Credits!

- **Chief Judge**: Balajiganapathi **Balajiganapathi** S
- **Contest Admin**: Vichitr **Vichitr** Gandas
- **Problem Coordinators**: Ashish **Ashishgup** Gupta, Kevin **kevinsogo** Atienza, Vaibhav **xennygrimmato** Tulsyan
- **Setters**: Ajinkya **ajinkya1p3** Parab, Vichitr **Vichitr** Gandas, Divyansh **failed_coder** Verma, Udit **T1duS** Sanghi
- **Testers**: Kshitij **kshitij_sodani** Sodani, Ashish **Ashishgup** Gupta, Kevin **kevinsogo** Atienza, Naveen **mnaveenkumar2009** Kumar, Soumyaditya **socho** Choudhuri, Shashwat **SleepyShashwat** Chandra, Suneet **ScarletS** Mahajan, Ajinkya **ajinkya1p3** Parab, Vichitr **Vichitr** Gandas, Divyansh **failed_coder** Verma, Udit **T1duS** Sanghi, Ritul **ritul_kr_singh** Singh, Het **hetshah1998** Shah, Varad **playing_it_my_way** Kulkarni, Aswin **aswinashok44** Ashok, Mayank **katana_handler** Pugalia, Daanish **7dan** Mahajan and Vaibhav **xennygrimmato** Tulsyan
- **Platform Coordinator**: Deepa **deepa_panwar** Panwar
- **RCD**: Maheshwara Chaitanya

1. The Married-Unmarried Riddle
2. Express the Number
3. Break, Merge and Sort
4. Minimum Weight Bi-partition
5. Strings and LIS
6. Tree Max Sum

Cakewalk
Simple
Easy
Easy-Medium
Medium
Medium-Hard

ICPC 2020 - India Preliminaries

Presentation of solutions

Problem 1

The Married-Unmarried Riddle

No. of accepted solutions: 3248

Author: Ajinkya Parab

The Married-Unmarried Riddle

- Check if there exists an unmarried person after a married person.
- If yes, then replacing the ‘?’s with anything and we’ll always get an MU pair.
- Otherwise, we can replace the ‘?’s to not get an MU pair. (How?)
- Complexity: **O(N)**, or $O(N^2)$ based on implementation.

Problem 2
Express the Number

No. of accepted solutions: 1918

Author: Vichitr

Express the Number

- We can use only odd powers.
- An even power can also be represented as a sum of odd powers.
- Write n as the sum of powers of 2.
- If there is some power $2^k > x$ and k is odd, then we can subtract 2^k from n and increase ans by 1.
- If k is even, we can subtract $2^{k-1} + 2^{k-1}$.
- Notice that after subtracting just one 2^{k-1}, n can become $\leq x$.
- After we get $n \leq x$, we can simply write it as A[1] = left-out n.
- When we can't express n as specified format? (Odd n, x = 0)
- Complexity: **O(log n)**

Problem 3

Break, Merge and Sort

No. of accepted solutions: 1069

Author: Ajinkya Parab

Break, Merge and Sort

- *Greedy insight 1:* We may do all splits before merges.
  - Any other sequence can be converted to an equivalent one where splits come before merges, with no worse cost.
- e.g.,
  (**abcd**,abcd) → (**a**a**b**b**c**c**d**d) → (**a**a**b**b,**c**c**d**d)
  can be converted to
  (**abcd**,abcd) → (**ab**,**cd**,abcd) → (**ab**,**cd**,ab,cd) → (**a**a**b**b,**cd**,cd) →
  (**a**a**b**b,**c**c**d**d)

Break, Merge and Sort

- Now, we obviously need to cut at places where A[i] > A[i+1].
- *Greedy insight 2:* No additional cuts are needed.
- Can be proven after discovering the optimal way to split and merge.
- *Greedy insight 3:* If $A = A_1 + A_2 + \ldots + A_n$, then the splitting of $A \rightarrow (A_1, A_2, \ldots, A_n)$ can be done greedily:
  - Repeatedly cut the leftmost or rightmost A_i, whichever is smaller.
- The cost is $|A| - \max |A_i|$.
- It can easily be shown to be optimal.

