CERC 2019

Presentation of solutions

December 3, 2019

ABB [ABBA]

ABB [ABBA]

- Task: Find the minimum number of letters to be appended to make the string palindromic.

ABB [ABBA]

- Task: Find the minimum number of letters to be appended to make the string palindromic.
- Observation: We want to find the longest palindrome among all suffixes of the string and append the reversed prefix to the end of the string.

ABB [ABBA]

- Task: Find the minimum number of letters to be appended to make the string palindromic.

- Observation: We want to find the longest palindrome among all suffixes of the string and append the reversed prefix to the end of the string.
- We can iterate over all suffixes and check whether it is palindromic.
- The check could be done by multiple string matching algorithms.
- Examples: Hash, Z-function, Suffix Arrays, Manacher, Palindromic Trees, ...

ABB [ABBA]

- Task: Find the minimum number of letters to be appended to make the string palindromic.

- Observation: We want to find the longest palindrome among all suffixes of the string and append the reversed prefix to the end of the string.
- We can iterate over all suffixes and check whether it is palindromic.
- The check could be done by multiple string matching algorithms.
- Examples: Hash, Z-function, Suffix Arrays, Manacher, Palindromic Trees, ...
- Complexity depends on chosen algorithm: $\mathcal{O}(N)$ is achievable.

Zeldain Garden [Linkin Park]

Zeldain Garden [Linkin Park]

- Task: Find the sum of divisor functions on a range.

Zeldain Garden [Linkin Park]

- Task: Find the sum of divisor functions on a range.
- Observation: The task can be splitted into two same tasks: Range(left,right) is equal to Range(1,right)-Range(1,left-1)

Zeldain Garden [Linkin Park]

- Task: Find the sum of divisor functions on a range.
- Observation: The task can be splitted into two same tasks: Range(left,right) is equal to Range(1,right)-Range(1,left-1)
- Let Q be equal to $\lfloor N \rfloor$
- All numbers "lesser" than sqrt: $\sum_{i=1}^{Q} \lfloor \frac{N}{i} \rfloor$
- All numbers "greater" than sqrt: $\sum_{i=1}^{Q} \lfloor \frac{N}{i} \rfloor$

Zeldain Garden [Linkin Park]

- Task: Find the sum of divisor functions on a range.

- Observation: The task can be splitted into two same tasks: Range(left,right) is equal to Range(1,right)-Range(1,left-1)
- Let Q be equal to $\lfloor N \rfloor$
- All numbers "lesser" than sqrt: $\sum_{i=1}^{Q} \lfloor \frac{N}{i} \rfloor$
- All numbers "greater" than sqrt: $\sum_{i=1}^{Q} \lfloor \frac{N}{i} \rfloor$
- Almost there, where is the mistake?

Zeldain Garden [Linkin Park]

- Task: Find the sum of divisor functions on a range.

- Observation: The task can be splitted into two same tasks: Range(left,right) is equal to Range(1,right)-Range(1,left-1)
- Let Q be equal to $\lfloor N \rfloor$
- All numbers "lesser" than sqrt: $\sum_{i=1}^{Q} \lfloor \frac{N}{i} \rfloor$
- All numbers "greater" than sqrt: $\sum_{i=1}^{Q} \lfloor \frac{N}{i} \rfloor$
- Almost there, where is the mistake?
- We included all pairs lesser/equal to Q twice.

Zeldain Garden [Linkin Park]

- Task: Find the sum of divisor functions on a range.
- Observation: The task can be splitted into two same tasks: Range(left,right) is equal to Range(1,right)-Range(1,left-1)
- Let Q be equal to $\lfloor N \rfloor$
- All numbers "lesser" than sqrt: $\sum_{i=1}^{Q} \lfloor \frac{N}{i} \rfloor$
- All numbers "greater" than sqrt: $\sum_{i=1}^{Q} \lfloor \frac{N}{i} \rfloor$
- Almost there, where is the mistake?
- We included all pairs lesser/equal to Q twice.
- $(2 \cdot \sum_{i=1}^{Q} \lfloor \frac{N}{i} \rfloor)$-$Q^2$

Zeldain Garden [Linkin Park]

- Task: Find the sum of divisor functions on a range.
- Observation: The task can be splitted into two same tasks: Range(left,right) is equal to Range(1,right)-Range(1,left-1)
- Let Q be equal to $\lfloor N \rfloor$
- All numbers "lesser" than sqrt: $\sum_{i=1}^{Q} \lfloor \frac{N}{i} \rfloor$
- All numbers "greater" than sqrt: $\sum_{i=1}^{Q} \lfloor \frac{N}{i} \rfloor$
- Almost there, where is the mistake?
- We included all pairs lesser/equal to Q twice.
- $(2 \cdot \sum_{i=1}^{Q} \lfloor \frac{N}{i} \rfloor)$-$Q^2$
- Complexity is $\mathcal{O}(\sqrt{N})$

The Bugs [The Beatles]

The Bugs [The Beatles]

- Task: Find the number of normalised 123 patterns in sequence.

