BRACU CP Workshop

Day 3

Shehran Rahman, A.J.M Istiaque

BRAC University, Dhaka

16 April 2025

Brute Force Approach

- **Core Idea:** Exhaustively check all possible solutions (Complete Search).

Brute Force Approach

- **Core Idea:** Exhaustively check all possible solutions (Complete Search).
- **Key Steps:**
  - Define Problem Space
  - Generate all candidates (e.g., subsets, permutations).
  - Validate each against problem constraints.
  - Track the best/valid solution.

Brute Force Approach

- **Core Idea:** Exhaustively check all possible solutions (Complete Search).
- **Key Steps:**
  - Define Problem Space
  - Generate all candidates (e.g., subsets, permutations).
  - Validate each against problem constraints.
  - Track the best/valid solution.
- **Techniques:**
  - Recursion + Backtracking.
  - Bitmasking.
  - Nested loops.

Brute Force Approach

- **Core Idea:** Exhaustively check all possible solutions (Complete Search).
- **Key Steps:**
  - Define Problem Space
  - Generate all candidates (e.g., subsets, permutations).
  - Validate each against problem constraints.
  - Track the best/valid solution.
- **Techniques:**
  - Recursion + Backtracking.
  - Bitmasking.
  - Nested loops.
- **Pros and Cons**
  - Generally simple to implement.
  - Guarantees correctness (if a solution exists, it will be found).
  - Generally inefficient for large inputs.

Problem

For Wizards, the Exam Is Easy, but I Couldn't Handle It

- **Problem Statement:**
  - Perform **exactly one** cyclic left shift on any subarray $[l, r]$
  - Goal: Minimize the number of inversions in resulting array
  - Inversion: Pair (i, j) where $i < j$ and $a_i > a_j$
- **Input:**
  - t test cases $(1 \leq t \leq 10^4)$
  - Per test case: n $(1 \leq n \leq 2000)$ and array a $(1 \leq a_i \leq 2000)$
  - Total n^2 across tests $\leq 4 \times 10^6$
- **Output:**
  - Optimal l and r (1-based) for the cyclic shift
- **Example:**
  - Input: 4
    2 1 2 1
  - Possible solution: Shift $[1, 4] \rightarrow$ 1 2 1 2 (fewer inversions)
  - Output: 1 4

Generating Subsets

Using Recursion

- **Key Idea:**
  - Each element has two choices: **include** or **exclude**.
  - Recursively explore both options to generate all 2^n subsets.

Generating Subsets

Using Recursion

- **Key Idea:**
  - Each element has two choices: **include** or **exclude**.
  - Recursively explore both options to generate all 2^n subsets.
- **Steps:**
  - Start with an empty subset.
  - At each step, branch:
    - Include current element.
    - Exclude current element.
  - Recurse until all elements are processed.

Generating Subsets

Using Recursion

- **Key Idea:**
  - Each element has two choices: **include** or **exclude**.
  - Recursively explore both options to generate all 2^n subsets.
- **Steps:**
  - Start with an empty subset.
  - At each step, branch:
    - Include current element.
    - Exclude current element.
  - Recurse until all elements are processed.
- **Complexity:**
  - Time: $O(2^n)$ (total subsets).
  - Space: $O(n)$ (recursion depth).

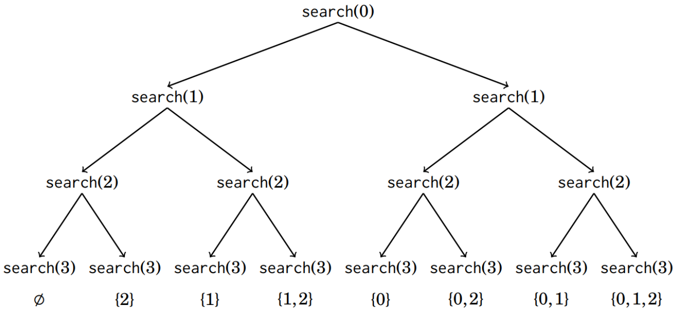
Code

Generating Subsets

```
void search(int k) {
    if (k == n) {
        // process subset
    } else {
        // not take
        search(k+1);
        subset.push_back(k);
        // take
        search(k+1);
        subset.pop_back();
    }
}
```

Generating Subsets

Complete Search Tree



Bitmasking for Subset Generation

- **Problem:**
  - Generate all possible subsets of a set with n elements
  - Example: For $\{A, B, C\} \rightarrow \{\emptyset, \{A\}, \{B\}, \ldots, \{A, B, C\}\}$

Bitmasking for Subset Generation

- **Problem:**
  - Generate all possible subsets of a set with n elements
  - Example: For $\{A, B, C\} \to \{\emptyset, \{A\}, \{B\}, \ldots, \{A, B, C\}\}$
- **Bitmask Approach:**
  - Each subset represented by an n-bit binary number
  - Bit $i = 1 \to$ include i^{th} element
  - n elements $\to 2^n$ possible subsets

Bitmasking for Subset Generation

- **Problem:**
  - Generate all possible subsets of a set with n elements
  - Example: For $\{A, B, C\} \rightarrow \{\emptyset, \{A\}, \{B\}, \ldots, \{A, B, C\}\}$
- **Bitmask Approach:**
  - Each subset represented by an n-bit binary number
  - Bit $i = 1 \rightarrow$ include i^{th} element
  - n elements $\rightarrow 2^n$ possible subsets
- **Key Insight:**
  - Iterate from 0 to $2^n - 1$ (all possible bitmasks)
  - Each number's binary representation = unique subset

Bitmasking for Subset Generation

- **Problem:**
  - Generate all possible subsets of a set with n elements
  - Example: For $\{A, B, C\} \to \{\emptyset, \{A\}, \{B\}, \ldots, \{A, B, C\}\}$
- **Bitmask Approach:**
  - Each subset represented by an n-bit binary number
  - Bit $i = 1 \to$ include i^{th} element
  - n elements $\to 2^n$ possible subsets
- **Key Insight:**
  - Iterate from 0 to $2^n - 1$ (all possible bitmasks)
  - Each number's binary representation $=$ unique subset
- **Operations:**
  - Check if element i is in subset: `mask & (1 << i)`
  - Add element i: `mask | (1 << i)`
  - Remove element i: `mask & ~(1 << i)`

Code

Generating Subsets (Using Bitmasks)

```
for(int mask = 0; mask < (1 << n); ++mask){
    vector<int> subset;
    for(int i = 0; i < n; ++i){
        if(mask & (1 << i)) subset.push_back(i);
    }

    // do something on the subset
}
```

Problem 1

Apple Division

- **Objective:**
  - Divide n apples into **two groups**
  - Minimize the absolute difference in their total weights
- **Input and Constraints:**
  - n: Number of apples $(1 \leq n \leq 20)$
  - $p_1, p_2, \ldots, p_n$: Weights $(1 \leq p_i \leq 10^9)$
- **Output:**
  - Single integer: Minimum possible weight difference
- **Example:**
  - Input: 5 apples with weights [3, 2, 7, 4, 1]
  - Optimal division:
    - Group A: $2 + 3 + 4 = \mathbf{9}$
    - Group B: $1 + 7 = \mathbf{8}$
  - Output: **1**

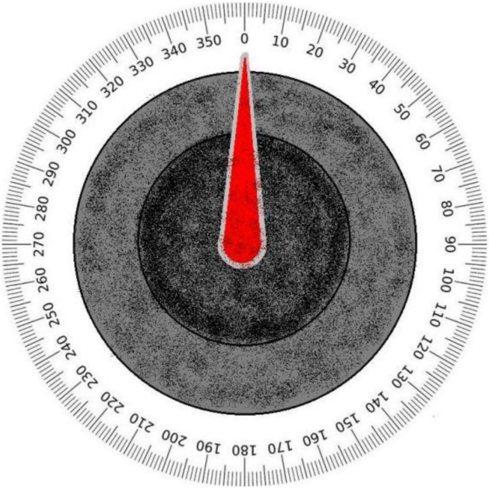
Problem 2

Petr and Combination Lock

- **Problem Statement:**
  - Lock has 360° scale, starts at 0°
  - Must perform n rotations (choose $\pm a_i$ each time)
  - After all rotations, must return to 0°
- **Input:**
  - n rotations ($1 \leq n \leq 15$)
  - Angles a_1 to a_n ($1 \leq a_i \leq 180$)
- **Output:**
  - "YES" if possible, "NO" otherwise

Petr and Combination Lock

Figure



Problem 3

Preparing Olympiad

- **Problem Statement:**
  - Select a subset of ≥ 2 problems from n available problems
  - Must satisfy three conditions:
    1. Total difficulty $\in [l, r]$
    2. Max-min difficulty $\geq x$
- **Input:**
  - n, l, r, x $(1 \leq n \leq 15,\ 1 \leq l \leq r \leq 10^9,\ 1 \leq x \leq 10^6)$
  - $c_1, c_2, \ldots, c_n$ $(1 \leq c_i \leq 10^6)$
- **Output:**
  - Number of valid subsets

Generating Permutations Recursively

- **Core Approach:**
  - Build permutations incrementally by selecting unused elements
  - Maintain:
    - Current partial permutation
    - Tracking of used elements
- **Base Case:**
  - When current permutation reaches full size (n elements)
  - A complete permutation is ready for processing
- **Recursive Step:**
  - For each element not yet in current permutation:
    - Mark element as used
    - Add it to current permutation
    - Recurse to build remainder
    - Backtrack: unmark element and remove from permutation

Code

Generating Permutations Recursively

```
void search() {
    if (permutation.size() == n) {
        // process permutation
    } else {
        for (int i = 0; i < n; i++) {
            if (chosen[i]) continue;
            chosen[i] = true;
            permutation.push_back(i);
            search();
            chosen[i] = false;
            permutation.pop_back();
        }
    }
}
```

Generating Permutations Using `next_permutation()`

- **Purpose:**
  - Efficiently generates **lexicographically ordered** permutations
  - Modifies sequence in-place to next greater permutation
- **Requirements:**
  - Input sequence must be **sorted** (to get all permutations)
  - Elements must have **defined comparison** ($<$ operator)
- **Usage Pattern:**
  - Start with sorted sequence
  - Call in loop until it returns `false`
  - Each call generates next permutation
  - Can it be used to generate **all combinations (all r selections out of n)** ???

Code

Generating Permutations Using next_permutation()

```
vector<int> permutation;
for (int i = 0; i < n; i++) {
    permutation.push_back(i);
}
do {
    // process permutation
} while (next_permutation(permutation.begin(),
   permutation.end()));
```

Problem 4

Snake Years

- **Problem Statement:**
  - Given n digits, find all valid years by possible rearrangements:
    - No leading zeros
    - Use exactly the given digits
    - Follows a 12 year cycle
    - Year Pattern: 2001, 2013, 2025, 2037, 2049, ...
- **Input:**
  - $1 \leq n \leq 6$ - number of digits
  - $d_1, d_2, \ldots, d_n$ (0-9) - available digits
- **Output:**
  - Count of valid Snake years formable
- **Example:**
  - Input: 4
    2 0 2 5
  - Valid year: 2025 (next Snake year)
  - Output: 2

Problem 5

Find the Number

- **Given**:
  - Sorted array of N distinct integers ($1 \leq N \leq 10^5$)
  - Elements lie in the range 1 to 10^9
  - Q queries ($1 \leq Q \leq 10^5$)
- **Query**:
  - An integer X
  - Check if X exists in the array
- **Output**:
  - If X exists, output its **0-based index**
  - Else, output -1

Code

Find the Number

```
int l = 0, r = n - 1, ans = -1;
while (l <= r) {
    int m = (l + r) / 2;
    if (a[m] == X) {
        ans = m;
        break;
    }
    if (a[m] < X) l = m + 1;
    else r = m - 1;
}
//print ans
```

`lower_bound` and `upper_bound`

- `lower_bound`: Returns an iterator to the first element **not less than** the given value.
- `upper_bound`: Returns an iterator to the first element **greater than** the given value.

lower_bound and upper_bound

Usage

```cpp
vector<int> v = {2, 3, 7, 7, 7, 10, 14, 20, 23};
auto it = lower_bound(v.begin(), v.end(), 7);
int index = it - v.begin(); // index = 2
int value = *it;            // value = 7


auto it = upper_bound(v.begin(), v.end(), 7);
int index = it - v.begin(); // index = 5
int value = *it;            // value = 10
```

How to Identify a Binary Search Problem

- Existence of monotonic property

How to Identify a Binary Search Problem

- Existence of monotonic property
- Minimise the maximum value or maximize the minimum value

How to Identify a Binary Search Problem

- Existence of monotonic property
- Minimise the maximum value or maximize the minimum value
- Usually have two cases (searching for the answer itself, searching for a value that u need in an array)

Problem 6

Eating Queries

- **Given**:
  - n candies, each with a sugar value a_i $(1 \leq a_i \leq 10^4)$
  - q queries, each asking for a target sugar amount x_j
  - The same candy **cannot** be eaten twice in a single query
  - Queries are **independent** (Timur can reuse candies in different queries)
- **Query**:
  - For each x_j $(1 \leq x_j \leq 2 \cdot 10^9)$, find the **minimum number of candies** Timur needs to eat such that the total sugar consumed is $\geq x_j$
  - If it is not possible, output -1
- **Constraints**:
  - $1 \leq n, q \leq 10^5$

Problem 7

Eco-Friendly Wood Cutting

- **Given**:
  - N trees with heights $h_1, h_2, \ldots, h_N$
  - A required wood amount K
  - A sawblade that can be set to a height H
- **Cutting Rule**:
  - Any part of a tree that is **above** the height H is cut and collected
  - Trees with height $\leq H$ are not affected
  - Total wood collected is the sum of all $(h_i - H)$ for each $h_i > H$
- **Objective**:
  - Determine the **maximum possible value of** H such that at least K units of wood are collected
- **Constraints**:
  - $1 \leq N \leq 10^5$
  - $1 \leq M \leq 2 \cdot 10^9$
  - $1 \leq h_i \leq 10^9$
  - It is guaranteed that the total available wood is at least K

Code

Eco-Friendly Wood Cutting

```
bool ok(m) {
    ....Write your logic
}

int l = 0, r = 1e9, ans;

while (l <= r) {
    int m = (l + r) / 2;
    if (ok(m)) ans = m, l = m + 1;
    else r = m - 1;
}
//print ans
```

Problem 8

Limited Library

- **Given**:
  - A bookcase with n shelves, each with a specific height
  - m books, each with a specific height (all have the same width)
  - Each shelf can:
    - Hold up to x books if **no art piece** is placed
    - Hold up to y books if **an art piece** is placed (art takes space of $x - y$ books)
    - Only hold books whose heights are $\leq$ the shelf height
    - Have at most one art piece
- **Objective**:
  - Place all m books across the n shelves
  - Maximize the number of shelves that have an art piece
  - If it is not possible to place all the books, output `"impossible"`
- **Input**:
  - A line with four integers: n, m, x, y $(1 \leq n, m \leq 10^5,\ 1 \leq y < x \leq 1000)$
  - A line with n integers: shelf heights
  - A line with m integers: book heights

Problem 8

Limited Library

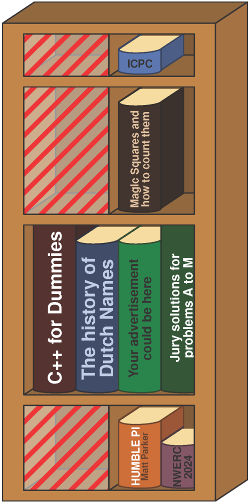


Figure: Three shelves can have art pieces in the hatched areas, while still fitting all new books.

Problem 9

Guess the Number (Interactive)

- Let's play a game!
- I will pick a secret number x in the range $[0, 10^5]$.
- In each attempt, you can ask me a number.
- I will respond with one of the following:
  - `"Bigger"` – if x is greater than your guess
  - `"Smaller"` – if x is less than your guess
  - `"Bingo!"` – if your guess is correct (you found x!)
- You are allowed to guess at most **20 times**.

Continuous Binary Search

Square Root of a Number

- Search space is over real numbers and need to compute answers with a certain **precision**
- Let's take an example:
  - Given a number x, find its square root up to 6 decimal places
  - That is, find r such that $r^2 \approx x$

Continuous Binary Search

Square Root of a Number

- Search space is over real numbers and need to compute answers with a certain **precision**
- Let's take an example:
  - Given a number x, find its square root up to 6 decimal places
  - That is, find r such that $r^2 \approx x$
- **Strategy**:
  - Search range: $[0, x]$
  - While (r − l) $> \varepsilon$, do:
    - $m = \frac{l+r}{2}$
    - if $m^2 < x$, move l to m
    - else, move r to m
  - Final answer is in range $[l, r]$

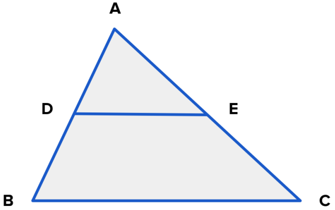
Continuous Binary Search

Square Root of a Number

- Search space is over real numbers and need to compute answers with a certain **precision**
- Let's take an example:
  - Given a number x, find its square root up to 6 decimal places
  - That is, find r such that $r^2 \approx x$
- **Strategy**:
  - Search range: $[0, x]$
  - While $(\mathrm{r} - \mathrm{l}) > \varepsilon$, do:
    - $m = \frac{l+r}{2}$
    - if $m^2 < x$, move l to m
    - else, move r to m
  - Final answer is in range $[l, r]$
- **Precision Output in C++**:
  - Use: `cout << fixed << setprecision(6) << answer;`

Problem 10

Triangle Partitioning



- **Statement**:
  - You are given AB, AC and BC. DE is parallel to BC. You are also given the area ratio between ADE and $BDEC$. You have to find the value of AD.
  - Errors less than 10^{-6} will be ignored
- **Constraints**:
  - $0 < AB, AC, BC \leq 10^4$
  - $0 < k \leq 10^6$

Ternary Search

Stick Lengths

- **Task:**
  - There are n sticks with some lengths. Modify the sticks so that each stick has the same length.
  - You can either lengthen or shorten each stick. Both operations cost x, where x is the difference between the new and original length.
  - What is the minimum total cost?

Ternary Search

Stick Lengths

- **Task:**
  - There are n sticks with some lengths. Modify the sticks so that each stick has the same length.
  - You can either lengthen or shorten each stick. Both operations cost x, where x is the difference between the new and original length.
  - What is the minimum total cost?
- **Constraints:**
  - $1 \leq n \leq 2 \cdot 10^5$
  - $1 \leq a_i \leq 10^9$

Code

Stick lengths

```
int cost(m) {
    ...Write your logic
}

int l = 0, r = 1e9, ans= INFINITY;
while (l <= r) {
    int m1 = l + (r - l) / 3;
    int m2 = r - (r - l) / 3;
    int c1 = cost(m1), c2 = cost(m2);
    if (c1 <= c2) {
        ans = min(ans, c1);
        r = m2 - 1;
    }
    else {
        ans = min(ans, c2);
        l = m1 + 1;
    }
}
//print ans
```