Break, Merge and Sort

- *Greedy insight 4:* Merging can also be done greedily, by merging the two shortest existing arrays at a time.
- *Proof:*
  - If A_i is merged c_i times, then it contributes $c_i|A_i|$ to the total cost.
  - Consider the "merging tree", where the leaves are A_i and the internal nodes represent merges. Then c_i is the depth of A_i.
  - Note that we can rearrange the leaves A_i however we want.
  - To minimize the cost, we want to sort the leaves A_i so that the shortest ones are the deepest, and vice versa.
  - We can also make the two shortest ones siblings.

Break, Merge and Sort

- *Proof:*
  - …
  - We can make the two shortest ones siblings ⇒ there is an optimal solution where the two shortest arrays are merged.
  - Repeat the argument until all arrays are merged.
  - Therefore, greedy merging works.
- The proof is basically the same as for Huffman coding.
- We can use a **priority queue** to get the two shortest arrays quickly.
- Overall complexity is **O(s log s)** where s is the sum of the lengths of all arrays.

Problem 4
Minimum Weight Bi-Partition

No. of accepted solutions: 255

Author: Vichitr, Prepared by: Ajinkya Parab

Minimum Weight Bi-Partition

- Observe that a tree is a connected bipartite graph.
- Hence, after adding new edges, finding an MST would be enough.

Minimum Weight Bi-Partition

- *Greedy insight*: After sorting nodes w.r.t. A[i], adding edges between **consecutive nodes** is enough as those would give n-1 edges having the smallest sum of |A[u] - A[v]|.
- We can form an MST out of these edges and the initial edges that have weight 0.
- Complexity: **O((n + m) log n)**

Problem 5
Strings and LIS

No. of accepted solutions: 56

Author: Divyansh Verma

Strings and LIS

- Notice that we can switch characters at most 25 times since there are only 26 lowercase letters.
- So almost all the time, we'll repeat just a single letter.
  - to be precise, at least q - 25 times
- It is enough to choose only one letter to repeat.
- Let x be that letter, $0 \le x \le 25$.
- We also choose the word w with the most occurrences of x.
- Then w will appear at least q - 25 times.
- Let maxocc[x] be the number of times x appears in w.

Strings and LIS

- The remaining few words will either be before w or after w.
- Let there be L words before w, and R after.
  - Then $L + R \le 25$.
  - And also $L + R \le q$.
- For $0 \le i \le j \le 25$ and $c \le 25$, let longest[c][i][j] be the longest subsequence we can form from c words with letters in i...j.
- Then the answer is the largest value of
  - longest[L][0][x] + maxocc[x] · (q - L - R) + longest[R][x][25]

Strings and LIS

- longest[c][i][j] can be computed with DP.
- longest[c][i][j] = $\max_k$ longest[c - 1][i][k] + longest[1][k][j] for i ≤ k ≤ j where longest[1][i][j] = longest subsequence we can form from a **single** word with letters in i...j.
- longest[1][i][j] is can be computed individually per word w in $O(|w|\alpha^2)$ time where α = alphabet size = 26.
- Overall complexity: $\mathbf{O(s\alpha^2 + \alpha^4)}$ where s is the sum of the lengths of all words.

Problem 6
Tree Max Sum

No. of accepted solutions: 0 

Author: Udit Sanghi

Tree Max Sum

- Root the tree, then DP on the tree.
- For each node i, compute an $h \times k$ table dp[i][H][K] = the largest sum if we take at most K nodes, all of whose depths are at least H.
- We can merge the dp of the children in $O(h^2k^2 \times \text{no. of children})$ per node
- Too slow: Takes $O(nh^2k^2)$ overall.
- The worst of the worst cases is then $\mathbf{O(n^5)}$ 😭

Tree Max Sum

- If we shrink the h×k table at node i into an h[i]×k[i] table where
  h[i] = min(h, height(i)),
  k[i] = min(k, size(i)),
  then after analysis, the running time improves to O(nhk min(h,k)).
- **O(n^4)** is still pretty bad though 😢
- O(n^4) can also be achieved in a few other ways.