The Bugs [The Beatles]

- Task: Find the number of normalised 123 patterns in sequence.
- There are 27 patterns which are not normalised (3^3).

The Bugs [The Beatles]

- Task: Find the number of normalised 123 patterns in sequence.

- There are 27 patterns which are not normalised (3^3).
- BUT: There are only 13 of patterns after normalisation: 111,112,121,122,123,132,211,212,213,221,231,312,321

The Bugs [The Beatles]

- Task: Find the number of normalised 123 patterns in sequence.

- There are 27 patterns which are not normalised (3^3).
- BUT: There are only 13 of patterns after normalisation: 111,112,121,122,123,132,211,212,213,221,231,312,321
- BUT: There are nonly 5 patterns which are not isomorphic after **reverse** and ∞-value: 111,112,121,123,132

The Bugs [The Beatles]

- Task: Find the number of normalised 123 patterns in sequence.

- There are 27 patterns which are not normalised (3^3).
- BUT: There are only 13 of patterns after normalisation: 111,112,121,122,123,132,211,212,213,221,231,312,321
- BUT: There are nonly 5 patterns which are not isomorphic after **reverse** and ∞-value: 111,112,121,123,132
- **111**: Simple frequence array.

The Bugs [The Beatles]

- Task: Find the number of normalised 123 patterns in sequence.
- There are 27 patterns which are not normalised (3^3).
- BUT: There are only 13 of patterns after normalisation: 111,112,121,122,123,132,211,212,213,221,231,312,321
- BUT: There are nonly 5 patterns which are not isomorphic after **reverse** and ∞-value: 111,112,121,123,132
- **111**: Simple frequence array.
- **123**: Fix the **2** and check prefix minimum and suffix maximum.

The Bugs [The Beatles]

- Task: Find the number of normalised 123 patterns in sequence.

- There are 27 patterns which are not normalised (3^3).
- BUT: There are only 13 of patterns after normalisation: 111,112,121,122,123,132,211,212,213,221,231,312,321
- BUT: There are nonly 5 patterns which are not isomorphic after **reverse** and ∞-value: 111,112,121,123,132
- **111**: Simple frequence array.
- **123**: Fix the **2** and check prefix minimum and suffix maximum.
- **112**: Combination of frequence array and suffix maximum.

The Bugs [The Beatles]

- Task: Find the number of normalised 123 patterns in sequence.

- There are 27 patterns which are not normalised (3^3).
- BUT: There are only 13 of patterns after normalisation: 111,112,121,122,123,132,211,212,213,221,231,312,321
- BUT: There are nonly 5 patterns which are not isomorphic after **reverse** and ∞-value: 111,112,121,123,132
- **111**: Simple frequence array.
- **123**: Fix the **2** and check prefix minimum and suffix maximum.
- **112**: Combination of frequence array and suffix maximum.
- **121**: Check maximum between first and last occurence of each value.

The Bugs [The Beatles]

- Task: Find the number of normalised 123 patterns in sequence.

- There are 27 patterns which are not normalised (3^3).
- BUT: There are only 13 of patterns after normalisation: 111,112,121,122,123,132,211,212,213,221,231,312,321
- BUT: There are nonly 5 patterns which are not isomorphic after **reverse** and ∞-value: 111,112,121,123,132
- **111**: Simple frequence array.
- **123**: Fix the **2** and check prefix minimum and suffix maximum.
- **112**: Combination of frequence array and suffix maximum.
- **121**: Check maximum between first and last occurence of each value.
- **132**: Fix the **3** and do combination of prefix minimum and biggest lesser value in suffix.

