DSA Course Solutions in C++

DSA Course Solutions in C++

I organize them by week and topic for clarity. For patterns, I provide the code to print the pattern. For concepts like "why global variables are bad" or "concepts", I provide explanations as they aren't coding problems. For articles or concepts (e.g., exponential search), I explain briefly as they are not direct codes.

Note: For LeetCode/GFG problems, I implement the function as per the problem signature (e.g., class Solution). For simple programs, I use main(). I include <bits/stdc++.h> for convenience in competitive coding style.

WEEK 01 TOPIC: Introduction to programming

CLASS NOTES: 01

No specific coding problems; concepts only.

CLASS HOMEWORK: 01

1. Examples of pseudocode and flowchart
    - Pseudocode example for adding two numbers:

```
START
READ a, b
sum = a + b
PRINT sum
END
```

        - Flowchart: (Text description) Start -> Input a, b -> Process sum = a + b -> Output sum -> End. (No code needed.)

WEEK 01 TOPIC: Write your first C++ program

CLASS NOTES: 02

1. Hello World

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

int main() {
    cout << "Hello World";
```

```
    return 0;
}
```

Output: `Hello World`

2. Print “Love Babbar”

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

int main() {
    cout << "Love Babbar";
    return 0;
}
```

Output: `Love Babbar`

3. Variables and Data Types example

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

int main() {
    int age = 21;
    float pi = 3.14;
    char grade = 'A';
    bool isStudent = true;
    cout << "Age: " << age << endl;
    cout << "Pi: " << pi << endl;
    cout << "Grade: " << grade << endl;
    cout << "Is Student: " << isStudent;
    return 0;
}
```

Output:

```
Age: 21
Pi: 3.14
Grade: A
Is Student: 1
```

4. Signed and Unsigned data

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

int main() {
    signed int x = -10;
```

```
    unsigned int y = 10;
    cout << "Signed: " << x << endl;
    cout << "Unsigned: " << y;
    return 0;
}
```

Output:

```
Signed: -10
Unsigned: 10
```

5. Operators in C++

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

int main() {
    int a = 10, b = 3;
    cout << "a+b = " << a+b << endl;
    cout << "a-b = " << a-b << endl;
    cout << "a*b = " << a*b << endl;
    cout << "a/b = " << a/b << endl;
    cout << "a%b = " << a%b << endl;
    return 0;
}
```

Output:

```
a+b = 13
a-b = 7
a*b = 30
a/b = 3
a%b = 1
```

CLASS HOMEWORK: 02

1. 32 bits VS 64 bit architecture
    - Explanation: 32-bit: Max 4GB memory, 32-bit addresses. 64-bit: Larger memory, faster. No code.
2. Typecasting: implicit and explicit
    - Implicit:

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

int main() {
    int a = 10;
```

```
    double b = a; // implicit (int to double)
    cout << b;
    return 0;
}
```

Output: `10`

- Explicit:

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

int main() {
    double x = 10.56;
    int y = (int)x; // explicit
    cout << y;
    return 0;
}
```

Output: `10`

3. Number system: binary to decimal

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

int binaryToDecimal(int n) {
    int decimal = 0, i = 0;
    while (n > 0) {
        int lastDigit = n % 10;
        decimal += lastDigit * pow(2, i);
        n /= 10;
        i++;
    }
    return decimal;
}

int main() {
    int binary;
    cin >> binary;
    cout << "Decimal = " << binaryToDecimal(binary);
    return 0;
}
```

Input: `1011`

Output: `Decimal = 11`

WEEK 01 TOPIC: Conditionals and loops

CLASS NOTES: 03

Pattern 1: Square pattern

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

int main() {
    int n;
    cin >> n;
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            cout << "*";
        }
        cout << endl;
    }
    return 0;
}
```

Input: `4`
Output:

```
****
****
****
****
```

Pattern 2: Rectangle pattern

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

int main() {
    int r, c;
    cin >> r >> c;
    for(int i = 0; i < r; i++) {
        for(int j = 0; j < c; j++) {
            cout << "*";
        }
        cout << endl;
    }
    return 0;
}
```

Input: `3 5`
Output:

```
*****
*****
*****
```

Pattern 3: Hollow Rectangle pattern

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

int main() {
    int r, c;
    cin >> r >> c;
    for(int i = 0; i < r; i++) {
        for(int j = 0; j < c; j++) {
            if(i == 0 || i == r-1 || j == 0 || j == c-1) cout << "*";
            else cout << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Input: `4 5`
Output:

```
*****
*   *
*   *
*****
```

Pattern 4: Half Pyramid pattern

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

int main() {
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= i; j++) {
            cout << "*";
        }
```

```
        cout << endl;
    }
    return 0;
}
```

Input: `4`
Output:

```
*
**
***
****
```

Pattern 5: Inverted Half Pyramid pattern

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

int main() {
    int n;
    cin >> n;
    for(int i = n; i >= 1; i--) {
        for(int j = 1; j <= i; j++) {
            cout << "*";
        }
        cout << endl;
    }
    return 0;
}
```

Input: `4`
Output:

```
****
***
**
*
```

Pattern 6: Numeric Half Pyramid pattern

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

int main() {
    int n;
    cin >> n;
```

```
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= i; j++) {
            cout << j;
        }
        cout << endl;
    }
    return 0;
}
```

Input: `4`
Output:

```
1
12
123
1234
```

Pattern 7: Inverted Numeric Half Pyramid pattern

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

int main() {
    int n;
    cin >> n;
    for(int i = n; i >= 1; i--) {
        for(int j = 1; j <= i; j++) {
            cout << j;
        }
        cout << endl;
    }
    return 0;
}
```

Input: `4`
Output:

```
1234
123
12
1
```

CLASS HOMEWORK: 03

0. All flowcharts are converted into CPP programs

- Assumption: Flowcharts for the following homework programs. (Codes below are the conversions.)

1. Multiply two numbers by taking input from user

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

int main() {
    int a, b;
    cin >> a >> b;
    cout << a * b;
    return 0;
}
```

Input: `3 4`
Output: `12`

2. Find the perimeter of a triangle

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

int main() {
    int a, b, c;
    cin >> a >> b >> c;
    cout << a + b + c;
    return 0;
}
```

Input: `3 4 5`
Output: `12`

3. Find the simple interest

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

int main() {
    float p, r, t;
    cin >> p >> r >> t;
    cout << (p * r * t) / 100;
    return 0;
}
```

Input: `1000 5 2`
Output: `100`

4. Find the compound interest

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

int main() {
    float p, r, t;
    cin >> p >> r >> t;
    cout << p * pow((1 + r/100), t) - p;
    return 0;
}
```

Input: `1000 5 2`
Output: `102.5` (approximate)

5. Print counting from n to 1

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

int main() {
    int n;
    cin >> n;
    for(int i = n; i >= 1; i--) {
        cout << i << " ";
    }
    return 0;
}
```

Input: `5`
Output: `5 4 3 2 1`

6. Find the factorial of a number

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

int main() {
    int n, fact = 1;
    cin >> n;
    for(int i = 1; i <= n; i++) {
        fact *= i;
    }
    cout << fact;
    return 0;
}
```

Input: `5`

Output: `120`

7. Check if number is prime or not

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

int main() {
    int n;
    cin >> n;
    bool isPrime = true;
    for(int i = 2; i * i <= n; i++) {
        if(n % i == 0) isPrime = false;
    }
    cout << (isPrime ? "Prime" : "Not Prime");
    return 0;
}
```

Input: `7`

Output: `Prime`

8. Check valid triangle or not

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

int main() {
    int a, b, c;
    cin >> a >> b >> c;
    if(a + b > c && a + c > b && b + c > a) cout << "Valid";
    else cout << "Invalid";
    return 0;
}
```

Input: `3 4 5`

Output: `Valid`

9. Print max of three numbers

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

int main() {
    int a, b, c;
    cin >> a >> b >> c;
    cout << max(a, max(b, c));
```

```
    return 0;
}
```

Input: `3 7 5`

Output: `7`

WEEK 02 TOPIC: Pattern continues

CLASS NOTES: 04

Pattern 08: Full Pyramid

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

int main() {
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n-i; j++) cout << " ";
        for(int j = 1; j <= 2*i - 1; j++) cout << "*";
        cout << endl;
    }
    return 0;
}
```

Input: `4`

Output:

```
   *
  ***
 *****
*******
```

Pattern 09: Inverted Full Pyramid

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

int main() {
    int n;
    cin >> n;
    for(int i = n; i >= 1; i--) {
        for(int j = 1; j <= n-i; j++) cout << " ";
        for(int j = 1; j <= 2*i - 1; j++) cout << "*";
        cout << endl;
```

```
    }
    return 0;
}
```

Input: `4`

Output:

```
*******
 *****
  ***
   *
```

Pattern 10: Dimond Pattern

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

int main() {
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n-i; j++) cout << " ";
        for(int j = 1; j <= 2*i - 1; j++) cout << "*";
        cout << endl;
    }
    for(int i = n-1; i >= 1; i--) {
        for(int j = 1; j <= n-i; j++) cout << " ";
        for(int j = 1; j <= 2*i - 1; j++) cout << "*";
        cout << endl;
    }
    return 0;
}
```

Input: `4`

Output:

```
 *
***
*****
*******
*****
***
 *
```

Pattern 11: Hollow Full Pyramid

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

int main() {
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n-i; j++) cout << " ";
        for(int j = 1; j <= 2*i - 1; j++) {
            if(j == 1 || j == 2*i-1 || i == n) cout << "*";
            else cout << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Input: `4`
Output:

```
   *
  * *
 *   *
*******
```

Pattern 12: Inverted Hollow Full Pyramid

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

int main() {
    int n;
    cin >> n;
    for(int i = n; i >= 1; i--) {
        for(int j = 1; j <= n-i; j++) cout << " ";
        for(int j = 1; j <= 2*i - 1; j++) {
            if(j == 1 || j = 2*i-1 || i == n) cout << "*";
            else cout << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Input: `4`
Output:

```
*****
*   *
 * *
  *
```

Pattern 13: Hollow Diamond pattern

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

int main() {
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n-i; j++) cout << " ";
        for(int j = 1; j <= 2*i - 1; j++) {
            if(j == 1 || j == 2*i-1) cout << "*";
            else cout << " ";
        }
        cout << endl;
    }
    for(int i = n-1; i >= 1; i--) {
        for(int j = 1; j <= n-i; j++) cout << " ";
        for(int j = 1; j <= 2*i - 1; j++) {
            if(j == 1 || j == 2*i-1) cout << "*";
            else cout << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Input: `4`
Output:

```
 *
* *
*   *
*     *
*   *
```

```
* *
 *
```

Pattern 14: Flipped Solid Diamond pattern

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

int main() {
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n-i+1; j++) cout << "*";
        for(int j = 1; j <= 2*i - 2; j++) cout << " ";
        for(int j = 1; j <= n-i+1; j++) cout << "*";
        cout << endl;
    }
    for(int i = n; i >= 1; i--) {
        for(int j = 1; j <= n-i+1; j++) cout << "*";
        for(int j = 1; j <= 2*i - 2; j++) cout << " ";
        for(int j = 1; j <= n-i+1; j++) cout << "*";
        cout << endl;
    }
    return 0;
}
```

Input: `4`
Output:

```
**** ****
***   ***
**     **
*       *
*       *
**     **
***   ***
**** ****
```

Pattern 15: Fancy pattern 1

- Assumption: Common fancy pattern, e.g., number pyramid.

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

```
int main() {
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= i; j++) cout << j << " ";
        cout << endl;
    }
    return 0;
}
```

Input: `4`

Output:

```
1
1 2
1 2 3
1 2 3 4
```

Pattern 17: Inverted Hollow Half Pyramid

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

int main() {
    int n;
    cin >> n;
    for(int i = n; i >= 1; i--) {
        for(int j = 1; j <= i; j++) {
            if(j == 1 || j == i || i == n) cout << "*";
            else cout << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Input: `4`

Output:

```
****
*  *
* *
*
```

Pattern 19: Fancy pattern 4

- Assumption: Numeric hollow pyramid.

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

int main() {
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n-i; j++) cout << " ";
        for(int j = 1; j <= i; j++) {
            if(j == 1 || j == i || i == n) cout << j;
            else cout << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Input: `4`
Output:

```
   1
  1 2
 1   3
1234
```

CLASS HOMEWORK: 04

Pattern 16: Fancy pattern 2

- Assumption: Alphabet pyramid.

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

int main() {
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= i; j++) cout << char('A' + j - 1) << " ";
        cout << endl;
    }
    return 0;
}
```

Input: `4`
Output:

```
A
A B
A B C
A B C D
```

Pattern 18: Fancy pattern 3 -> Numerical Hollow Half Pyramid

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

int main() {
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= i; j++) {
            if(j == 1 || j == i || i == n) cout << j;
            else cout << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Input: `5`
Output:

```
1
1 2
1  3
1   4
12345
```

Pattern 20: Numeric Hollow Inverted Half Pyramid

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

int main() {
    int n;
    cin >> n;
    for(int i = n; i >= 1; i--) {
        for(int j = 1; j <= i; j++) {
```

```
            if(j == 1 || j == i || i == n) cout << j;
            else cout << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Input: `5`
Output:

```
12345
1   4
1  3
1 2
1
```

Pattern 21: Numeric Palindrome Equilateral Pyramid

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

int main() {
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n-i; j++) cout << " ";
        for(int j = 1; j <= i; j++) cout << j;
        for(int j = i-1; j >= 1; j--) cout << j;
        cout << endl;
    }
    return 0;
}
```

Input: `4`
Output:

```
   1
  121
 12321
1234321
```

Pattern 22: Fancy pattern 5

- Assumption: Another fancy, e.g., diamond numbers.

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

int main() {
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n-i; j++) cout << " ";
        for(int j = 1; j <= i; j++) cout << j;
        for(int j = i-1; j >= 1; j--) cout << j;
        cout << endl;
    }
    for(int i = n-1; i >= 1; i--) {
        for(int j = 1; j <= n-i; j++) cout << " ";
        for(int j = 1; j <= i; j++) cout << j;
        for(int j = i-1; j >= 1; j--) cout << j;
        cout << endl;
    }
    return 0;
}
```

Input: `3`

Output:

```
  1
 121
12321
 121
  1
```

Pattern 23: Solid Half Diamond

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

int main() {
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= i; j++) cout << "*";
        cout << endl;
    }
    for(int i = n-1; i >= 1; i--) {
```

```
        for(int j = 1; j <= i; j++) cout << "*";
        cout << endl;
    }
    return 0;
}
```

Input: `4`
Output:

```
*
**
***
****
***
**
*
```

Pattern 24: Floyd Triangle

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

int main() {
    int n, num = 1;
    cin >> n;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= i; j++) {
            cout << num++ << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Input: `4`
Output:

```
1
2 3
4 5 6
7 8 9 10
```

Pattern 25: Butterfly Pattern

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

```
int main() {
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= i; j++) cout << "*";
        for(int j = 1; j <= 2*(n-i); j++) cout << " ";
        for(int j = 1; j <= i; j++) cout << "*";
        cout << endl;
    }
    for(int i = n; i >= 1; i--) {
        for(int j = 1; j <= i; j++) cout << "*";
        for(int j = 1; j <= 2*(n-i); j++) cout << " ";
        for(int j = 1; j <= i; j++) cout << "*";
        cout << endl;
    }
    return 0;
}
```

Input: `4`
Output:

```
*      *
**    **
***  ***
********
********
***  ***
**    **
*      *
```

WEEK 02 TOPIC: Bitwise operators and loops

CLASS NOTES: 05

1. Bitwise operators
    - Explanation: & AND, | OR, ^ XOR, ~ NOT, << left shift, >> right shift. No specific code.
2. Left and right shift operators
    - Example:

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

int main() {
```

```
    int a = 5;
    cout << (a << 1) << endl; // 10
    cout << (a >> 1); // 2
    return 0;
}
```

Output:

```
10
2
```

3. Pre/post increment and decrement operators
   - Example:

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

int main() {
    int a = 5;
    cout << a++ << endl; // 5
    cout << ++a; // 7
    return 0;
}
```

Output:

```
5
7
```

4. Break and continue keyword
   - Example:

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

int main() {
    for(int i = 1; i <= 5; i++) {
        if(i == 3) continue;
        if(i == 5) break;
        cout << i << " ";
    }
    return 0;
}
```

Output: `1 2 4`

5. Variable scoping

- Example:

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

int main() {
    int a = 10;
    {
        int a = 20;
        cout << a << endl; // 20
    }
    cout << a; // 10
    return 0;
}
```

Output:

```
20
10
```

6. Operator precedence table
   - Explanation: No code, refer to C++ docs.

CLASS HOMEWORK: 05

1. All homework programs
   - Assumption: The ones from previous, already covered.
2. Why global variables are bad practice
   - Explanation: Global variables can lead to name clashes, difficult debugging, unexpected side effects, and violate encapsulation. Use local variables or pass as parameters.

WEEK 02 TOPIC: Functions

CLASS NOTES: 06

1. Write a function to print sum of 3 numbers

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

void printSum(int a, int b, int c) {
    cout << a + b + c;
}

int main() {
    printSum(1,2,3);
```

```
    return 0;
}
```

Output: `6`

2. Write a function to return sum of 3 numbers

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

int returnSum(int a, int b, int c) {
    return a + b + c;
}

int main() {
    cout << returnSum(1,2,3);
    return 0;
}
```

Output: `6`

3. Find maximum of three numbers

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

int getMax(int a, int b, int c) {
    return max(a, max(b, c));
}

int main() {
    cout << getMax(1,3,2);
    return 0;
}
```

Output: `3`

4. Counting from 1 to N

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

void count1ToN(int n) {
    for(int i = 1; i <= n; i++) cout << i << " ";
}

int main() {
    count1ToN(5);
```

```
    return 0;
}
```

Output: `1 2 3 4 5`

5. Check prime or not prime number

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

bool isPrime(int n) {
    for(int i = 2; i * i <= n; i++) {
        if(n % i == 0) return false;
    }
    return true;
}

int main() {
    cout << (isPrime(7) ? "Prime" : "Not Prime");
    return 0;
}
```

Output: `Prime`

6. Check number is even or odd

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

string evenOdd(int n) {
    return n % 2 == 0 ? "Even" : "Odd";
}

int main() {
    cout << evenOdd(4);
    return 0;
}
```

Output: `Even`

7. Sum of all numbers upto 1 to N

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

int sum1ToN(int n) {
    return n * (n + 1) / 2;
}
```

```
int main() {
    cout << sum1ToN(5);
    return 0;
}
```

Output: `15`

8. Sum of all even numbers upto 1 to N

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

int sumEven1ToN(int n) {
    int sum = 0;
    for(int i = 2; i <= n; i += 2) sum += i;
    return sum;
}

int main() {
    cout << sumEven1ToN(6);
    return 0;
}
```

Output: `12`

CLASS HOMEWORK: 06

1. Function to find area of circle

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

double areaCircle(int r) {
    return 3.14 * r * r;
}

int main() {
    cout << areaCircle(5);
    return 0;
}
```

Output: `78.5`

2. Function to find factorial of a number

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

```
long long factorial(int n) {
    long long fact = 1;
    for(int i = 1; i <= n; i++) fact *= i;
    return fact;
}

int main() {
    cout << factorial(5);
    return 0;
}
```

Output: `120`

3. Print all prime numbers from 1 to N

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

void printPrimes(int n) {
    for(int i = 2; i <= n; i++) {
        bool isPrime = true;
        for(int j = 2; j * j <= i; j++) {
            if(i % j == 0) isPrime = false;
        }
        if(isPrime) cout << i << " ";
    }
}

int main() {
    printPrimes(20);
    return 0;
}
```

Output: `2 3 5 7 11 13 17 19`

4. Print all digits of an integer

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

void printDigits(int n) {
    while(n > 0) {
        cout << n % 10 << " ";
        n /= 10;
    }
}
```

```
int main() {
    printDigits(1234);
    return 0;
}
```

Output: `4 3 2 1`

5. Creating a number using digits

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

int createNumber(vector<int> digits) {
    int num = 0;
    for(int d : digits) {
        num = num * 10 + d;
    }
    return num;
}

int main() {
    vector<int> digits = {1,2,3,4};
    cout << createNumber(digits);
    return 0;
}
```

Output: `1234`

6. Print binary representation of a decimal number

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

void printBinary(int n) {
    string bin = "";
    while(n > 0) {
        bin = to_string(n % 2) + bin;
        n /= 2;
    }
    cout << bin;
}

int main() {
    printBinary(13);
    return 0;
}
```

Output: `1101`

7. Convert KM into Miles

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

double kmToMiles(double km) {
    return km * 0.621371;
}

int main() {
    cout << fixed << setprecision(2) << kmToMiles(5);
    return 0;
}
```

Output: `3.11`

8. Convert farenheit to celcius

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

double fToC(double f) {
    return (f - 32) * 5 / 9;
}

int main() {
    cout << fToC(98.6);
    return 0;
}
```

Output: `37`

9. Count all set bits of a number

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

int countSetBits(int n) {
    int count = 0;
    while(n > 0) {
        count += n & 1;
        n >>= 1;
    }
    return count;
}
```

```
int main() {
    cout << countSetBits(13); // 1101
    return 0;
}
```

Output: `3`

10. Check even/odd using bitwise operator

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

string evenOddBitwise(int n) {
    return (n & 1) == 0 ? "Even" : "Odd";
}

int main() {
    cout << evenOddBitwise(4);
    return 0;
}
```

Output: `Even`

WEEK 03 TOPIC: Arrays - Level 1

CLASS NOTES: 07

1. Count 0's and 1's in an array

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

void countZeroOne(vector<int> arr) {
    int zero = 0, one = 0;
    for(int num : arr) {
        if(num == 0) zero++;
        else if(num == 1) one++;
    }
    cout << "Zeros: " << zero << " Ones: " << one;
}

int main() {
    vector<int> arr = {0,1,0,1,1};
    countZeroOne(arr);
    return 0;
}
```

Output: `Zeros: 2 Ones: 3`

2. Minimum and maximum number in an array

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

pair<int, int> minMax(vector<int> arr) {
    int mn = *min_element(arr.begin(), arr.end());
    int mx = *max_element(arr.begin(), arr.end());
    return {mn, mx};
}

int main() {
    vector<int> arr = {3,1,4,2};
    auto p = minMax(arr);
    cout << "Min: " << p.first << " Max: " << p.second;
    return 0;
}
```

Output: `Min: 1 Max: 4`

3. Reverse an array (Two pointer approach)

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

void reverseArray(vector<int>& arr) {
    int i = 0, j = arr.size() - 1;
    while(i < j) {
        swap(arr[i++], arr[j--]);
    }
}

int main() {
    vector<int> arr = {1,2,3,4};
    reverseArray(arr);
    for(int num : arr) cout << num << " ";
    return 0;
}
```

Output: `4 3 2 1`

4. Extreme print an array

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

```
void extremePrint(vector<int> arr) {
    int i = 0, j = arr.size() - 1;
    while(i <= j) {
        if(i == j) cout << arr[i];
        else cout << arr[i] << " " << arr[j] << " ";
        i++;
        j--;
    }
}

int main() {
    vector<int> arr = {1,2,3,4,5};
    extremePrint(arr);
    return 0;
}
```

Output: `1 5 2 4 3`

CLASS HOMEWORK: 07

No homework

WEEK 03 TOPIC: Arrays - Level 2

CLASS NOTES: 08

1. Find unique element
    - Approach: XOR all elements (cancels duplicates)

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

int findUnique(vector<int> arr) {
    int xorAll = 0;
    for(int num : arr) xorAll ^= num;
    return xorAll;
}

int main() {
    vector<int> arr = {1,2,1,3,2};
    cout << findUnique(arr);
    return 0;
}
```

Output: `3`

2. Print all pairs

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

void printPairs(vector<int> arr) {
    for(int i = 0; i < arr.size(); i++) {
        for(int j = i+1; j < arr.size(); j++) {
            cout << "(" << arr[i] << "," << arr[j] << ") ";
        }
    }
}

int main() {
    vector<int> arr = {1,2,3};
    printPairs(arr);
    return 0;
}
```

Output: `(1,2) (1,3) (2,3)`

3. Print all triplets

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

void printTriplets(vector<int> arr) {
    for(int i = 0; i < arr.size(); i++) {
        for(int j = i+1; j < arr.size(); j++) {
            for(int k = j+1; k < arr.size(); k++) {
                cout << "(" << arr[i] << "," << arr[j] << "," << arr[k] << ") ";
            }
        }
    }
}

int main() {
    vector<int> arr = {1,2,3,4};
    printTriplets(arr);
    return 0;
}
```

Output: `(1,2,3) (1,2,4) (1,3,4) (2,3,4)`

4. Sort 0's and 1's
    - Approach: Two pointer

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

void sortZeroOne(vector<int>& arr) {
    int i = 0, j = arr.size() - 1;
    while(i < j) {
        if(arr[i] == 1 && arr[j] == 0) swap(arr[i], arr[j]);
        if(arr[i] == 0) i++;
        if(arr[j] == 1) j--;
    }
}

int main() {
    vector<int> arr = {0,1,0,1,1,0};
    sortZeroOne(arr);
    for(int num : arr) cout << num << " ";
    return 0;
}
```

Output: `0 0 0 1 1 1`

CLASS HOMEWORK: 08

1. Program 05: Shift array's element by one (Right to left)
    - Approach: Rotate right by 1

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

void shiftRightOne(vector<int>& arr) {
    int last = arr.back();
    for(int i = arr.size()-1; i >= 1; i--) arr[i] = arr[i-1];
    arr[0] = last;
}

int main() {
    vector<int> arr = {1,2,3,4};
    shiftRightOne(arr);
    for(int num : arr) cout << num << " ";
    return 0;
}
```

Output: `4 1 2 3`

2. Program 06: Shift array's element by two (Right to left)

- Approach: Rotate right by 2

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

void shiftRightTwo(vector<int>& arr) {
    int n = arr.size();
    reverse(arr.begin(), arr.end());
    reverse(arr.begin(), arr.begin()+2);
    reverse(arr.begin()+2, arr.end());
}

int main() {
    vector<int> arr = {1,2,3,4,5};
    shiftRightTwo(arr);
    for(int num : arr) cout << num << " ";
    return 0;
}
```

Output: `4 5 1 2 3`

3. HW 01: Shift array's element by one (Left to right)
   - Approach: Rotate left by 1

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

void shiftLeftOne(vector<int>& arr) {
    int first = arr[0];
    for(int i = 0; i < arr.size()-1; i++) arr[i] = arr[i+1];
    arr.back() = first;
}

int main() {
    vector<int> arr = {1,2,3,4};
    shiftLeftOne(arr);
    for(int num : arr) cout << num << " ";
    return 0;
}
```

Output: `2 3 4 1`

WEEK 03 TOPIC: Arrays - Level 3

CLASS NOTES: 09

1. Linear Search in 2D-Array

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

bool linearSearch2D(vector<vector<int>> mat, int target) {
    for(auto row : mat) {
        for(int num : row) {
            if(num == target) return true;
        }
    }
    return false;
}

int main() {
    vector<vector<int>> mat = {{1,2},{3,4}};
    cout << (linearSearch2D(mat, 3) ? "Found" : "Not Found");
    return 0;
}
```

Output: `Found`

2. Maximum and Minimum in 2D-Array

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

pair<int, int> minMax2D(vector<vector<int>> mat) {
    int mn = INT_MAX, mx = INT_MIN;
    for(auto row : mat) {
        for(int num : row) {
            mn = min(mn, num);
            mx = max(mx, num);
        }
    }
    return {mn, mx};
}

int main() {
    vector<vector<int>> mat = {{1,2},{3,4}};
    auto p = minMax2D(mat);
    cout << "Min: " << p.first << " Max: " << p.second;
    return 0;
}
```

Output: `Min: 1 Max: 4`

3. Print row wise and column wise sum of 2D-Array

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

void rowColumnSum(vector<vector<int>> mat) {
    cout << "Row sums: ";
    for(auto row : mat) {
        int sum = 0;
        for(int num : row) sum += num;
        cout << sum << " ";
    }
    cout << endl << "Column sums: ";
    for(int j = 0; j < mat[0].size(); j++) {
        int sum = 0;
        for(int i = 0; i < mat.size(); i++) sum += mat[i][j];
        cout << sum << " ";
    }
}

int main() {
    vector<vector<int>> mat = {{1,2},{3,4}};
    rowColumnSum(mat);
    return 0;
}
```

Output: `Row sums: 3 7 Column sums: 4 6`

4. Sum of principal diagonal elements of a matrix

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

int principalDiagonalSum(vector<vector<int>> mat) {
    int sum = 0;
    for(int i = 0; i < mat.size(); i++) sum += mat[i][i];
    return sum;
}

int main() {
    vector<vector<int>> mat = {{1,2},{3,4}};
    cout << principalDiagonalSum(mat);
    return 0;
}
```

Output: `5`

5. Transpose of a matrix

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

void transpose(vector<vector<int>>& mat) {
    int n = mat.size();
    for(int i = 0; i < n; i++) {
        for(int j = i+1; j < n; j++) {
            swap(mat[i][j], mat[j][i]);
        }
    }
}

int main() {
    vector<vector<int>> mat = {{1,2},{3,4}};
    transpose(mat);
    for(auto row : mat) {
        for(int num : row) cout << num << " ";
        cout << endl;
    }
    return 0;
}
```

Output:

```
1 3
2 4
```

6. Jagged Array
   - Example creation:

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

int main() {
    vector<vector<int>> jagged = {{1}, {2,3}, {4,5,6}};
    for(auto row : jagged) {
        for(int num : row) cout << num << " ";
        cout << endl;
    }
    return 0;
}
```

Output:

```
1
2 3
4 5 6
```

CLASS HOMEWORK: 09

1. Column wise sum of a 2D-Array
   - Already in class, see above.
2. Sum of secondary diagonal elements

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

int secondaryDiagonalSum(vector<vector<int>> mat) {
    int sum = 0, n = mat.size();
    for(int i = 0; i < n; i++) sum += mat[i][n-1-i];
    return sum;
}

int main() {
    vector<vector<int>> mat = {{1,2},{3,4}};
    cout << secondaryDiagonalSum(mat);
    return 0;
}
```

Output: `5`

WEEK 03 TOPIC: Arrays - Extra Class

CLASS NOTES: 10

1. Moving All Negative Number to the Left Side of an Array
   - Approach: Two pointer

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

void moveNegativeLeft(vector<int>& arr) {
    int i = 0, j = arr.size() - 1;
    while(i <= j) {
        if(arr[i] < 0) i++;
        else if(arr[j] >= 0) j--;
        else swap(arr[i++], arr[j--]);
```

```
    }
}

int main() {
    vector<int> arr = {-1,2,-3,4,-5};
    moveNegativeLeft(arr);
    for(int num : arr) cout << num << " ";
    return 0;
}
```

Output: `-5 -3 -1 4 2`

2. Sort Colors (Leetcode-75)
    - Approach: Dutch National Flag (three pointers)

```
class Solution {
public:
    void sortColors(vector<int>& nums) {
        int low = 0, mid = 0, high = nums.size() - 1;
        while(mid <= high) {
            if(nums[mid] == 0) swap(nums[low++], nums[mid++]);
            else if(nums[mid] == 1) mid++;
            else swap(nums[mid], nums[high--]);
        }
    }
};
```

Example: Input [2,0,2,1,1,0], Output [0,0,1,1,2,2]

3. Rotate Array (Leetcode-189)
    - Approach: Reverse method

```
class Solution {
public:
    void rotate(vector<int>& nums, int k) {
        int n = nums.size();
        k %= n;
        reverse(nums.begin(), nums.end());
        reverse(nums.begin(), nums.begin() + k);
        reverse(nums.begin() + k, nums.end());
    }
};
```

Example: Input [1,2,3,4,5], k=2, Output [4,5,1,2,3]

4. Missing Number (Leetcode-268)
    - Approach: XOR

```
class Solution {
public:
    int missingNumber(vector<int>& nums) {
        int xorAll = 0;
        for(int i = 0; i <= nums.size(); i++) xorAll ^= i;
        for(int num : nums) xorAll ^= num;
        return xorAll;
    }
};
```

Example: Input [0,1,3], Output 2

5. Row with maximum ones (VVImp Leetcode-2643)
    - Approach: Traverse and count

```
class Solution {
public:
    vector<int> rowAndMaximumOnes(vector<vector<int>>& mat) {
        int maxCount = 0, row = 0;
        for(int i = 0; i < mat.size(); i++) {
            int count = 0;
            for(int num : mat[i]) count += num;
            if(count > maxCount) {
                maxCount = count;
                row = i;
            }
        }
        return {row, maxCount};
    }
};
```

Example: Input [[0,1],[1,0]], Output [0,1] or [1,1] (smallest row if tie)

6. Rotate Image by 90 degree (VVImp Leetcode-48)
    - Approach: Transpose then reverse rows

```
class Solution {
public:
    void rotate(vector<vector<int>>& matrix) {
        int n = matrix.size();
        for(int i = 0; i < n; i++) {
            for(int j = i; j < n; j++) {
                swap(matrix[i][j], matrix[j][i]);
            }
        }
        for(int i = 0; i < n; i++) reverse(matrix[i].begin(), matrix[i].end());
```

```
    }
};
```

Example: Input [[1,2],[3,4]], Output [[3,1],[4,2]]

CLASS HOMEWORK: 10

1. Re-arrange array elements (Leetcode-2149)
    - Approach: Two pointers for positive and negative

```
class Solution {
public:
    vector<int> rearrangeArray(vector<int>& nums) {
        vector<int> pos, neg;
        for(int num : nums) {
            if(num > 0) pos.push_back(num);
            else neg.push_back(num);
        }
        vector<int> result;
        for(int i = 0; i < pos.size(); i++) {
            result.push_back(pos[i]);
            result.push_back(neg[i]);
        }
        return result;
    }
};
```

Example: Input [3,1,-2,-5], Output [3,-2,1,-5]

2. Find Pivot Index (Leetcode-724)
    - Approach: Prefix sum

```
class Solution {
public:
    int pivotIndex(vector<int>& nums) {
        int total = 0;
        for(int num : nums) total += num;
        int left = 0;
        for(int i = 0; i < nums.size(); i++) {
            if(left == total - left - nums[i]) return i;
            left += nums[i];
        }
        return -1;
    }
};
```

Example: Input [1,7,3,6,5,6], Output 3

3. Find Duplicate Number (Leetcode-287)
   - Approach: Floyd's cycle detection

```
class Solution {
public:
    int findDuplicate(vector<int>& nums) {
        int slow = nums[0], fast = nums[0];
        do {
            slow = nums[slow];
            fast = nums[nums[fast]];
        } while(slow != fast);
        slow = nums[0];
        while(slow != fast) {
            slow = nums[slow];
            fast = nums[fast];
        }
        return slow;
    }
};
```

Example: Input [1,3,4,2,2], Output 2

4. Missing Element From An Array With Duplicates (GFG)
   - Approach: XOR with expected range
   - Assumption: Array with 1 to n, one missing, one duplicate

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

int findMissingWithDup(vector<int> arr, int n) {
    int xorAll = 0;
    for(int i = 1; i <= n; i++) xorAll ^= i;
    for(int num : arr) xorAll ^= num;
    return xorAll;
}

int main() {
    vector<int> arr = {1,2,3,2};
    cout << findMissingWithDup(arr, 4);
    return 0;
}
```

Output: `4` (if missing 4, dup 2)

5. Find First Repeating Element (GFG)
    - Approach: Unordered map

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

int firstRepeating(vector<int> arr) {
    unordered_map<int, int> mp;
    for(int num : arr) mp[num]++;
    for(int num : arr) if(mp[num] > 1) return num;
    return -1;
}

int main() {
    vector<int> arr = {1,5,3,4,3,5};
    cout << firstRepeating(arr);
    return 0;
}
```

Output: `3`

6. Common Element in 3 Sorted Array (GFG)
    - Approach: Three pointers

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

vector<int> commonElements(vector<int> a, vector<int> b, vector<int> c) {
    vector<int> result;
    int i = 0, j = 0, k = 0;
    while(i < a.size() && j < b.size() && k < c.size()) {
        if(a[i] == b[j] && b[j] == c[k]) {
            result.push_back(a[i]);
            i++; j++; k++;
        } else if(a[i] < b[j]) i++;
        else if(b[j] < c[k]) j++;
        else k++;
    }
    return result;
}

int main() {
    vector<int> a = {1,5,10}, b = {3,5,10}, c = {5,10,20};
    auto res = commonElements(a,b,c);
```

```
    for(int num : res) cout << num << " ";
    return 0;
}
```

Output: `5 10`

7. Wave Print A Matrix (GFG)
   - Approach: Even rows left to right, odd right to left

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

void wavePrint(vector<vector<int>> mat) {
    for(int i = 0; i < mat.size(); i++) {
        if(i % 2 == 0) for(int j = 0; j < mat[0].size(); j++) cout << mat[i][j]
<< " ";
        else for(int j = mat[0].size()-1; j >= 0; j--) cout << mat[i][j] << " ";
    }
}

int main() {
    vector<vector<int>> mat = {{1,2,3},{4,5,6}};
    wavePrint(mat);
    return 0;
}
```

Output: `1 2 3 6 5 4`

8. Spiral Print A Matrix (Leetcode-54)
   - Approach: Four pointers for boundaries

```
class Solution {
public:
    vector<int> spiralOrder(vector<vector<int>>& matrix) {
        vector<int> result;
        int left = 0, right = matrix[0].size() - 1;
        int top = 0, bottom = matrix.size() - 1;
        while(left <= right && top <= bottom) {
            for(int j = left; j <= right; j++) result.push_back(matrix[top][j]);
            top++;
            for(int i = top; i <= bottom; i++) result.push_back(matrix[i]
[right]);
            right--;
            if(top <= bottom) {
                for(int j = right; j >= left; j--)
result.push_back(matrix[bottom][j]);
```

```
                bottom--;
            }
            if(left <= right) {
                for(int i = bottom; i >= top; i--) result.push_back(matrix[i]
[left]);
                left++;
            }
        }
        return result;
    }
};
```

Example: Input [[1,2,3],[4,5,6],[7,8,9]], Output [1,2,3,6,9,8,7,4,5]

9. Factorial of A Large Number (GFG)
    - Approach: Big integer using vector

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

vector<int> largeFactorial(int n) {
    vector<int> result = {1};
    for(int i = 2; i <= n; i++) {
        int carry = 0;
        for(int &digit : result) {
            int prod = digit * i + carry;
            digit = prod % 10;
            carry = prod / 10;
        }
        while(carry) {
            result.push_back(carry % 10);
            carry /= 10;
        }
    }
    reverse(result.begin(), result.end());
    return result;
}

int main() {
    auto res = largeFactorial(10);
    for(int d : res) cout << d;
    return 0;
}
```

Output: `3628800`

10. Key Pair/Two Sum (GFG and Leetcode-1)
    - Approach: Unordered map

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map<int, int> mp;
        for(int i = 0; i < nums.size(); i++) {
            if(mp.count(target - nums[i]) ) return {mp[target - nums[i]], i};
            mp[nums[i]] = i;
        }
        return {};
    }
};
```

    Example: Input [2,7,11,15], target=9, Output [0,1]

11. Remove Duplicates From Sorted Array (Leetcode-26)
    - Approach: Two pointers

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        int i = 0;
        for(int j = 1; j < nums.size(); j++) {
            if(nums[j] != nums[i]) nums[++i] = nums[j];
        }
        return i + 1;
    }
};
```

    Example: Input [1,1,2], Output 2 (array [1,2])

12. Maximum Average Subarray 1 (Leetcode-643)
    - Approach: Sliding window fixed size

```
class Solution {
public:
    double findMaxAverage(vector<int>& nums, int k) {
        double sum = 0;
        for(int i = 0; i < k; i++) sum += nums[i];
        double maxAvg = sum / k;
        for(int i = k; i < nums.size(); i++) {
            sum += nums[i] - nums[i-k];
            maxAvg = max(maxAvg, sum / k);
        }
```

```
        return maxAvg;
    }
};
```

Example: Input [1,12,-5,-6,50,3], k=4, Output 12.75

13. Find Pivot Index with prefix sum approach (Leetcode-724)
    - Already in homework 2, see above.
14. Missing Number with XOR operator (Leetcode-268)
    - Already in class 4, see above.
15. Add two numbers represented by two array (GFG)
    - Approach: Add from end, handle carry

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

string addTwoArrays(vector<int> a, vector<int> b) {
    string result = "";
    int i = a.size() - 1, j = b.size() - 1, carry = 0;
    while(i >= 0 || j >= 0 || carry) {
        int sum = carry;
        if(i >= 0) sum += a[i--];
        if(j >= 0) sum += b[j--];
        result = to_string(sum % 10) + result;
        carry = sum / 10;
    }
    return result;
}

int main() {
    vector<int> a = {1,2}, b = {3,4};
    cout << addTwoArrays(a, b);
    return 0;
}
```

Output: `46`

WEEK 04 TOPIC: Searching and Sorting - Level 1

CLASS NOTES: 11

1. Linear search

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

```
int linearSearch(vector<int> arr, int target) {
    for(int i = 0; i < arr.size(); i++) if(arr[i] == target) return i;
    return -1;
}

int main() {
    vector<int> arr = {4,2,5,1};
    cout << linearSearch(arr, 5);
    return 0;
}
```

Output: `2`

2. Binary search (MONOTONIC ARRAY)

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

int binarySearch(vector<int> arr, int target) {
    int low = 0, high = arr.size() - 1;
    while(low <= high) {
        int mid = low + (high - low) / 2;
        if(arr[mid] == target) return mid;
        else if(arr[mid] < target) low = mid + 1;
        else high = mid - 1;
    }
    return -1;
}

int main() {
    vector<int> arr = {1,2,3,4,5};
    cout << binarySearch(arr, 3);
    return 0;
}
```

Output: `2`

3. Find first occurrence of a number in sorted array

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

int firstOccurrence(vector<int> arr, int target) {
    int low = 0, high = arr.size() - 1, ans = -1;
    while(low <= high) {
        int mid = low + (high - low) / 2;
```

```
        if(arr[mid] == target) {
            ans = mid;
            high = mid - 1;
        } else if(arr[mid] < target) low = mid + 1;
        else high = mid - 1;
    }
    return ans;
}

int main() {
    vector<int> arr = {1,2,2,3,3};
    cout << firstOccurrence(arr, 3);
    return 0;
}
```

Output: `3`

4. Find last occurrence of a number in sorted array

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

int lastOccurrence(vector<int> arr, int target) {
    int low = 0, high = arr.size() - 1, ans = -1;
    while(low <= high) {
        int mid = low + (high - low) / 2;
        if(arr[mid] == target) {
            ans = mid;
            low = mid + 1;
        } else if(arr[mid] < target) low = mid + 1;
        else high = mid - 1;
    }
    return ans;
}

int main() {
    vector<int> arr = {1,2,2,3,3};
    cout << lastOccurrence(arr, 2);
    return 0;
}
```

Output: `2`

5. Find total occurrence of a number in sorted array

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

```
int totalOccurrence(vector<int> arr, int target) {
    int first = firstOccurrence(arr, target); // from above
    int last = lastOccurrence(arr, target); // from above
    if(first == -1) return 0;
    return last - first + 1;
}

int main() {
    vector<int> arr = {1,2,2,3,3};
    cout << totalOccurrence(arr, 2);
    return 0;
}
```

Output: `2`

6. Find missing element in sorted array (GFG)
    - Approach: Binary search for difference

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

int findMissingSorted(vector<int> arr) {
    int low = 0, high = arr.size() - 1;
    while(low <= high) {
        int mid = low + (high - low) / 2;
        if(arr[mid] != mid + 1) {
            if(mid == 0 || arr[mid-1] == mid) return mid + 1;
            high = mid - 1;
        } else low = mid + 1;
    }
    return -1;
}

int main() {
    vector<int> arr = {1,2,3,5};
    cout << findMissingSorted(arr);
    return 0;
}
```

Output: `4`

7. Peak element/index in a mountain array (Leetcode-852)
    - Approach: Binary search for peak

```
class Solution {
public:
    int peakIndexInMountainArray(vector<int>& arr) {
        int low = 1, high = arr.size() - 2;
        while(low <= high) {
            int mid = low + (high - low) / 2;
            if(arr[mid] > arr[mid-1] && arr[mid] > arr[mid+1]) return mid;
            else if(arr[mid] < arr[mid+1]) low = mid + 1;
            else high = mid - 1;
        }
        return -1;
    }
};
```

Example: Input [0,10,5,2], Output 1

CLASS HOMEWORK: 11

1. Find pivot element (LeftSum equals to RightSum)(Leetcode-724)
    - Already in previous, see Leetcode-724

WEEK 04 TOPIC: Searching and Sorting - Level 2

CLASS NOTES: 12

1. Find pivot element index from sorted and rotated array
    - Approach: Binary search for min element index

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

int findPivotRotated(vector<int> arr) {
    int low = 0, high = arr.size() - 1;
    while(low < high) {
        int mid = low + (high - low) / 2;
        if(arr[mid] > arr[high]) low = mid + 1;
        else high = mid;
    }
    return low;
}

int main() {
    vector<int> arr = {4,5,1,2,3};
    cout << findPivotRotated(arr);
```

```
    return 0;
}
```

Output: `2`

2. Search in a rotated and sorted array (Leetcode-33)
    - Approach: Binary search with rotation handling

```
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int low = 0, high = nums.size() - 1;
        while(low <= high) {
            int mid = low + (high - low) / 2;
            if(nums[mid] == target) return mid;
            if(nums[low] <= nums[mid]) {
                if(nums[low] <= target && target < nums[mid]) high = mid - 1;
                else low = mid + 1;
            } else {
                if(nums[mid] < target && target <= nums[high]) low = mid + 1;
                else high = mid - 1;
            }
        }
        return -1;
    }
};
```

Example: Input [4,5,6,7,0,1,2], target=0, Output 4

3. Sqrt of X (Leetcode-69)
    - Approach: Binary search

```
class Solution {
public:
    int mySqrt(int x) {
        long low = 0, high = x;
        while(low <= high) {
            long mid = low + (high - low) / 2;
            long sq = mid * mid;
            if(sq == x) return mid;
            else if(sq < x) low = mid + 1;
            else high = mid - 1;
        }
        return high;
    }
};
```

Example: Input 8, Output 2

4. Binary search in 2D array (Leetcode-74)
    - Approach: Treat as 1D, binary search

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int m = matrix.size(), n = matrix[0].size();
        int low = 0, high = m*n - 1;
        while(low <= high) {
            int mid = low + (high - low) / 2;
            int row = mid / n, col = mid % n;
            if(matrix[row][col] == target) return true;
            else if(matrix[row][col] < target) low = mid + 1;
            else high = mid - 1;
        }
        return false;
    }
};
```

Example: Input [[1,3,5],[10,11,16],[23,30,34]], target=3, Output true

CLASS HOMEWORK: 12

1. Find sqrt of X upto N decimal place
    - Approach: Binary search + decimal adjustment

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

double sqrtDecimal(double x, int precision) {
    double low = 0, high = x, ans = 0;
    while(low <= high) {
        double mid = low + (high - low) / 2;
        if(mid * mid <= x) {
            ans = mid;
            low = mid + 1e-6;
        } else high = mid - 1e-6;
    }
    // Adjust for precision
    double factor = 1;
    for(int i = 0; i < precision; i++) {
        factor /= 10;
        for(double j = ans; j*j < x; j += factor) ans = j;
```

```
    }
    return ans;
}

int main() {
    cout << fixed << setprecision(3) << sqrtDecimal(10, 3);
    return 0;
}
```

Output: `3.162`

WEEK 04 TOPIC: Searching and Sorting - Level 3

CLASS NOTES: 13

1. Divide two number using Binary search without using any / and % operator
    - Approach: Binary search for quotient

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

int divideBinary(int dividend, int divisor) {
    if(divisor == 0) return INT_MAX;
    int low = 0, high = abs(dividend);
    int ans = 0;
    while(low <= high) {
        int mid = low + (high - low) / 2;
        if(abs(mid * divisor) <= abs(dividend)) {
            ans = mid;
            low = mid + 1;
        } else high = mid - 1;
    }
    return (dividend < 0) ^ (divisor < 0) ? -ans : ans;
}

int main() {
    cout << divideBinary(10, 3);
    return 0;
}
```

Output: `3`

2. Binary search on nearly sorted array
    - Approach: Check mid, mid-1, mid+1

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

int binaryNearlySorted(vector<int> arr, int target) {
    int low = 0, high = arr.size() - 1;
    while(low <= high) {
        int mid = low + (high - low) / 2;
        if(arr[mid] == target) return mid;
        if(mid > 0 && arr[mid-1] == target) return mid-1;
        if(mid < arr.size()-1 && arr[mid+1] == target) return mid+1;
        if(arr[mid] < target) low = mid + 2;
        else high = mid - 2;
    }
    return -1;
}

int main() {
    vector<int> arr = {10,3,40,20,50,80,70};
    cout << binaryNearlySorted(arr, 40);
    return 0;
}
```

Output: `2`

3. Find the Number Occurring Odd Number of Times (Leetcode-540)
    - Approach: Binary search for odd count

```cpp
class Solution {
public:
    int singleNonDuplicate(vector<int>& nums) {
        int low = 0, high = nums.size() - 1;
        while(low < high) {
            int mid = low + (high - low) / 2;
            if(mid % 2 == 1) mid--;
            if(nums[mid] == nums[mid+1]) low = mid + 2;
            else high = mid;
        }
        return nums[low];
    }
};
```

Example: Input [1,1,2,3,3,4,4,8,8], Output 2

CLASS HOMEWORK: 13

1. K-Diff Pairs in An Array (Leetcode-532)
    - Approach: Sort and two pointers

```
class Solution {
public:
    int findPairs(vector<int>& nums, int k) {
        sort(nums.begin(), nums.end());
        int count = 0, i = 0, j = 1;
        while(j < nums.size()) {
            int diff = nums[j] - nums[i];
            if(diff == k) {
                count++;
                i++;
                j++;
                while(j < nums.size() && nums[j] == nums[j-1]) j++;
            } else if(diff < k) j++;
            else i++;
            if(i == j) j++;
        }
        return count;
    }
};
```

    Example: Input [3,1,4,1,5], k=2, Output 2

2. Find K-Closest Element (Leetcode-658)
    - Approach: Binary search for starting point

```
class Solution {
public:
    vector<int> findClosestElements(vector<int>& arr, int k, int x) {
        int low = 0, high = arr.size() - k;
        while(low < high) {
            int mid = low + (high - low) / 2;
            if(x - arr[mid] > arr[mid + k] - x) low = mid + 1;
            else high = mid;
        }
        return vector<int>(arr.begin() + low, arr.begin() + low + k);
    }
};
```

    Example: Input [1,2,3,4,5], k=4, x=3, Output [1,2,3,4]

3. Exponential Search (Concept)
    - Explanation: Find range by exponential growth, then binary search. No code, conceptual.

4. Unbounded Binary Search (Concept)
    - Explanation: For unbounded array, double range until target in range, then binary. No code.
5. Book Allocation Problem (GFG & Code studio)
    - Approach: Binary search on max pages

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

bool isPossible(vector<int> books, int students, int maxPages) {
    int count = 1, sum = 0;
    for(int p : books) {
        if(p > maxPages) return false;
        if(sum + p > maxPages) {
            count++;
            sum = p;
            if(count > students) return false;
        } else sum += p;
    }
    return true;
}

int bookAllocation(vector<int> books, int students) {
    int low = 0, high = 0;
    for(int p : books) {
        low = max(low, p);
        high += p;
    }
    int ans = -1;
    while(low <= high) {
        int mid = low + (high - low) / 2;
        if(isPossible(books, students, mid)) {
            ans = mid;
            high = mid - 1;
        } else low = mid + 1;
    }
    return ans;
}

int main() {
    vector<int> books = {12,34,67,90};
    cout << bookAllocation(books, 2);
```

```
    return 0;
}
```

Output: `113`

6. Painters Partition Problem (GFG & Code studio)
    - Similar to book allocation, replace books with boards, students with painters.
7. Aggressive Cows (GFG & Code studio)
    - Approach: Binary search on min distance

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

bool isPossible(vector<int> stalls, int cows, int minDist) {
    int count = 1, last = stalls[0];
    for(int s : stalls) {
        if(s - last >= minDist) {
            count++;
            last = s;
            if(count >= cows) return true;
        }
    }
    return false;
}

int aggressiveCows(vector<int> stalls, int cows) {
    sort(stalls.begin(), stalls.end());
    int low = 1, high = stalls.back() - stalls[0];
    int ans = 0;
    while(low <= high) {
        int mid = low + (high - low) / 2;
        if(isPossible(stalls, cows, mid)) {
            ans = mid;
            low = mid + 1;
        } else high = mid - 1;
    }
    return ans;
}

int main() {
    vector<int> stalls = {1,2,4,8,9};
    cout << aggressiveCows(stalls, 3);
    return 0;
}
```

Output: `3`

8. EKO SPOJ
    - Approach: Binary search on height
    - Code similar to aggressive cows, but for wood collection.
9. PRATA SPOJ
    - Approach: Binary search on time, check if cooks can make enough
    - Code similar to book allocation.
10. Find SQRT of Integer N using Binary Search with K point decimal precision.
    - Already in homework 12, see sqrt decimal.
11. Divide using Binary Search with K point decimal precision.
    - Similar to divide binary, add precision loop.
12. Majority Element (Leetcode-169)
    - Approach: Boyer-Moore Voting

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        int candidate = nums[0], count = 1;
        for(int i = 1; i < nums.size(); i++) {
            if(nums[i] == candidate) count++;
            else {
                count--;
                if(count == 0) {
                    candidate = nums[i];
                    count = 1;
                }
            }
        }
        return candidate;
    }
};
```

Example: Input [2,2,1,1,1,2,2], Output 2

Sortings

1. BUBBLE SORT

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

```cpp
void bubbleSort(vector<int>& arr) {
    int n = arr.size();
    for(int i = 0; i < n-1; i++) {
        for(int j = 0; j < n-i-1; j++) {
            if(arr[j] > arr[j+1]) swap(arr[j], arr[j+1]);
        }
    }
}

int main() {
    vector<int> arr = {5,4,3,2,1};
    bubbleSort(arr);
    for(int num : arr) cout << num << " ";
    return 0;
}
```

Output: `1 2 3 4 5`

2. SELECTION SORT

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

void selectionSort(vector<int>& arr) {
    int n = arr.size();
    for(int i = 0; i < n-1; i++) {
        int minIdx = i;
        for(int j = i+1; j < n; j++) {
            if(arr[j] < arr[minIdx]) minIdx = j;
        }
        swap(arr[i], arr[minIdx]);
    }
}

int main() {
    vector<int> arr = {5,4,3,2,1};
    selectionSort(arr);
    for(int num : arr) cout << num << " ";
    return 0;
}
```

Output: `1 2 3 4 5`

3. INSERTION SORT

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

```
void insertionSort(vector<int>& arr) {
    int n = arr.size();
    for(int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while(j >= 0 && arr[j] > key) {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = key;
    }
}

int main() {
    vector<int> arr = {5,4,3,2,1};
    insertionSort(arr);
    for(int num : arr) cout << num << " ";
    return 0;
}
```

Output: `1 2 3 4 5`

CUTOM COMPARATOR

1. SORT A VECTOR
    - Example descending

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

bool desc(int a, int b) {
    return a > b;
}

int main() {
    vector<int> arr = {5,4,3,2,1};
    sort(arr.begin(), arr.end(), desc);
    for(int num : arr) cout << num << " ";
    return 0;
}
```

Output: `5 4 3 2 1`

2. SORT VECTOR OF VECTOR
    - Example sort by second element

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

bool sortSecond(vector<int> a, vector<int> b) {
    return a[1] < b[1];
}

int main() {
    vector<vector<int>> arr = {{1,5},{2,3},{3,4}};
    sort(arr.begin(), arr.end(), sortSecond);
    for(auto row : arr) cout << row[0] << " " << row[1] << endl;
    return 0;
}
```

Output:

```
2 3
3 4
1 5
```

WEEK 05 TOPIC: Char Arrays and Strings - Level 1

CLASS NOTES: 14

1. Program 01: Length of string

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

int stringLength(char str[]) {
    int len = 0;
    while(str[len] != '\0') len++;
    return len;
}

int main() {
    char str[100] = "hello";
    cout << stringLength(str);
    return 0;
}
```

Output: `5`

2. Program 02: Reverse string

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

```
void reverseString(char str[]) {
    int len = strlen(str);
    int i = 0, j = len - 1;
    while(i < j) swap(str[i++], str[j--]);
}

int main() {
    char str[100] = "hello";
    reverseString(str);
    cout << str;
    return 0;
}
```

Output: `olleh`

3. Program 03: Uppercase to lowercase and vice versa

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

void toggleCase(char str[]) {
    for(int i = 0; str[i] != '\0'; i++) {
        if(isupper(str[i])) str[i] = tolower(str[i]);
        else if(islower(str[i])) str[i] = toupper(str[i]);
    }
}

int main() {
    char str[100] = "Hello";
    toggleCase(str);
    cout << str;
    return 0;
}
```

Output: `hELLO`

4. Program 04: Replace @ with the white space

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

void replaceAtWithSpace(char str[]) {
    for(int i = 0; str[i] != '\0'; i++) {
        if(str[i] == '@') str[i] = ' ';
    }
}
```

```
int main() {
    char str[100] = "hello@world";
    replaceAtWithSpace(str);
    cout << str;
    return 0;
}
```

Output: `hello world`

5. Program 05: Check palindrome

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

bool isPalindrome(char str[]) {
    int len = strlen(str);
    int i = 0, j = len - 1;
    while(i < j) {
        if(str[i++] != str[j--]) return false;
    }
    return true;
}

int main() {
    char str[100] = "radar";
    cout << (isPalindrome(str) ? "Yes" : "No");
    return 0;
}
```

Output: `Yes`

For strings, methods like length, append, etc., are built-in.

CLASS HOMEWORK: 14

1. ASCII CHARACTER CODE 256
    - Explanation: ASCII 0-127 standard, 128-255 extended. No code.
2. Explore build in method from CPLUSHCPLUSH.COM
    - Explanation: Site for C++ docs, e.g., string::size(), string::find(), etc. No code.

WEEK 05 TOPIC: Char Arrays and Strings - Level 2

CLASS NOTES: 15

1. Remove All Adjacent Duplicates In String (Leetcode-1047)

- Approach: Stack

```
class Solution {
public:
    string removeDuplicates(string s) {
        string result = "";
        for(char c : s) {
            if(!result.empty() && result.back() == c) result.pop_back();
            else result.push_back(c);
        }
        return result;
    }
};
```

Example: Input "abbaca", Output "ca"

2. Remove All Occurrences of a Substring (Leetcode-1910)
    - Approach: Find and erase loop

```
class Solution {
public:
    string removeOccurrences(string s, string part) {
        while(s.find(part) != string::npos) {
            s.erase(s.find(part), part.length());
        }
        return s;
    }
};
```

Example: Input "daabcbaabcbc", part="abc", Output "dab"

3. Valid Palindrome II (Leetcode-680)
    - Approach: Two pointers, skip one if mismatch

```
class Solution {
public:
    bool isPalin(const string& s, int i, int j) {
        while(i < j) {
            if(s[i++] != s[j--]) return false;
        }
        return true;
    }
    bool validPalindrome(string s) {
        int i = 0, j = s.length() - 1;
        while(i < j) {
            if(s[i] != s[j]) return isPalin(s, i+1, j) || isPalin(s, i, j-1);
```

```
            i++;
            j--;
        }
        return true;
    }
};
```

Example: Input "aba", Output true

4. Palindromic Substrings (Leetcode-647)
    - Approach: Expand around center

```
class Solution {
public:
    int countSubstrings(string s) {
        int count = 0;
        for(int i = 0; i < s.length(); i++) {
            count += expand(s, i, i); // odd
            count += expand(s, i, i+1); // even
        }
        return count;
    }
    int expand(const string& s, int left, int right) {
        int cnt = 0;
        while(left >= 0 && right < s.length() && s[left--] == s[right++]) cnt++;
        return cnt;
    }
};
```

Example: Input "aaa", Output 6

CLASS HOMEWORK: 15

1. Remove All Adjacent Duplicates in String II (Leetcode-1209)
    - Approach: Stack with count

```
class Solution {
public:
    string removeDuplicates(string s, int k) {
        vector<pair<char, int>> st;
        for(char c : s) {
            if(st.empty() || st.back().first != c) st.push_back({c, 1});
            else if(++st.back().second == k) st.pop_back();
        }
        string result = "";
        for(auto p : st) result += string(p.second, p.first);
```

```
        return result;
    }
};
```

Example: Input "deeedbbcccbdaa", k=3, Output "aa"

2. Minimum Time Difference (Leetcode-539)
    - Approach: Sort times in minutes, find min diff

```
class Solution {
public:
    int findMinDifference(vector<string>& timePoints) {
        vector<int> minutes;
        for(string t : timePoints) {
            int h = stoi(t.substr(0,2)), m = stoi(t.substr(3));
            minutes.push_back(h*60 + m);
        }
        sort(minutes.begin(), minutes.end());
        int minDiff = INT_MAX;
        for(int i = 1; i < minutes.size(); i++) minDiff = min(minDiff,
minutes[i] - minutes[i-1]);
        minDiff = min(minDiff, minutes[0] + 1440 - minutes.back());
        return minDiff;
    }
};
```

Example: Input ["23:59","00:00"], Output 1

WEEK 05 TOPIC: Char Arrays and Strings - Level 3

CLASS NOTES: 16

1. Decode the Message (Leetcode-2325)
    - Approach: Map key to alphabet

```
class Solution {
public:
    string decodeMessage(string key, string message) {
        unordered_map<char, char> mp;
        char ch = 'a';
        for(char c : key) {
            if(c != ' ' && mp.find(c) == mp.end()) mp[c] = ch++;
        }
        string result = "";
        for(char c : message) result += (c == ' ' ? ' ' : mp[c]);
        return result;
```

```
    }
};
```

Example: Input key="the quick brown fox jumps over the lazy dog", message="vkbs bs t suepuv", Output "this is a secret"

2. Minimum Amount of Time to Collect Garbage (Leetcode-2391)
    - Approach: Simulate travel and count

```
class Solution {
public:
    int garbageCollection(vector<string>& garbage, vector<int>& travel) {
        int time = 0, lastM = 0, lastP = 0, lastG = 0;
        for(int i = 0; i < garbage.size(); i++) {
            time += garbage[i].length();
            if(garbage[i].find('M') != string::npos) lastM = i;
            if(garbage[i].find('P') != string::npos) lastP = i;
            if(garbage[i].find('G') != string::npos) lastG = i;
        }
        for(int i = 1; i < travel.size(); i++) travel[i] += travel[i-1];
        time += lastM > 0 ? travel[lastM-1] : 0;
        time += lastP > 0 ? travel[lastP-1] : 0;
        time += lastG > 0 ? travel[lastG-1] : 0;
        return time;
    }
};
```

Example: Input garbage=["G","P","GP","GG"], travel=[2,4,3], Output 21

3. Custom Sort String (Leetcode-791)
    - Approach: Custom comparator based on order

```
class Solution {
public:
    string customSortString(string order, string s) {
        unordered_map<char, int> mp;
        for(int i = 0; i < order.length(); i++) mp[order[i]] = i;
        sort(s.begin(), s.end(), [&](char a, char b) {
            return mp[a] < mp[b];
        });
        return s;
    }
};
```

Example: Input order="cba", s="abcd", Output "cbad"

4. Find and Replace Pattern (Leetcode-890)

- Approach: Normalize patterns

```
class Solution {
public:
    string normalize(string w) {
        unordered_map<char, char> mp;
        char ch = 'a';
        string res = "";
        for(char c : w) {
            if(mp.find(c) == mp.end()) mp[c] = ch++;
            res += mp[c];
        }
        return res;
    }
    vector<string> findAndReplacePattern(vector<string>& words, string pattern)
{
        string pat = normalize(pattern);
        vector<string> result;
        for(string w : words) {
            if(normalize(w) == pat) result.push_back(w);
        }
        return result;
    }
};
```

Example: Input words=["abc","deq","mee"], pattern="abb", Output ["mee"]

CLASS HOMEWORK: 16

1. Valid Anagram (Leetcode-242)
   - Approach: Sort or count

```
class Solution {
public:
    bool isAnagram(string s, string t) {
        if(s.length() != t.length()) return false;
        sort(s.begin(), s.end());
        sort(t.begin(), t.end());
        return s == t;
    }
};
```

Example: Input "anagram", "nagaram", Output true

2. Reverse Only Letters (Leetcode-917)

- Approach: Two pointers, skip non-letters

```
class Solution {
public:
    string reverseOnlyLetters(string s) {
        int i = 0, j = s.length() - 1;
        while(i < j) {
            if(!isalpha(s[i])) i++;
            else if(!isalpha(s[j])) j--;
            else swap(s[i++], s[j--]);
        }
        return s;
    }
};
```

Example: Input "ab-cd", Output "dc-ba"

3. Longest Common Prefix (Leetcode-14)
    - Approach: Sort and compare first and last

```
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
        sort(strs.begin(), strs.end());
        string first = strs[0], last = strs.back();
        string res = "";
        for(int i = 0; i < min(first.length(), last.length()); i++) {
            if(first[i] != last[i]) break;
            res += first[i];
        }
        return res;
    }
};
```

Example: Input ["flower","flow","flight"], Output "fl"

4. Reverse Vowels of a String (Leetcode-345)
    - Approach: Two pointers for vowels

```
class Solution {
public:
    string reverseVowels(string s) {
        unordered_set<char> vowels = {'a','e','i','o','u','A','E','I','O','U'};
        int i = 0, j = s.length() - 1;
        while(i < j) {
            if(vowels.find(s[i]) == vowels.end()) i++;
```

```
            else if(vowels.find(s[j]) == vowels.end()) j--;
            else swap(s[i++], s[j--]);
        }
        return s;
    }
};
```

Example: Input "hello", Output "holle"

5. Isomorphic Strings (Leetcode-205)
    - Approach: Two maps for mapping

```
class Solution {
public:
    bool isIsomorphic(string s, string t) {
        unordered_map<char, char> mp1, mp2;
        for(int i = 0; i < s.length(); i++) {
            if(mp1.count(s[i]) && mp1[s[i]] != t[i]) return false;
            if(mp2.count(t[i]) && mp2[t[i]] != s[i]) return false;
            mp1[s[i]] = t[i];
            mp2[t[i]] = s[i];
        }
        return true;
    }
};
```

Example: Input "egg", "add", Output true

6. Group Anagrams (Leetcode-49)
    - Approach: Sort string as key

```
class Solution {
public:
    vector<vector<string>> groupAnagrams(vector<string>& strs) {
        unordered_map<string, vector<string>> mp;
        for(string str : strs) {
            string key = str;
            sort(key.begin(), key.end());
            mp[key].push_back(str);
        }
        vector<vector<string>> result;
        for(auto p : mp) result.push_back(p.second);
        return result;
    }
};
```

Example: Input ["eat","tea","tan","ate","nat","bat"], Output [["bat"],["nat","tan"],["ate","eat","tea"]]

7. Reorganise String
    - Approach: Priority queue for max frequency

```
class Solution {
public:
    string reorganizeString(string s) {
        vector<int> freq(26, 0);
        for(char c : s) freq[c - 'a']++;
        priority_queue<pair<int, char>> pq;
        for(int i = 0; i < 26; i++) if(freq[i] > 0) pq.push({freq[i], 'a' + i});
        string result = "";
        while(!pq.empty()) {
            auto top = pq.top(); pq.pop();
            result += top.second;
            top.first--;
            pair<int, char> sec = {0, '#'};
            if(!pq.empty()) {
                sec = pq.top(); pq.pop();
                result += sec.second;
                sec.first--;
            }
            if(top.first > 0) pq.push(top);
            if(sec.first > 0) pq.push(sec);
            if(pq.size() == 1 && pq.top().first > 1) return "";
        }
        return result;
    }
};
```

Example: Input "aab", Output "aba"

8. Longest Palindromic Substring
    - Approach: Expand around center

```
class Solution {
public:
    string expand(string s, int left, int right) {
        while(left >= 0 && right < s.length() && s[left] == s[right]) {
            left--;
            right++;
        }
        return s.substr(left + 1, right - left - 1);
    }
```

```
    string longestPalindrome(string s) {
        string maxPal = "";
        for(int i = 0; i < s.length(); i++) {
            string odd = expand(s, i, i);
            string even = expand(s, i, i+1);
            if(odd.length() > maxPal.length()) maxPal = odd;
            if(even.length() > maxPal.length()) maxPal = even;
        }
        return maxPal;
    }
};
```

Example: Input "babad", Output "bab"

9. Find the Index of the First Occurrence in a String
    - Approach: KMP or built-in find

```
class Solution {
public:
    int strStr(string haystack, string needle) {
        return haystack.find(needle);
    }
};
```

Example: Input "sadbutsad", "sad", Output 0

10. String to Integer (atoi)
    - Approach: Handle signs, overflow

```
class Solution {
public:
    int myAtoi(string s) {
        int i = 0, n = s.length(), sign = 1;
        long result = 0;
        while(i < n && s[i] == ' ') i++;
        if(i < n && (s[i] == '+' || s[i] == '-')) sign = (s[i++] == '-') ? -1 :
1;
        while(i < n && isdigit(s[i])) {
            result = result * 10 + (s[i++] - '0');
            if(result * sign > INT_MAX) return INT_MAX;
            if(result * sign < INT_MIN) return INT_MIN;
        }
        return result * sign;
    }
};
```

Example: Input "42", Output 42

11. String Compression
    - Approach: Count consecutive

```
class Solution {
public:
    int compress(vector<char>& chars) {
        int i = 0, n = chars.size();
        for(int j = 0; j < n;) {
            char c = chars[j];
            int count = 0;
            while(j < n && chars[j] == c) {
                j++;
                count++;
            }
            chars[i++] = c;
            if(count > 1) {
                for(char d : to_string(count)) chars[i++] = d;
            }
        }
        return i;
    }
};
```

Example: Input ['a','a','b','b','c','c','c'], Output 6 (array ["a","2","b","2","c","3"])

12. Integer to Roman
    - Approach: Greedy with values

```
class Solution {
public:
    string intToRoman(int num) {
        vector<pair<int, string>> vals = {{1000, "M"}, {900, "CM"}, {500, "D"},
{400, "CD"}, {100, "C"}, {90, "XC"}, {50, "L"}, {40, "XL"}, {10, "X"}, {9,
"IX"}, {5, "V"}, {4, "IV"}, {1, "I"}};
        string result = "";
        for(auto p : vals) {
            while(num >= p.first) {
                result += p.second;
                num -= p.first;
            }
        }
        return result;
```

```
    }
};
```

Example: Input 1994, Output "MCMXCIV"

13. Zig-zag Conversion
    - Approach: Simulate rows

```
class Solution {
public:
    string convert(string s, int numRows) {
        if(numRows == 1) return s;
        vector<string> rows(numRows, "");
        int dir = 1, row = 0;
        for(char c : s) {
            rows[row] += c;
            row += dir;
            if(row == numRows - 1) dir = -1;
            else if(row == 0) dir = 1;
        }
        string result = "";
        for(string r : rows) result += r;
        return result;
    }
};
```

Example: Input "PAYPALISHIRING", 3, Output "PAHNAPLSIIGYIR"

14. Largest Number
    - Approach: Custom sort

```
class Solution {
public:
    static bool comp(string a, string b) {
        return a + b > b + a;
    }
    string largestNumber(vector<int>& nums) {
        vector<string> strNums;
        for(int num : nums) strNums.push_back(to_string(num));
        sort(strNums.begin(), strNums.end(), comp);
        if(strNums[0] == "0") return "0";
        string result = "";
        for(string s : strNums) result += s;
        return result;
    }
};
```

Example: Input [3,30,34,5,9], Output "9534330"

15. Remove All Adjacent Duplicates in String II (Leetcode-1209)
    - Already in homework 1, see above.
16. Implement std::string::erase()
    - Example:

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

int main() {
    string s = "hello world";
    s.erase(5, 1); // remove space
    cout << s;
    return 0;
}
```

    Output: `helloworld`

17. Minimum Time Difference (Leetcode-539)
    - Already in homework 2, see above.
18. Number of Laser Beams in a Bank (Leetcode-2125)
    - Approach: Count devices per row, multiply consecutive

```cpp
class Solution {
public:
    int numberOfBeams(vector<string>& bank) {
        int total = 0, prev = 0;
        for(string row : bank) {
            int count = 0;
            for(char c : row) count += (c == '1');
            if(count > 0) {
                total += prev * count;
                prev = count;
            }
        }
        return total;
    }
};
```

Example: Input ["011001","000000","010100"], Output 8

WEEK 06 TOPIC: Basic Maths & Pointers - Level 1

CLASS NOTES: 17

Pointers concepts, no specific programs.

CLASS HOMEWORK: 17

1. Why pointer size was coming 8 while printing
    - Explanation: On 64-bit system, pointer size is 8 bytes.
2. why we can not do [arr = arr + 1;] in C++
    - Explanation: Array name is constant pointer, can't reassign.
3. Wild pointer in C++
    - Explanation: Uninitialized pointer, points to garbage.
4. Void pointer in C++
    - Explanation: Generic pointer, can point to any type, need cast to use.
5. Dangling pointer in C++
    - Explanation: Pointer to deallocated memory.
6. Pointers imporatant doubt
    - Explanation: Common doubts: * for dereference, & for address.

BASIC MATHEMATICS FOR DSA

1. PROGRAM 01: Count primes (Leetcode-204)
    - Approach 01: Naive

```
class Solution {
public:
    int countPrimes(int n) {
        int count = 0;
        for(int i = 2; i < n; i++) {
            bool prime = true;
            for(int j = 2; j < i; j++) if(i % j == 0) prime = false;
            if(prime) count++;
        }
        return count;
    }
};
```

    - Approach 02: SQRT

```
class Solution {
public:
```

```
    int countPrimes(int n) {
        int count = 0;
        for(int i = 2; i < n; i++) {
            bool prime = true;
            for(int j = 2; j * j <= i; j++) if(i % j == 0) prime = false;
            if(prime) count++;
        }
        return count;
    }
};
```

- Approach 03: Sieve of Eratosthenes

```
class Solution {
public:
    int countPrimes(int n) {
        if(n < 2) return 0;
        vector<bool> isPrime(n, true);
        isPrime[0] = isPrime[1] = false;
        for(int i = 2; i * i < n; i++) {
            if(isPrime[i]) {
                for(int j = i*i; j < n; j += i) isPrime[j] = false;
            }
        }
        return count_if(isPrime.begin(), isPrime.end(), [](bool b){return
b;});
    }
};
```

- Approach 04: Segmented sieve
  - For large n, divide into segments.

Example: n=10, Output 4 (2,3,5,7)

2. PROGRAM 02: Find GCD/HCF using Euclids Algorithm (GFG)

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

int gcd(int a, int b) {
    while(b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
```

```
}

int main() {
    cout << gcd(48, 18);
    return 0;
}
```

Output: `6`

3. PROGRAM 03: Find LCM (GFG)

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

long long lcm(int a, int b) {
    return (a / gcd(a, b)) * (long long)b;
}

int main() {
    cout << lcm(4, 6);
    return 0;
}
```

Output: `12`

CONCEPT 01: Modulo Arithmetic

- Explanation: (a + b) % m = (a % m + b % m) % m, similar for -, *. For division, use modular inverse.

4. PROGRAM 04: Fast exponentiation (GFG)
    - Approach 01: Naive

```
long long powNaive(long long a, long long b) {
    long long res = 1;
    for(long long i = 0; i < b; i++) res *= a;
    return res;
}
```

    - Approach 02: Better solution (binary exp)

```
long long fastExp(long long a, long long b) {
    long long res = 1;
    while(b > 0) {
        if(b & 1) res *= a;
        a *= a;
        b >>= 1;
    }
```

```
    return res;
}
```

5. PROGRAM 05: Modular Exponentiation for large numbers (GFG)
   - Approach: Binary exp with mod

```
long long modExp(long long a, long long b, long long mod) {
    long long res = 1;
    a %= mod;
    while(b > 0) {
        if(b & 1) res = (res * a) % mod;
        a = (a * a) % mod;
        b >>= 1;
    }
    return res;
}
```

6. PROGRAM 06: Optimising Sieve of Eratosthenes
   - Start from i=2, mark multiples from i*i, step i.
7. PROGRAM 07: Segmented Sieve (GFG)
   - For large range, sieve segments.
8. PROGRAM 08: HW 🔗 Product of primes (GFG)
   - Approach: Sieve to find primes, product with mod

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

long long productPrimes(int l, int r, long long mod) {
    vector<bool> isPrime(r+1, true);
    isPrime[0] = isPrime[1] = false;
    for(int i = 2; i * i <= r; i++) {
        if(isPrime[i]) {
            for(int j = i*i; j <= r; j += i) isPrime[j] = false;
        }
    }
    long long prod = 1;
    for(int i = l; i <= r; i++) if(isPrime[i]) prod = (prod * i) % mod;
    return prod;
}

int main() {
    cout << productPrimes(1, 10, 1000000007);
```

```
    return 0;
}
```

Output: `210`

WEEK 06 TOPIC: Basic Maths & Pointers - Level 2

CLASS NOTES: 18

Pointers examples, no new programs.

WEEK 07 TOPIC: Recursion - Level 1

CLASS NOTES: 19

4. Factorial of n number

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

int factorialRec(int n) {
    if(n == 0) return 1;
    return n * factorialRec(n-1);
}

int main() {
    cout << factorialRec(5);
    return 0;
}
```

Output: `120`

5. Reverse counting from n to 1

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

void reverseCount(int n) {
    if(n == 0) return;
    cout << n << " ";
    reverseCount(n-1);
}

int main() {
    reverseCount(5);
    return 0;
}
```

Output: `5 4 3 2 1`

6. Pow(2,N)

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

int pow2N(int n) {
    if(n == 0) return 1;
    return 2 * pow2N(n-1);
}

int main() {
    cout << pow2N(3);
    return 0;
}
```

Output: `8`

7. Fibonacci series

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

int fib(int n) {
    if(n <= 1) return n;
    return fib(n-1) + fib(n-2);
}

int main() {
    cout << fib(5);
    return 0;
}
```

Output: `5`

12. Return sum from n to 1

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

int sumNTo1(int n) {
    if(n == 0) return 0;
    return n + sumNTo1(n-1);
}

int main() {
```

```
    cout << sumNTo1(5);
    return 0;
}
```

Output: `15`

CLASS HOMEWORK: 19

1. Time and space complexity of recursion
    - Explanation: Time: O(2^n) for fib, Space: O(n) stack.

WEEK 07 TOPIC: Recursion - Level 2

CLASS NOTES: 20

1. Climbing stairs (Leetcode-70)
    - Approach: Recursion (inefficient, but as per class)

```
class Solution {
public:
    int climbStairs(int n) {
        if(n <= 2) return n;
        return climbStairs(n-1) + climbStairs(n-2);
    }
};
```

Example: n=3, Output 3

2. Print array

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

void printArray(vector<int> arr, int i) {
    if(i == arr.size()) return;
    cout << arr[i] << " ";
    printArray(arr, i+1);
}

int main() {
    vector<int> arr = {1,2,3};
    printArray(arr, 0);
    return 0;
}
```

Output: `1 2 3`

3. Search in array

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

bool searchArray(vector<int> arr, int target, int i) {
    if(i == arr.size()) return false;
    if(arr[i] == target) return true;
    return searchArray(arr, target, i+1);
}

int main() {
    vector<int> arr = {1,2,3};
    cout << searchArray(arr, 2, 0);
    return 0;
}
```

Output: `1`

4. Minimum in array

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

int minArray(vector<int> arr, int i, int mn) {
    if(i == arr.size()) return mn;
    return minArray(arr, i+1, min(mn, arr[i]));
}

int main() {
    vector<int> arr = {3,1,2};
    cout << minArray(arr, 0, INT_MAX);
    return 0;
}
```

Output: `1`

5. Arrays even elements stored in vector

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

void evenElements(vector<int> arr, int i, vector<int>& evens) {
    if(i == arr.size()) return;
    if(arr[i] % 2 == 0) evens.push_back(arr[i]);
    evenElements(arr, i+1, evens);
}
```

```
int main() {
    vector<int> arr = {1,2,3,4}, evens;
    evenElements(arr, 0, evens);
    for(int e : evens) cout << e << " ";
    return 0;
}
```

Output: `2 4`

6. Double each element

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

void doubleElements(vector<int>& arr, int i) {
    if(i == arr.size()) return;
    arr[i] *= 2;
    doubleElements(arr, i+1);
}

int main() {
    vector<int> arr = {1,2,3};
    doubleElements(arr, 0);
    for(int num : arr) cout << num << " ";
    return 0;
}
```

Output: `2 4 6`

7. Find in Array
    - Similar to search, see 3.
8. Print index of all occurrence of target

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

void printOccurrences(vector<int> arr, int target, int i) {
    if(i == arr.size()) return;
    if(arr[i] == target) cout << i << " ";
    printOccurrences(arr, target, i+1);
}

int main() {
    vector<int> arr = {1,2,2,3};
    printOccurrences(arr, 2, 0);
```

```
    return 0;
}
```

Output: `1 2`

9. Return vector with all occurrences of target

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

vector<int> getOccurrences(vector<int> arr, int target, int i, vector<int> res)
{
    if(i == arr.size()) return res;
    if(arr[i] == target) res.push_back(i);
    return getOccurrences(arr, target, i+1, res);
}

int main() {
    vector<int> arr = {1,2,2,3}, res;
    res = getOccurrences(arr, 2, 0, res);
    for(int idx : res) cout << idx << " ";
    return 0;
}
```

Output: `1 2`

10. Print the digits of the number

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

void printDigits(int n) {
    if(n == 0) return;
    printDigits(n / 10);
    cout << n % 10 << " ";
}

int main() {
    printDigits(1234);
    return 0;
}
```

Output: `1 2 3 4`

CLASS HOMEWORK: 20

1. Print the Fibonacci series using an iterative method

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

void printFibIter(int n) {
    int a = 0, b = 1;
    cout << a << " " << b << " ";
    for(int i = 2; i < n; i++) {
        int next = a + b;
        cout << next << " ";
        a = b;
        b = next;
    }
}

int main() {
    printFibIter(5);
    return 0;
}
```

Output: `0 1 1 2 3`

2. Maximum in an array

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

int maxArray(vector<int> arr, int i, int mx) {
    if(i == arr.size()) return mx;
    return maxArray(arr, i+1, max(mx, arr[i]));
}

int main() {
    vector<int> arr = {3,1,4};
    cout << maxArray(arr, 0, INT_MIN);
    return 0;
}
```

Output: `4`

3. Print number of digits

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

int countDigits(int n) {
    if(n == 0) return 0;
```

```
    return 1 + countDigits(n / 10);
}

int main() {
    cout << countDigits(1234);
    return 0;
}
```

Output: `4`

4. Find target in string and print it's target indices

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

void printTargetIndices(string s, char target, int i) {
    if(i == s.length()) return;
    if(s[i] == target) cout << i << " ";
    printTargetIndices(s, target, i+1);
}

int main() {
    string s = "hello";
    printTargetIndices(s, 'l', 0);
    return 0;
}
```

Output: `2 3`

WEEK 07 TOPIC: Recursion - Level 3

CLASS NOTES: 21

1. Check array sorted or not

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

bool isSorted(vector<int> arr, int i) {
    if(i == arr.size() - 1) return true;
    if(arr[i] > arr[i+1]) return false;
    return isSorted(arr, i+1);
}

int main() {
    vector<int> arr = {1,2,3,4};
    cout << isSorted(arr, 0);
```

```
    return 0;
}
```

Output: `1`

2. Binary search recursive solution

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

int binarySearchRec(vector<int> arr, int target, int low, int high) {
    if(low > high) return -1;
    int mid = low + (high - low) / 2;
    if(arr[mid] == target) return mid;
    else if(arr[mid] < target) return binarySearchRec(arr, target, mid+1, high);
    else return binarySearchRec(arr, target, low, mid-1);
}

int main() {
    vector<int> arr = {1,2,3,4,5};
    cout << binarySearchRec(arr, 3, 0, 4);
    return 0;
}
```

Output: `2`

3. Pattern 01: Include and exclude pattern Problem: Subsequence of string

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

void printSubsequences(string s, string current, int i) {
    if(i == s.length()) {
        cout << current << " ";
        return;
    }
    printSubsequences(s, current, i+1); // exclude
    printSubsequences(s, current + s[i], i+1); // include
}

int main() {
    printSubsequences("abc", "", 0);
    return 0;
}
```

Output: `c b bc a ac ab abc`

4. Pattern 02: Exploring all possible ways pattern Problem: Maximize the cost segment (GFG)

- Assumption: Max cut segments

```
class Solution {
public:
    int maximizeCuts(int n, int x, int y, int z) {
        if(n == 0) return 0;
        if(n < 0) return INT_MIN;
        int op1 = maximizeCuts(n - x, x, y, z);
        int op2 = maximizeCuts(n - y, x, y, z);
        int op3 = maximizeCuts(n - z, x, y, z);
        return 1 + max({op1, op2, op3});
    }
};
```

5. Coin change (Leetcode-322)

- Approach: Recursion

```
class Solution {
public:
    int coinChange(vector<int>& coins, int amount) {
        if(amount == 0) return 0;
        if(amount < 0) return INT_MAX;
        int minCoins = INT_MAX;
        for(int coin : coins) {
            int res = coinChange(coins, amount - coin);
            if(res != INT_MAX) minCoins = min(minCoins, 1 + res);
        }
        return minCoins == INT_MAX ? -1 : minCoins;
    }
};
```

Example: coins [1,2,5], amount 11, Output 3

6. House Robber (Leetcode-198)

- Approach: Recursion

```
class Solution {
public:
    int rob(vector<int>& nums) {
        int n = nums.size();
        if(n == 0) return 0;
        if(n == 1) return nums[0];
        vector<int> dp(n, -1);
        dp[0] = nums[0];
        dp[1] = max(nums[0], nums[1]);
```

```
        for(int i = 2; i < n; i++) {
            dp[i] = max(dp[i-1], nums[i] + dp[i-2]);
        }
        return dp[n-1];
    }
};
```

Example: Input [1,2,3,1], Output 4

CLASS HOMEWORK: 21

1. Last occurrence of a char

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

int lastOccChar(string s, char target, int i) {
    if(i < 0) return -1;
    if(s[i] == target) return i;
    return lastOccChar(s, target, i-1);
}

int main() {
    string s = "hello";
    cout << lastOccChar(s, 'l', s.length()-1);
    return 0;
}
```

Output: `3`

2. Reverse a String

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

void reverseStr(string& s, int i, int j) {
    if(i >= j) return;
    swap(s[i], s[j]);
    reverseStr(s, i+1, j-1);
}

int main() {
    string s = "hello";
    reverseStr(s, 0, s.length()-1);
    cout << s;
    return 0;
}
```

Output: `olleh`

3. Add Two Strings (Integer stored as String)

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

string addStrings(string a, string b) {
    string result = "";
    int i = a.length() - 1, j = b.length() - 1, carry = 0;
    while(i >= 0 || j >= 0 || carry) {
        int sum = carry;
        if(i >= 0) sum += a[i--] - '0';
        if(j >= 0) sum += b[j--] - '0';
        result = to_string(sum % 10) + result;
        carry = sum / 10;
    }
    return result;
}

int main() {
    cout << addStrings("11", "123");
    return 0;
}
```

Output: `134`

4. Palindrome Check
    - Already in level 1, see check palindrome.
5. Print all Subarray

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

void printSubarrays(vector<int> arr, int start, int end) {
    if(end == arr.size()) return;
    if(start > end) printSubarrays(arr, 0, end+1);
    else {
        for(int i = start; i <= end; i++) cout << arr[i] << " ";
        cout << endl;
        printSubarrays(arr, start+1, end);
    }
}

int main() {
```

```
    vector<int> arr = {1,2,3};
    printSubarrays(arr, 0, 0);
    return 0;
}
```

Output:

```
1
1 2
1 2 3
2
2 3
3
```

6. Remove all Occurrence of a Substring
   - Already in level 2, see Leetcode-1910.
7. Buy and sell stocks
   - Approach: One pass for max profit

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int minPrice = INT_MAX, maxProf = 0;
        for(int p : prices) {
            minPrice = min(minPrice, p);
            maxProf = max(maxProf, p - minPrice);
        }
        return maxProf;
    }
};
```

8. House Robbery problem
   - Already in class, see Leetcode-198.
9. