Theme : Sorting

By : Azgax & Salim The Feminist

TOC

Problem A : Busy Schedule

The Problem

- Context:
  - You have multiple daily appointments written in the format H:M Z (e.g., 3:00 p.m.).
  - Each day begins at midnight (12:00 a.m.), and appointments must be sorted chronologically.
  - Input ends with a test case count of 0.
- Goal:
  - Sort the appointments for each day from earliest to latest.
  - Output formatted times for each test case, separated by blank lines.

Solution Strategy

Step to solve

1 Parse the Input

- Read the number of appointments.
- Extract appointment times for each test case.

2 Convert Time for Sorting

- Convert H:M Z into a comparable 24-hour numeric format.
- Handle special cases:
- 12:00 a.m. becomes 00:00.
- 12:00 p.m. remains 12:00.

3 Sort the Times

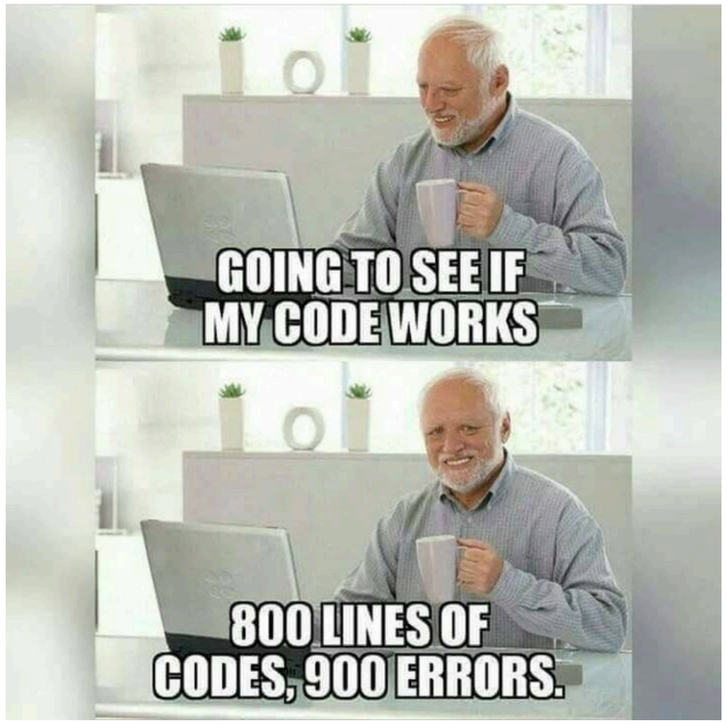
- Use a custom sorting function to arrange the times numerically.

4 Format Output

- Convert sorted numeric times back into the original format.
- Separate test case outputs with a blank line.

Code Explanation

Lessssgooooo to



Code Explanation

busyschedule.cpp

```cpp
#include <bits/stdc++.h>
#include <algorithm>
using namespace std;
int main()
{
 int n=100;
 while(n>0)
 {
  cin>>n;
  vector<vector<string>> v(n);
  for(int i=0;i<n;i++)
  {
   string time,period;
   cin>>time>>period;
   int del=0;
   while(time[del]!=':')
   {
    del++;
   }
   string hour(time.begin(),time.begin()+del);
   string minute(time.begin()+del+1,time.end());
   v[i]={hour,minute,period};
  }
  for(int i=0;i<n;i++)
 {
  int ind=i;
```

busyschedule.cpp

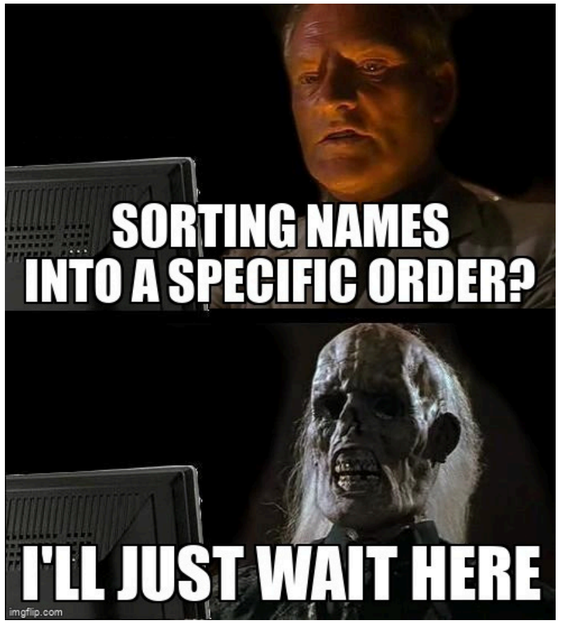
```cpp
  for(int j=i+1;j<n;j++)
  {
   bool isLess=false;
   if (v[j][2] != v[ind][2]) {
    isLess = (v[j][2] == "a.m.");
   } else
   {
    int hour1 = (v[ind][0] == "12") ? 0 : stoi(v[ind][0]);
    int hour2 = (v[j][0] == "12") ? 0 : stoi(v[j][0]);
    if (hour1 != hour2) {
     isLess = hour2 < hour1;
    } else {
     isLess = stoi(v[j][1]) < stoi(v[ind][1]);
    }}
   if (isLess) {ind = j;}
  }
  swap(v[i],v[ind]);
 }

 for(int i=0;i<n;i++)
 {
  cout<<v[i][0]<<":"<<v[i][1]<<" "<<v[i][2]<<"\n";
 }
 cout<<'\n';
}
return 0;
}
```