Tree Max Sum

```
void dfs(int u,int p){
    for(auto v:adj[u]){
        if(v == p) continue;
        dfs(v,u);
    }
    FOR(i,1,k+1) dp[u][0][i] = a[u];
    int cur = 1;
    for(auto v:adj[u]){
        if(v == p) continue;
        for(int i = 0; i <= h; i ++){
            for(int j = 0; j <= k; j ++) tmp[i][j] = dp[u][i][j];
        }
        cur += sz[v];
        for(int i = 0; i <= h; i ++){ // can also do i <= min(h,height)
            for(int j = 0; j <= k; j ++){ // can also do j <= min(k,cur)
                // this next loop can be avoided by only considering these 2 values:
                // iv = max(i-1,h-1-i)
                // iv = max(0,h-1-i)
                for(int iv = max(0,h-1-i); iv <= h; iv ++){
                    for(int jv = 0; jv <= k-j; jv ++){//can also do jv <= min(j,j,sz[v])
                        remax(dp[u][min(i,iv+1)][j+jv],tmp[i][j]+dp[v][iv][jv]);
                    }
                }
            }
        }
        REP(j,k+1){
            FORD(i,h,0){
                if(i < h) remax(dp[u][i][j],dp[u][i+1][j]);
                if(j > 0) remax(dp[u][i][j],dp[u][i][j-1]);
            }
        }
    }
    sz[u] = cur;
}
```

Tree Max Sum

- *Insight 1:* If h is large, then we can't take too many nodes.
- In fact, we can show that we can take at most $\lceil 2n/h \rceil$ nodes.
  - The worst case is a star-like tree.
    (Chains of length h/2 connected to the center)
- Thus, we can replace k with $\min(k, \lceil 2n/h \rceil)$.
  - i.e., we can assume that $hk = O(n)$.
- Running time improves: if $hk = O(n)$, then $\min(h,k) = O(n^{0.5})$,
  - so $O(nhk \min(h,k))$ becomes $\mathbf{O(n^{2.5})}$ 🙃
  - Much better, but still a bit slow.

Tree Max Sum

- *Proof of insight 1:*
- Consider a node u that is taken. Mark that node and the nodes with a distance of $< h/2$ to it.
- Observe that, at each step, you are marking at least $\lceil h/2 \rceil$ nodes, and no node is marked twice.
  - except if we take only 1 node
- If we take $t > 1$ nodes, then we mark at least $t \cdot \lceil h/2 \rceil$ nodes.
  - So $t \cdot \lceil h/2 \rceil \leq n \Rightarrow t \leq n / \lceil h/2 \rceil \leq 2n/h$.
- Thus, we can only take at most $2n/h$ nodes (or 1 node).

