BRACU CP Workshop

Day 5

Mazed Hossain Parag

Ahnaf Shahriar Asif

BRAC University, Dhaka

23 April 2025

Table of Contents

What is STL?

- STL = Standard Template Library

What is STL?

- STL = Standard Template Library
- Part of the C++ Standard Library

What is STL?

- STL = Standard Template Library
- Part of the C++ Standard Library
- set, multiset, map, priority queue, dequeue

Set

- Stores unique, sorted elements

Set

- Stores unique, sorted elements
- Member functions:
  - s.insert(x)
  - s.erase(x)

Set

- Stores unique, sorted elements
- Member functions:
  - s.insert(x)
  - s.erase(x)
  - s.find(x)

- Time complexity: $\mathcal{O}(\log N)$ for insert, erase, search

Set

- Stores unique, sorted elements
- Member functions:
  - s.insert(x)
  - s.erase(x)
  - s.find(x)
  - s.lower_bound(x)
- Time complexity: $\mathcal{O}(\log N)$ for insert, erase, search

Set

- Stores unique, sorted elements
- Member functions:
  - s.insert(x)
  - s.erase(x)
  - s.find(x)
  - s.lower_bound(x)
  - s.upper_bound(x)
- Time complexity: $\mathcal{O}(\log N)$ for insert, erase, search

Multiset

- Duplicate allowed, sorted elements

Multiset

- Duplicate allowed, sorted elements
- Member functions:
  - ms.insert(x)
  - ms.erase(x)

Multiset

- Duplicate allowed, sorted elements
- Member functions:
  - ms.insert(x)
  - ms.erase(x)
  - ms.find(x)

- Time complexity: $\mathcal{O}(\log N)$ for insert, erase, search

Multiset

- Duplicate allowed, sorted elements
- Member functions:
  - ms.insert(x)
  - ms.erase(x)
  - ms.find(x)
  - ms.lower_bound(x)
- Time complexity: $\mathcal{O}(\log N)$ for insert, erase, search

Multiset

- Duplicate allowed, sorted elements
- Member functions:
  - ms.insert(x)
  - ms.erase(x)
  - ms.find(x)
  - ms.lower_bound(x)
  - ms.upper_bound(x)
- Time complexity: $\mathcal{O}(\log N)$ for insert, erase, search

Map

- Stores key-value pairs, keys are unique and sorted

Map

- Stores key-value pairs, keys are unique and sorted
- Member functions:
  - mp[key] = value
  - mp.erase(key)

Map

- Stores key-value pairs, keys are unique and sorted
- Member functions:
  - mp[key] = value
  - mp.erase(key)
  - mp.find(key)
- Time complexity: $\mathcal{O}(\log N)$ for insert, access, and erase

Map

- Stores key-value pairs, keys are unique and sorted
- Member functions:
  - mp[key] = value
  - mp.erase(key)
  - mp.find(key)
  - mp.lower_bound(key)
- Time complexity: $\mathcal{O}(\log N)$ for insert, access, and erase

Map

- Stores key-value pairs, keys are unique and sorted
- Member functions:
  - mp[key] = value
  - mp.erase(key)
  - mp.find(key)
  - mp.lower_bound(key)
  - mp.upper_bound(key)
- Time complexity: $\mathcal{O}(\log N)$ for insert, access, and erase

Priority queue

- Max-heap by default (top element is largest)

Priority queue

- Max-heap by default (top element is largest)
- For min-heap:

```
priority_queue<int, vector<int>, greater<int>> min_pq;
```

Priority queue

- Max-heap by default (top element is largest)
- For min-heap:

```
priority_queue<int, vector<int>, greater<int>> min_pq;
```

- Member functions:
  - pq.push(value)
  - pq.pop()
  - pq.top()

Priority queue

- Max-heap by default (top element is largest)
- For min-heap:

```
priority_queue<int, vector<int>, greater<int>> min_pq;
```

- Member functions:
  - pq.push(value)
  - pq.pop()
  - pq.top()
  - pq.empty()

- Time complexity: $\mathcal{O}(\log N)$ for push and pop

Priority queue

- Max-heap by default (top element is largest)
- For min-heap:

```
priority_queue<int, vector<int>, greater<int>> min_pq;
```

- Member functions:
  - pq.push(value)
  - pq.pop()
  - pq.top()
  - pq.empty()
  - pq.size()
- Time complexity: $\mathcal{O}(\log N)$ for push and pop

Boxes Packing

Codeforces 903C

- You have n cube-shaped boxes, each with side length a_i

Boxes Packing

Codeforces 903C

- You have n cube-shaped boxes, each with side length a_i
- You can put box i into box j if $a_i < a_j$, box i is not already inside another box, and box j is empty

Boxes Packing

Codeforces 903C

- You have n cube-shaped boxes, each with side length a_i
- You can put box i into box j if $a_i < a_j$, box i is not already inside another box, and box j is empty
- Boxes can be nested multiple times

Boxes Packing

Codeforces 903C

- You have n cube-shaped boxes, each with side length a_i
- You can put box i into box j if $a_i < a_j$, box i is not already inside another box, and box j is empty
- Boxes can be nested multiple times
- A box is *visible* if it is not inside any other box. Your goal is to minimize the number of visible boxes

Boxes Packing

Codeforces 903C

- You have n cube-shaped boxes, each with side length a_i
- You can put box i into box j if $a_i < a_j$, box i is not already inside another box, and box j is empty
- Boxes can be nested multiple times
- A box is *visible* if it is not inside any other box. Your goal is to minimize the number of visible boxes
- **Input:** one integer n ($1 \leq n \leq 5000$), followed by a list of n integers a_1 to a_n ($1 \leq a_i \leq 10^9$)

Boxes Packing

Codeforces 903C

- You have n cube-shaped boxes, each with side length a_i
- You can put box i into box j if $a_i < a_j$, box i is not already inside another box, and box j is empty
- Boxes can be nested multiple times
- A box is *visible* if it is not inside any other box. Your goal is to minimize the number of visible boxes
- **Input:** one integer n ($1 \leq n \leq 5000$), followed by a list of n integers a_1 to a_n ($1 \leq a_i \leq 10^9$)
- **Output:** the minimum number of visible boxes

Boxes Packing

Rough

Solution

Boxes Packing

- The answer is equal to the most frequent box size in the array

Solution

Boxes Packing

- The answer is equal to the most frequent box size in the array
- To find the result, use a map or dictionary

Solution

Boxes Packing

- The answer is equal to the most frequent box size in the array
- To find the result, use a map or dictionary
- Take the most frequent boxes and put smaller boxes (in decreasing order of size) into free ones (there will always be space)

Solution

Boxes Packing

- The answer is equal to the most frequent box size in the array
- To find the result, use a map or dictionary
- Take the most frequent boxes and put smaller boxes (in decreasing order of size) into free ones (there will always be space)
- Time complexity: $O(n \cdot log(n))$

Room Allocation

CSES 1164

- You manage a hotel with n customers, each needing a single room

Room Allocation

CSES 1164

- You manage a hotel with n customers, each needing a single room
- A customer stays from day a to day b

Room Allocation

CSES 1164

- You manage a hotel with n customers, each needing a single room
- A customer stays from day a to day b
- Two customers can share a room if the first one leaves before the second one arrives

Room Allocation

CSES 1164

- You manage a hotel with n customers, each needing a single room
- A customer stays from day a to day b
- Two customers can share a room if the first one leaves before the second one arrives
- Your task is to find the minimum number of rooms needed and assign rooms to each customer

Room Allocation

CSES 1164

- You manage a hotel with n customers, each needing a single room
- A customer stays from day a to day b
- Two customers can share a room if the first one leaves before the second one arrives
- Your task is to find the minimum number of rooms needed and assign rooms to each customer
- Input: One integer n (number of customers). Then n lines with two integers a and b (arrival and departure days)

Room Allocation

CSES 1164

- You manage a hotel with n customers, each needing a single room
- A customer stays from day a to day b
- Two customers can share a room if the first one leaves before the second one arrives
- Your task is to find the minimum number of rooms needed and assign rooms to each customer
- Input: One integer n (number of customers). Then n lines with two integers a and b (arrival and departure days)
- Output: One integer k (minimum rooms required) A line of n integers — the room number for each customer in input order

Room Allocation

Rough

Solution

Room Allocation

- Sort all customers by their arrival time to handle them in order
- Use a min-priority queue to keep track of current room occupancies by departure time
- Iterate through each customer
  - If the earliest departure in the queue is before the current customer's arrival, reuse that room
  - Otherwise, assign a new room
- This ensures rooms are reused efficiently and total rooms used is minimized
- Time complexity: $O(n \cdot log(n))$

Concert Tickets

CSES1091

- There are n concert tickets, each with a given price

Concert Tickets

CSES1091

- There are n concert tickets, each with a given price
- m customers arrive one by one

Concert Tickets

CSES1091

- There are n concert tickets, each with a given price
- m customers arrive one by one
- Each customer gives their maximum price t_i they are willing to pay

Concert Tickets

CSES1091

- There are n concert tickets, each with a given price
- m customers arrive one by one
- Each customer gives their maximum price t_i they are willing to pay
- A customer receives the most expensive ticket with price $\leq t_i$, if available

Concert Tickets

CSES1091

- There are n concert tickets, each with a given price
- m customers arrive one by one
- Each customer gives their maximum price t_i they are willing to pay
- A customer receives the most expensive ticket with price $\leq t_i$, if available
- Each ticket can be sold to only one customer

Concert Tickets

CSES1091

- There are n concert tickets, each with a given price
- m customers arrive one by one
- Each customer gives their maximum price t_i they are willing to pay
- A customer receives the most expensive ticket with price $\leq t_i$, if available
- Each ticket can be sold to only one customer
- Input
  - First line: integers n and m — number of tickets and customers
  - Second line: n integers $h_1, h_2, \ldots, h_n$ — ticket prices
  - Third line: m integers $t_1, t_2, \ldots, t_m$ — max price each customer is willing to pay

Concert Tickets

CSES1091

- There are n concert tickets, each with a given price
- m customers arrive one by one
- Each customer gives their maximum price t_i they are willing to pay
- A customer receives the most expensive ticket with price $\leq t_i$, if available
- Each ticket can be sold to only one customer
- Input
  - First line: integers n and m — number of tickets and customers
  - Second line: n integers $h_1, h_2, \ldots, h_n$ — ticket prices
  - Third line: m integers $t_1, t_2, \ldots, t_m$ — max price each customer is willing to pay
- Output: For each customer, print the price they pay or -1 if no ticket is available

Concert Tickets

Rough

Solution

Concert Tickets

- We will store all of the tickets prices in a multiset
- For each customer we have find the largest number in the multiset which doesn't exceed t_i then erase it.
- if such number doesn't exist we have to print -1
- we can do it using a multiset and upper_bound

Bit inversions

CSES 1188

- You are given a bit string of length n, consisting of 0s and 1s

Bit inversions

CSES 1188

- You are given a bit string of length n, consisting of 0s and 1s
- you are given m changes, each inverting a specific bit at position x_i

Bit inversions

CSES 1188

- You are given a bit string of length n, consisting of 0s and 1s
- you are given m changes, each inverting a specific bit at position x_i
- After each change, you must find the length of the longest substring where all bits are the same

Bit inversions

CSES 1188

- You are given a bit string of length n, consisting of 0s and 1s
- you are given m changes, each inverting a specific bit at position x_i
- After each change, you must find the length of the longest substring where all bits are the same

Bit inversions

CSES 1188

- You are given a bit string of length n, consisting of 0s and 1s
- you are given m changes, each inverting a specific bit at position x_i
- After each change, you must find the length of the longest substring where all bits are the same
- Input: First line: a bit string of length n. Second line: an integer m (number of changes). Third line: m integers (positions of bits to invert)

Bit inversions

CSES 1188

- You are given a bit string of length n, consisting of 0s and 1s
- you are given m changes, each inverting a specific bit at position x_i
- After each change, you must find the length of the longest substring where all bits are the same
- Input: First line: a bit string of length n. Second line: an integer m (number of changes). Third line: m integers (positions of bits to invert)
- Output: After each change, print the length of the longest uniform bit substring

Bit inversions

Rough

Solution

Bit inversions

- Represent the bit string as $s = s_0 s_1 s_2 \ldots s_{n-1}$
- Use a set `dif` to store indices i where $s_i \neq s_{i-1}$ (include $i = 0$ and $i = n$)
- The lengths of uniform segments are the differences between consecutive elements in `dif`
- Store these segment lengths in a multiset `ret`
- The maximum value in `ret` gives the length of the longest uniform substring
- To handle a bit flip at position x (0-indexed):
  - Insert or remove x and $x + 1$ in `dif` (toggle their presence)
  - Update `ret` by removing and adding affected segment lengths
- After each update, output max(`ret`)

Powering the Hero

Codeforces 1800C2

- You are given a sequence of cards consisting of **two types**: hero cards (power $= 0$) and bonus cards (positive power). You process the cards from left to right. Bonus cards can be stacked or discarded. When a hero card is found, if the bonus stack is not empty, its top value is added to the hero's power, and the hero is added to your army. The goal is to maximize the total power of your army.

Powering the Hero

Codeforces 1800C2

- You are given a sequence of cards consisting of **two types**: hero cards (power $= 0$) and bonus cards (positive power). You process the cards from left to right. Bonus cards can be stacked or discarded. When a hero card is found, if the bonus stack is not empty, its top value is added to the hero's power, and the hero is added to your army. The goal is to maximize the total power of your army.
- example: for $[2, 1, 5, 0, 0]$, the answer is 7

Powering the Hero

Rough

Solution

Powering the Hero

- if $a_i > 0$, push that to a max-heap. Otherwise, pop the top element and add that to the answer.

Unexpressed

Atcoder abc193_c

- Given an integer $N(1 \leq N \leq 10^{10})$. How many integers from 1 to N cannot be expressed as a^b for some integers $a, b > 1$?

Unexpressed

Atcoder abc193_c

- Given an integer $N(1 \leq N \leq 10^{10})$. How many integers from 1 to N cannot be expressed as a^b for some integers $a, b > 1$?
- Example: for $N = 8$, answer is 6 because $1, 2, 3, 5, 6, 7$ these numbers cannot be expressed as some a^b.

Unexpressed

Rough

Solution

Unexpressed

- Store the numbers that **can be** expressed in a set. Then for some N, you can use **lower_bound** to find out how many numbers $\leq N$ can be expressed. The rest of the numbers will be unexpressed.

Sequence Pair Weight

Codeforces 1527C

- The *weight* of a sequence is defined as the number of unordered pairs of indexes (i, j) (here $i < j$) with the same value ($a_i = a_j$). For example, the weight of sequence $a = [1, 1, 2, 2, 1]$ is 4. The set of unordered pairs of indices with the same value is $(1, 2)$, $(1, 5)$, $(2, 5)$, and $(3, 4)$.

Sequence Pair Weight

Codeforces 1527C

- The *weight* of a sequence is defined as the number of unordered pairs of indexes (i, j) (here $i < j$) with the same value ($a_i = a_j$). For example, the weight of sequence $a = [1, 1, 2, 2, 1]$ is 4. The set of unordered pairs of indices with the same value is $(1, 2)$, $(1, 5)$, $(2, 5)$, and $(3, 4)$.
- You are given a sequence $A (1 \leq A_i \leq 10^9)$ of $N (1 \leq N \leq 10^5)$ integers. Print the sum of the weight of all subsegments of A.

Sequence Pair Weight

Rough

Solution

Sequence Pair Weight

- First try to solve for only one subsegment.

Solution

Sequence Pair Weight

- First try to solve for only one subsegment.
- Try to think about how many times A_i can appear as a right element of the pairs (x, A_i). How many possible x can be there?

Solution

Sequence Pair Weight

- First try to solve for only one subsegment.
- Try to think about how many times A_i can appear as a right element of the pairs (x, A_i). How many possible x can be there?
- A_i can be a right-contributor of a subsegment $\sum_{\substack{j<i \\ A_j=A_i}} 1$ times.

Solution

Sequence Pair Weight

- First try to solve for only one subsegment.
- Try to think about how many times A_i can appear as a right element of the pairs (x, A_i). How many possible x can be there?
- A_i can be a right-contributor of a subsegment $\sum_{\substack{j<i \\ A_j=A_i}} 1$ times.
- For the final solution, multiply the previous portion with how many times A_i can appear in a subsegment.

Solution

Sequence Pair Weight

- First try to solve for only one subsegment.
- Try to think about how many times A_i can appear as a right element of the pairs (x, A_i). How many possible x can be there?
- A_i can be a right-contributor of a subsegment $\sum_{\substack{j<i \\ A_j=A_i}} 1$ times.
- For the final solution, multiply the previous portion with how many times A_i can appear in a subsegment.
- Final Answer: $\sum_{i=1}^{n} F(i)$ Where,

$$F(i) = (n - i + 1) \sum_{\substack{j<i, \\ A_j=A_i}} j$$

Potions (Hard Version)

Codeforces 1526C2

- Given an array $A(-10^9 \leq A_i \leq 10^9)$ of $N(1 \leq N \leq 2 \cdot 10^5)$. Go from left to right, you can either take A_i or discard it. If you take, add that to your sum. Your sum cannot be negative at any point. What is the maximum number of elements you can take?

Potions(Hard Version)

Rough

Solution

Potions(Hard Version)

- If $A_i \geq 0$, take it, no loss.

Solution

Potions(Hard Version)

- If $A_i \geq 0$, take it, no loss.
- Otherwise, if sum $+ A_i \geq 0$, take it and insert A_i to a min-heap.

Solution

Potions(Hard Version)

- If $A_i \geq 0$, take it, no loss.
- Otherwise, if sum $+ A_i \geq 0$, take it and insert A_i to a min-heap.
- If sum $+ A_i < 0$, check the top element of the min heap, and if it's smaller than A_i, swap that with A_i.

Towers

CSES1073

- Given $N(1 \leq N \leq 10^5$ cubes, with i-th cube having length $h_i(1 \leq h_i \leq 10^9)$. You have to make towers with them. To make a tower, you have to place a smaller cube on top of a larger cube.

Towers

CSES1073

- Given $N(1 \leq N \leq 10^5$ cubes, with i-th cube having length $h_i(1 \leq h_i \leq 10^9)$. You have to make towers with them. To make a tower, you have to place a smaller cube on top of a larger cube.
- What is the minimum number of towers can you make?

Powering the Hero

Rough

Solution

Towers

- for some cube with h_i, you can make a new tower, or put that on top of another tower, that has another cube on top of it, let it be g_i, and $g_i > h_i$. Among all those options, put it on top of the tower, which has the smallest possible cube currently.
- keep a multiset to keep track of the top of all towers.

- Good Bye!