The Bugs [The Beatles]

- Task: Find the number of normalised 123 patterns in sequence.
- There are 27 patterns which are not normalised (3^3).
- BUT: There are only 13 of patterns after normalisation: 111,112,121,122,123,132,211,212,213,221,231,312,321
- BUT: There are nonly 5 patterns which are not isomorphic after **reverse** and ∞-value: 111,112,121,123,132
- **111**: Simple frequence array.
- **123**: Fix the **2** and check prefix minimum and suffix maximum.
- **112**: Combination of frequence array and suffix maximum.
- **121**: Check maximum between first and last occurence of each value.
- **132**: Fix the **3** and do combination of prefix minimum and biggest lesser value in suffix.
- Complexity is $\mathcal{O}(N \log N)$

Be Geeks [Bee Gees]

Be Geeks [Bee Gees]

- Task: Sum the products of *GCD* and *max* for each subarray.

Be Geeks [Bee Gees]

- Task: Sum the products of *GCD* and *max* for each subarray.

- Lets Divide and Conquer the array by *maximum*.
- This could be efficiently done by some data structure: For example **Segment Tree** or **Sparse Table**.
- Now we can proccess all subarrays and multiply them by the *maximum*.

Be Geeks [Bee Gees]

- Task: Sum the products of *GCD* and *max* for each subarray.
- Lets Divide and Conquer the array by *maximum*.
- This could be efficiently done by some data structure: For example **Segment Tree** or **Sparse Table**.
- Now we can proccess all subarrays and multiply them by the *maximum*.
- Now we have to find all blocks with the same *GCD* to left and to right: Then we can count the answer from all combinations.

Be Geeks [Bee Gees]

- Task: Sum the products of *GCD* and *max* for each subarray.

- Lets Divide and Conquer the array by *maximum*.
- This could be efficiently done by some data structure: For example **Segment Tree** or **Sparse Table**.
- Now we can proccess all subarrays and multiply them by the *maximum*.
- Now we have to find all blocks with the same *GCD* to left and to right: Then we can count the answer from all combinations.
- Observation: There are at most $\mathcal{O}(\log MAX)$ such blocks.

Be Geeks [Bee Gees]

- Task: Sum the products of *GCD* and *max* for each subarray.

- Lets Divide and Conquer the array by *maximum*.
- This could be efficiently done by some data structure: For example **Segment Tree** or **Sparse Table**.
- Now we can proccess all subarrays and multiply them by the *maximum*.
- Now we have to find all blocks with the same *GCD* to left and to right: Then we can count the answer from all combinations.
- Observation: There are at most $\mathcal{O}(\log MAX)$ such blocks.
- There are multiple methods to find the blocks – for example:
  - Combination of previously stated Data Structure and Binary search.
  - Factorisation and searching for next.

Be Geeks [Bee Gees]

- Task: Sum the products of *GCD* and *max* for each subarray.

- Lets Divide and Conquer the array by *maximum*.
- This could be efficiently done by some data structure: For example **Segment Tree** or **Sparse Table**.
- Now we can proccess all subarrays and multiply them by the *maximum*.
- Now we have to find all blocks with the same *GCD* to left and to right: Then we can count the answer from all combinations.
- Observation: There are at most $\mathcal{O}(\log MAX)$ such blocks.
- There are multiple methods to find the blocks – for example:
  - Combination of previously stated Data Structure and Binary search.
  - Factorisation and searching for next.
- Complexity is $\mathcal{O}(N \log^3 MAX)$

K==S [Kiss]

K==S [Kiss]

- Task: Find the number of strings of length N which do not contain any of given strings as a substring.

K==S [Kiss]

- Observation: It can also be solved by computing the number of strings containing any of them as a substring. Then the solution is $26^N - ANS$.

K==S [Kiss]

- Observation: It can also be solved by computing the number of strings containing any of them as a substring. Then the solution is $26^N - ANS$.
- We have to find out the way to build all such strings.

K==S [Kiss]

- Observation: It can also be solved by computing the number of strings containing any of them as a substring. Then the solution is $26^N - ANS$.
- We have to find out the way to build all such strings.
- This can be done by constructing the Aho-Corasick automaton.
- It wasn't needed to construct this automaton efficiently as there are at most $Q = 100$ characters involved (there will be $\approx Q$ states in the automaton).

K==S [Kiss]

- Then we figure out approach to solve the problem, though for much lower limit of N.
- We will use Dynamic Programming where the state configuration is combination of length of the string and the state of the automation that we're currently in.
- Time complexity of this approach is $\mathcal{O}(N \cdot Q)$.
- Too slow to fit the constraints of the problem $\cdots$.

K==S [Kiss]

- Finally we need to optimize DP approach to fit in the time limit.
- We will do so by using Matrix Exponentiation.
- Similarly to the DP approach we will create the transition matrix and use its N-th power to compute the answer.
- Time complexity of this approach is $\mathcal{O}(L^3 \log N)$.