Tree Max Sum

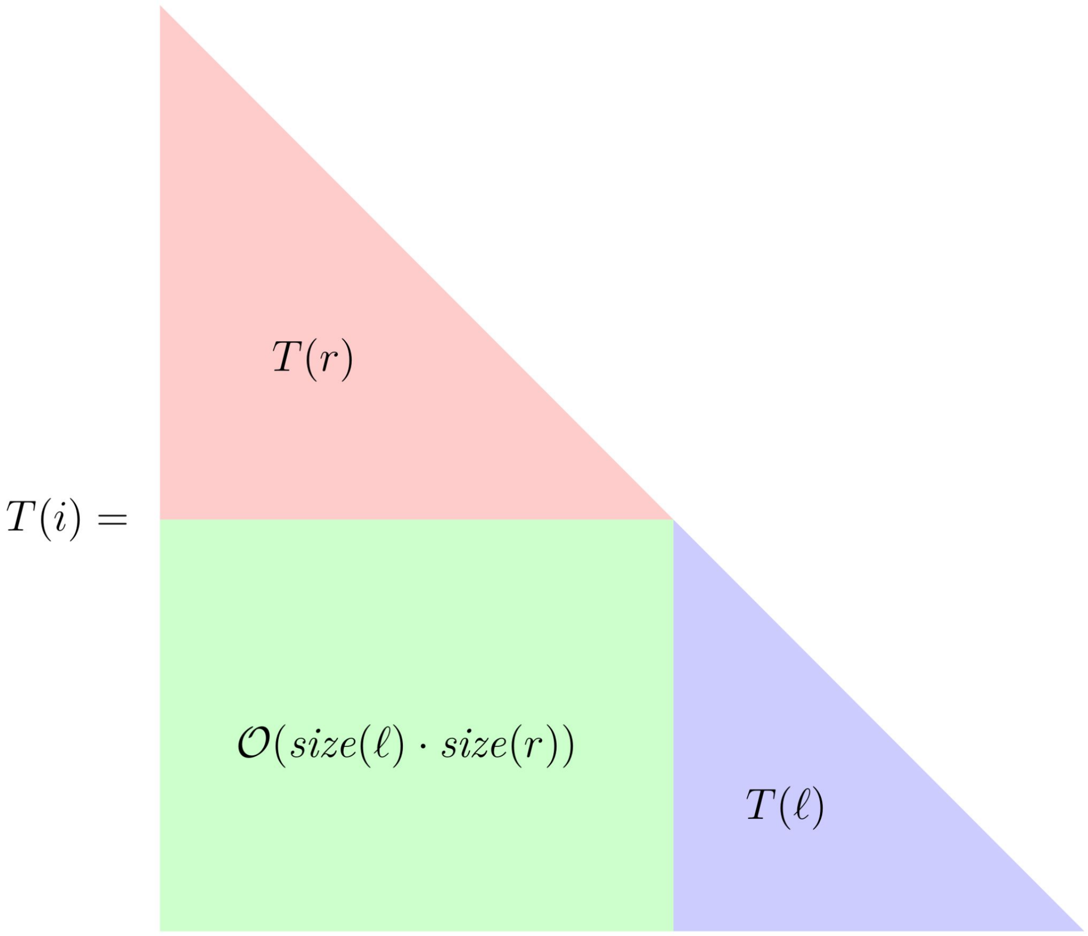
- *Insight 2:* For each node i, only create an $h[i] \times k[i]$ table, where
  - $h[i] = \min(h, \text{height}(i))$
  - $k[i] = \min(k, \lceil 2 \cdot \text{size}(i)/h \rceil)$
- Now, each table will only have a size of $O(\text{size}(i))$.
- Processing per node i becomes $O(\text{size}(L) \cdot \text{size}(R))$, where L and R are the children of i. The running time recurrence becomes

$$T(i) = T(L) + T(R) + O(\text{size}(L) \cdot \text{size}(R))$$

- Well-known recurrence with solution $\mathbf{O(n^2)}$ 😄
- The recurrence generalizes easily to non-binary trees.

Tree Max Sum

- The proof that it is $O(n^2)$ can be extracted from the image on the right.



Tree Max Sum

```cpp
void dfs(int u,int p){
    for(auto v:adj[u]){
        if(v == p) continue;
        dfs(v,u);
    }
    FOR(i,1,k+1) dp[u][0][i] = a[u];
    int cur = 1;
    for(auto v:adj[u]){
        if(v == p) continue;
        for(int i = 0; i <= h; i ++){
            for(int j = 0; j <= k; j ++) tmp[i][j] = dp[u][i][j];
        }
        int ku = getmaxk(cur);
        int kv = getmaxk(sz[v]);
        cur += sz[v];
        for(int i = 0; i <= h; i ++){
            for(int j = 0; j <= ku; j ++){
                int iv = max(i-1,h-1-i);
                for(int jv = 0; jv <= min(k-j,kv); jv ++){
                    remax(dp[u][i][j+jv],tmp[i][j]+dp[v][iv][jv]);
                }
                iv = max(0,h-1-i);
                if(iv >= i) continue;
                for(int jv = 0; jv <= min(k-j,kv); jv ++){
                    remax(dp[u][iv+1][j+jv],tmp[i][j]+dp[v][iv][jv]);
                }
            }
        }
        REP(j,k+1){
            FORD(i,h,0){
                if(i < h) remax(dp[u][i][j],dp[u][i+1][j]);
                if(j > 0) remax(dp[u][i][j],dp[u][i][j-1]);
            }
        }
    }
    sz[u] = cur;
}
```