Problem B : Sort of Sorting

The Problem

- Context:
  - The school administration needs to quickly access student records.
  - Names need to be sorted based only on their first two letters.
  - Sorting must be stable (maintain the original order for names with the same first two letters).
- Goal:
  - Implement a stable sorting algorithm using only the first two letters of names.



Solution Strategy

Steps to solve

1 Parse the Input

- Read the number of names in the test case.
- Extract names for each test case.

2 Convert Time for Sorting

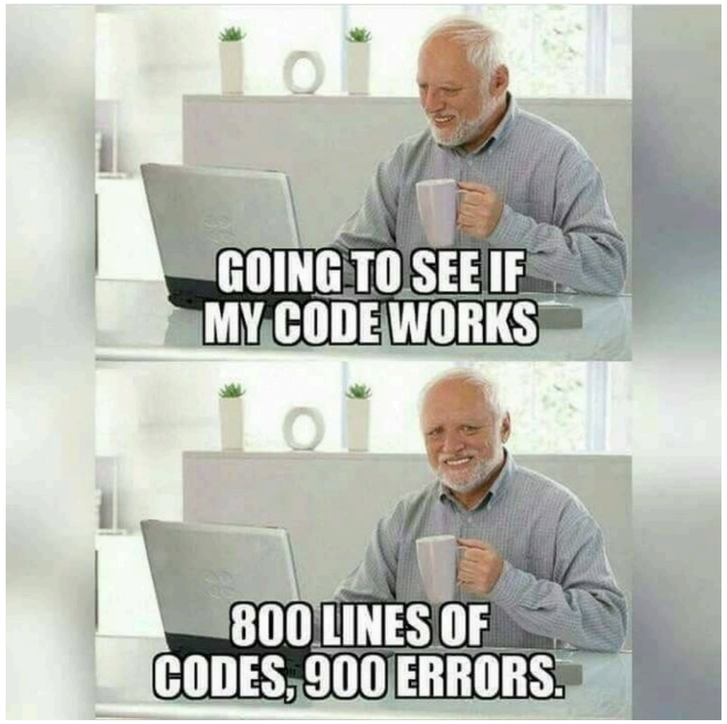
- Use a stable sorting algorithm (e.g., bubble , insertion...) to preserve the input order for ties.

3 Output the Results

- Print names in sorted order for each test case.
- Separate outputs of consecutive test cases with a blank line.

Code Explanation

Lessssgooooo to



Code Explanation

sortofsorting.cpp

```
#include <bits/stdc++.h>
#include <algorithm>
using namespace std;

int main()
{
 int n=100;
 while(n)
 {
  cin>>n;
  vector<string> v(n);
  for(int i=0;i<n;i++)
  {
   cin>>v[i];
  }
  for(int i=1;i<n;i++)
  {
   int j=i-1;
   string cur=v[i];
   while(j>=0 && (cur[0]<v[j][0] || (cur[0]==v[j][0] && cur[1]<v[j]
[1])))
   {
    v[j+1]=v[j];
    j--;
   }
   v[j+1]=cur;
  }
```

sortofsorting.cpp

```
   v[j+1]=cur;
  }

  for(int i=0;i<n;i++)
  {
   cout<<v[i]<<"\n";
  }
  cout<<'\n';
 }
 return 0;
}
```

Problem C : Odd Man Out

The Problem

- Context:
  - You are hosting a party with an odd number of guests.
  - Each couple receives a unique invitation number.
  - One guest has arrived alone (their invitation number appears only once)
- Goal:
  - Identify the invitation number of the guest who is alone.

Solution Strategy

Step to solve

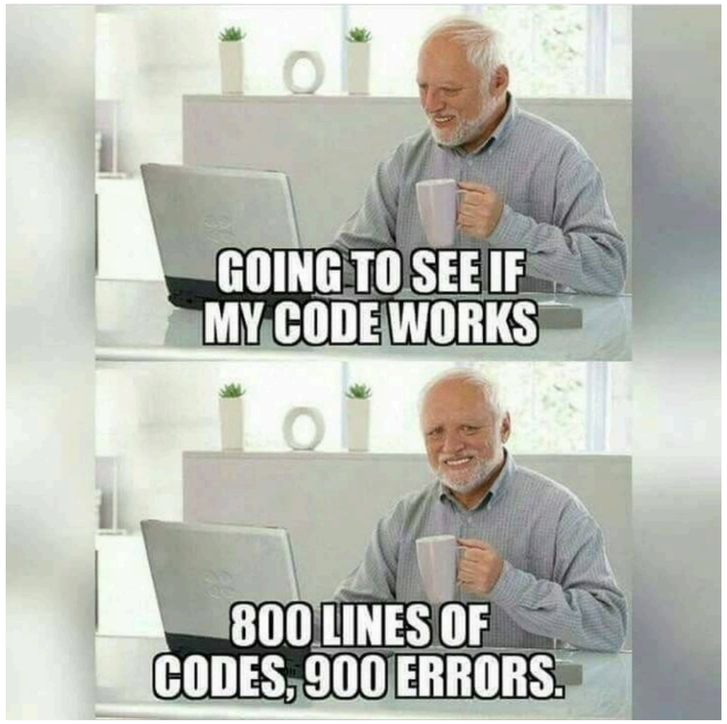
1. **Parse the Input**
   - Read the number of test cases, T.
   - For each test case, read the number of guests, G, and their invitation numbers.

2. **find the lonely one**
   - sort the Numbers in increasing order.
   - each time , you compare ith element and i+1th element, if they aren't we found our target

4. **Format and Output Results**
   - Output results in the format Case #x: y.

Code Explanation

Lessssgooooo to



Code Explanation

oddmanout.cpp

```cpp
#include <bits/stdc++.h>

using namespace std;

int main()
{
 int t;
 cin>>t;
 int k=0;
 while(k<t)
 {
  int g;cin>>g;
  vector<long long> v(g);
  for(int i=0;i<g;i++)
  {
   cin>>v[i];
  }

  for(int i=0;i<g;i++)
  {
   int ind=i;
   for(int j=i+1;j<g;j++)
   {
    if(v[j]<v[ind])
    {
     ind=j;
    }
   }
```

oddmanout.cpp

```cpp
  swap(v[ind],v[i]);
  }

  int i=0;
  while(i<g && v[i]==v[i+1])
  {
     i=i+2;
  }
   cout<<"Case #"<<k+1<<": "<<v[i]<<endl;
   k++;
  }
  return 0;
 }
```

Problem D : Quick Brown Fox

The Problem

- Context:
  - A pangram contains every letter of the alphabet (a-z) at least once.
  - Example pangram: "The quick brown fox jumps over the lazy dog."
- Goal:
  - Determine if a phrase is a pangram.
  - If not, identify the missing letters (in lowercase and sorted alphabetically).

Solution Strategy

Step to solve

1. **Parse the Input**
   - Read the number of test cases, T.
   - For each test case, read the number of guests, G, and their invitation numbers.

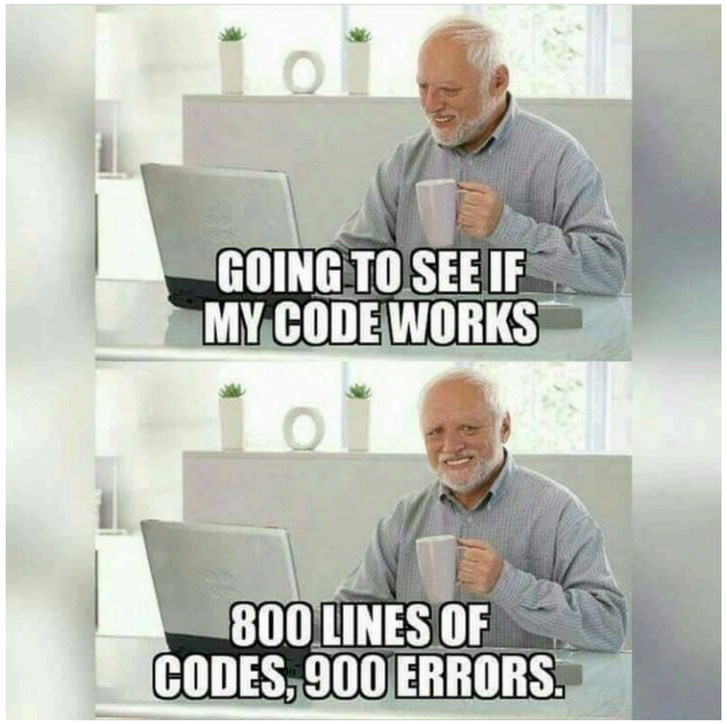
2. **Check for Pangram**
   - Convert all letters in the phrase to lowercase.
   - Use a queue to track letters in order.
   - Compare the set of letters in the phrase to the full alphabet set (a-z).

3. **Identify Missing Letters**
   - find if the queue is empty or not.

4. **Format and Output Results**
   - Print pangram or missing followed by the missing letters for each phrase.

Code Explanation

Lessssgooooo to



Code Explanation

quickbrownfox.cpp

```cpp
#include <iostream>
#include <queue>
#include <bits/stdc++.h>



using namespace std;

int main()
{
 int n;cin>>n;cin.ignore();
 queue<char> q;

 while(n--)
 {
  string str;
  getline(cin,str);
  transform(str.begin(), str.end(), str.begin(),::tolower);
  for(int i='a';i<='z';i++)
  {
   if(find(str.begin(),str.end(),i)==str.end())
   {
    q.push(i);
   }
  }
```

quickbrownfox.cpp

```cpp
  if(q.empty())
  {
    cout<<"pangram"<<endl;
  }
  else
  {
    cout<<"missing ";
    while(!q.empty())
   {
     cout<<q.front();
     q.pop();
   }
   cout<<'\n';
  }
 }

}
```

Problem E : Closest Sums

The Problem

- Context:
  - For each query, you need to find the sum of two distinct numbers from the set of integers such that this sum is closest to the query value.
- Goal:
  - For each query, print the query integer followed by the closest sum.

Solution Strategy

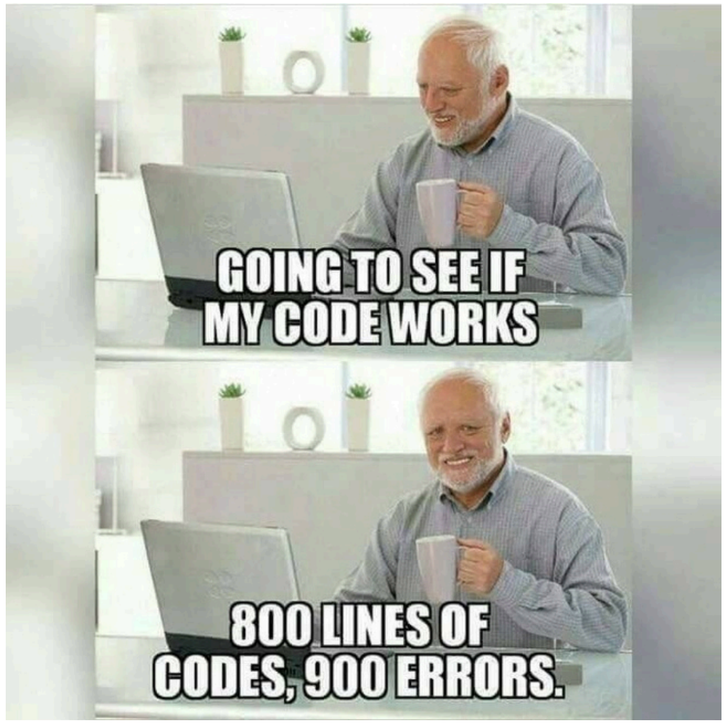
Step to solve

1. **Parse the Input**
   - the number of test cases is unknown so the loop is open.
   - read the array
   - each time reach the query

2. **sort and sort and sort**
   - we sort the array.
   - for each query, we search the closest sum of two elements.
   - once we we found it we format the output.

Code Explanation

Lessssgooooo to



Code Explanation

closestsums.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
 int case_num = 1;

 while (true) {
  int n;
  if (!(cin >> n)) break;

  int arr[n];
  for (int i = 0; i < n; i++) {
   cin >> arr[i];
  }

  int m;
  cin >> m;
  cout << "Case " << case_num << ":\n";

  for (int i = 0; i < m; i++) {
   int q;
   cin >> q;
   int ans = INT_MAX;
   int min_diff = INT_MAX;
```

closestsums.cpp

```cpp
 for (int j = 0; j < n; j++)
  {
  for (int k = j + 1; k < n; k++)
   {
  int sum = arr[j] + arr[k];
  int diff = abs(sum - q);


    if (diff < min_diff)
    {
      min_diff = diff;
      ans = sum;
     }
    }
   }
 cout << "Closest sum to " << q << " is " << ans << ".\n";
  }
 case_num++;
 }

 return 0;
}
```