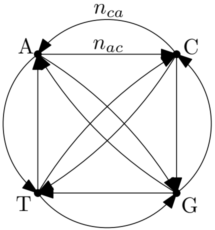
Ponk Warshall [Pink Floyd]

Ponk Warshall [Pink Floyd]

- Task: Compose a reordering of a string on four letters from fewest possible swaps.

Ponk Warshall [Pink Floyd]

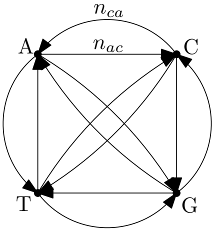
- Task: Compose a reordering of a string on four letters from fewest possible swaps.
- Reformulation:
  - Directed graph on 4 vertices, edges with multiplicities.



  - Decompose the graph into maximum possible number of cycles.

Ponk Warshall [Pink Floyd]

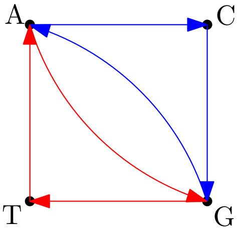
- Task: Compose a reordering of a string on four letters from fewest possible swaps.
- Reformulation:
  - Directed graph on 4 vertices, edges with multiplicities.



  - Decompose the graph into maximum possible number of cycles.
- Greedily start with 2-cycles, then 3-cycles and cover the rest with 4-cycles.
- Complexity is $\mathcal{O}(N)$.

Ponk Warshall 2-cycles

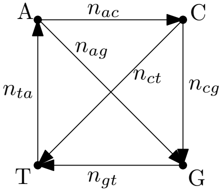
- Assume the 2-cycle is covered differently.



- Then we can change the covering without decreasing the number of cycles.

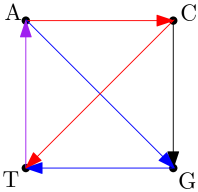
Ponk Warshall 3-cycles

- After covering 2-cycles, only one case remains



(up to isomorphism)

- Only two directed 3-cycles (with one shared edge).



Light Emitting Hindenburg [Led Zeppelin]

Light Emitting Hindenburg [Led Zeppelin]

- Task: Find the maximum binary *AND* which could be obtained as sequence of K numbers.

Light Emitting Hindenburg [Led Zeppelin]

- Task: Find the maximum binary *AND* which could be obtained as sequence of K numbers.
- Observation: We can never get bigger answer from lesser bits than we would get if we achieve to persist a higher bit.

Light Emitting Hindenburg [Led Zeppelin]

- Task: Find the maximum binary *AND* which could be obtained as sequence of K numbers.

- Observation: We can never get bigger answer from lesser bits than we would get if we achieve to persist a higher bit.
- This means we can greedily iterate from most significant bits and check, whether there are at least K of them in numbers we have:
- In case of YES: We reduce the set of numbers to those with this bit on and OR 2^i with the answer.
- In case of NO: We simply go to another bit.

Light Emitting Hindenburg [Led Zeppelin]

- Task: Find the maximum binary AND which could be obtained as sequence of K numbers.

- Observation: We can never get bigger answer from lesser bits than we would get if we achieve to persist a higher bit.
- This means we can greedily iterate from most significant bits and check, whether there are at least K of them in numbers we have:
- In case of YES: We reduce the set of numbers to those with this bit on and OR 2^i with the answer.
- In case of NO: We simply go to another bit.
- Complexity is $\mathcal{O}(N \log MAX)$

Crimson Sexy Jalapeños [Red Hot Chilli Peppers]

Crimson Sexy Jalapeños [Red Hot Chilli Peppers]

- Task: Win a game of chockolate-eating against "AI".

Crimson Sexy Jalapeños [Red Hot Chilli Peppers]

- Task: Win a game of chockolate-eating against "AI".
- Observation: We can divide the game into four "independent" games – as for each side – played at once.

Crimson Sexy Jalapeños [Red Hot Chilli Peppers]

- Task: Win a game of chockolate-eating against "AI".
- Observation: We can divide the game into four "independent" games – as for each side – played at once.
- It can be transformed into a NIM game with four piles (while subtracting any number from a pile).

Crimson Sexy Jalapeños [Red Hot Chilli Peppers]

- Task: Win a game of chockolate-eating against "AI".

- Observation: We can divide the game into four "independent" games – as for each side – played at once.
- It can be transformed into a NIM game with four piles (while subtracting any number from a pile).
- The winability of game can be decided by XORing the sizes of state while.
- If the XOR is equal to 0 the state is lost.

Crimson Sexy Jalapeños [Red Hot Chilli Peppers]

- Task: Win a game of chockolate-eating against "AI".

- Observation: We can divide the game into four "independent" games – as for each side – played at once.
- It can be transformed into a NIM game with four piles (while subtracting any number from a pile).
- The winability of game can be decided by XORing the sizes of state while.
- If the XOR is equal to 0 the state is lost.
- Similary we can decide the next move:
- We have to take a number from a pile, such that after taking the number, the XOR will be 0.
- NOTE: It can be proven that it is always possible to find such move.

Crimson Sexy Jalapeños [Red Hot Chilli Peppers]

- Task: Win a game of chockolate-eating against "AI".

- Observation: We can divide the game into four "independent" games – as for each side – played at once.
- It can be transformed into a NIM game with four piles (while subtracting any number from a pile).
- The winability of game can be decided by XORing the sizes of state while.
- If the XOR is equal to 0 the state is lost.
- Similary we can decide the next move:
- We have to take a number from a pile, such that after taking the number, the XOR will be 0.
- NOTE: It can be proven that it is always possible to find such move.
- Complexity is $\mathcal{O}(N)$ (as the game could consist of $\mathcal{O}(N)$ moves.

Bob in Wonderlands [Alice in Chains]

Bob in Wonderlands [Alice in Chains]

- Task: Minimal number of edge reconnections to make a tree become a path.

Bob in Wonderlands [Alice in Chains]

- Task: Minimal number of edge reconnections to make a tree become a path.
- Lower bound: We have to get rid of all degrees larger than 2.

Bob in Wonderlands [Alice in Chains]

- Task: Minimal number of edge reconnections to make a tree become a path.
- Lower bound: We have to get rid of all degrees larger than 2.
- Procedure: Disconnect every such an edge and connect it to some leaf (it always exists).

Bob in Wonderlands [Alice in Chains]

- Task: Minimal number of edge reconnections to make a tree become a path.
- Lower bound: We have to get rid of all degrees larger than 2.
- Procedure: Disconnect every such an edge and connect it to some leaf (it always exists).
- Obviously we don't have to simulate the procedure!

Bob in Wonderlands [Alice in Chains]

- Task: Minimal number of edge reconnections to make a tree become a path.
- Lower bound: We have to get rid of all degrees larger than 2.
- Procedure: Disconnect every such an edge and connect it to some leaf (it always exists).
- Obviously we don't have to simulate the procedure!
- Simply count the number of reconnections (sum of all degrees higher than 2 minus 2).

Bob in Wonderlands [Alice in Chains]

- Task: Minimal number of edge reconnections to make a tree become a path.

- Lower bound: We have to get rid of all degrees larger than 2.
- Procedure: Disconnect every such an edge and connect it to some leaf (it always exists).
- Obviously we don't have to simulate the procedure!
- Simply count the number of reconnections (sum of all degrees higher than 2 minus 2).
- Complexity is $\mathcal{O}(N)$

Saba1000kg [Sabaton]

Saba1000kg [Sabaton]

- Task: For each induced subgraph count the number of components.

Saba1000kg [Sabaton]

- Task: For each induced subgraph count the number of components.
- Observation (for $N = M = Q = 10^5$): The total number of edges among all subgraphs is at most $\mathcal{O}(N\sqrt{N})$

Saba1000kg [Sabaton]

- Task: For each induced subgraph count the number of components.

- Observation (for $N = M = Q = 10^5$): The total number of edges among all subgraphs is at most $\mathcal{O}(N\sqrt{N})$
- Proof: Worst case if **clique** in each query. Note that we can't make clique with more than $\mathcal{O}(N)$ edges (by taking M as $\sqrt{N}$). There are at most $\sqrt{N}$ such queries.

Saba1000kg [Sabaton]

- Task: For each induced subgraph count the number of components.
- Observation (for $N = M = Q = 10^5$): The total number of edges among all subgraphs is at most $\mathcal{O}(N\sqrt{N})$
- Proof: Worst case if **clique** in each query. Note that we can't make clique with more than $\mathcal{O}(N)$ edges (by taking M as $\sqrt{N}$). There are at most $\sqrt{N}$ such queries.
- To build the graph we have to consider two cases:
  1. For each query of size M greater than $\sqrt{N}$, go through all edges and check, whether they are in the given set.
     # of checked edges: $\frac{N}{M}N < \frac{N}{\sqrt{N}}N < N\sqrt{N}$
  2. For each query of size M lesser/equal than $\sqrt{N}$, check for all pairs of vertices, whether there is an edge connecting them.
     # of checked edges: $\frac{N}{M}M^2 = NM \leq N\sqrt{N}$

Saba1000kg [Sabaton]

- Task: For each induced subgraph count the number of components.

- Observation (for $N = M = Q = 10^5$): The total number of edges among all subgraphs is at most $\mathcal{O}(N\sqrt{N})$
- Proof: Worst case if **clique** in each query. Note that we can't make clique with more than $\mathcal{O}(N)$ edges (by taking M as $\sqrt{N}$). There are at most $\sqrt{N}$ such queries.
- To build the graph we have to consider two cases:
  1. For each query of size M greater than $\sqrt{N}$, go through all edges and check, whether they are in the given set.
     # of checked edges: $\frac{N}{M}N < \frac{N}{\sqrt{N}}N < N\sqrt{N}$
  2. For each query of size M lesser/equal than $\sqrt{N}$, check for all pairs of vertices, whether there is an edge connecting them.
     # of checked edges: $\frac{N}{M}M^2 = NM \leq N\sqrt{N}$
- The checking step can be done (for example) with $\mathcal{O}(\log N)$ overhead if we use some standard set implementation.

Saba1000kg [Sabaton]

- Whenever the graph is built, the number of components is easily obtained in linear time by a couple of DFS calls.

Saba1000kg [Sabaton]

- Whenever the graph is built, the number of components is easily obtained in linear time by a couple of DFS calls.
- Complexity is $\mathcal{O}(N\sqrt{N}\log N)$

Deep800080 [Deep Purple]

Deep800080 [Deep Purple]

- Task: Sum the products of *GCD* and *max* for each subarray.

Deep800080 [Deep Purple]

- Task: Sum the products of *GCD* and *max* for each subarray.

- Lets Divide and Conquer the array by *maximum*.
- This could be efficiently done by some data structure: For example **Segment Tree** or **Sparse Table**.
- Now we can proccess all subarrays and multiply them by the *maximum*.

Deep800080 [Deep Purple]

- Task: Sum the products of *GCD* and *max* for each subarray.

- Lets Divide and Conquer the array by *maximum*.
- This could be efficiently done by some data structure: For example **Segment Tree** or **Sparse Table**.
- Now we can proccess all subarrays and multiply them by the *maximum*.
- Now we have to find all blocks with the same *GCD* to left and to right: Then we can count the answer from all combinations.

Deep800080 [Deep Purple]

- Task: Sum the products of *GCD* and *max* for each subarray.

- Lets Divide and Conquer the array by *maximum*.
- This could be efficiently done by some data structure: For example **Segment Tree** or **Sparse Table**.
- Now we can proccess all subarrays and multiply them by the *maximum*.
- Now we have to find all blocks with the same *GCD* to left and to right: Then we can count the answer from all combinations.
- Observation: There are at most $\mathcal{O}(\log MAX)$ such blocks.

Deep800080 [Deep Purple]

- Task: Sum the products of *GCD* and *max* for each subarray.

- Lets Divide and Conquer the array by *maximum*.
- This could be efficiently done by some data structure: For example **Segment Tree** or **Sparse Table**.
- Now we can proccess all subarrays and multiply them by the *maximum*.
- Now we have to find all blocks with the same *GCD* to left and to right: Then we can count the answer from all combinations.
- Observation: There are at most $\mathcal{O}(\log MAX)$ such blocks.
- There are multiple methods to find the blocks – for example:
  - Combination of previously stated Data Structure and Binary search.
  - Factorisation and searching for next.

Deep800080 [Deep Purple]

- Task: Sum the products of *GCD* and *max* for each subarray.

- Lets Divide and Conquer the array by *maximum*.
- This could be efficiently done by some data structure: For example **Segment Tree** or **Sparse Table**.
- Now we can proccess all subarrays and multiply them by the *maximum*.
- Now we have to find all blocks with the same *GCD* to left and to right: Then we can count the answer from all combinations.
- Observation: There are at most $\mathcal{O}(\log MAX)$ such blocks.
- There are multiple methods to find the blocks – for example:
  - Combination of previously stated Data Structure and Binary search.
  - Factorisation and searching for next.
- Complexity is $\mathcal{O}(N \log^3 MAX)$

Screamers in the Storm [by Emerald Sun]

Screamers in the Storm [by Emerald Sun]

Thank you for your attention!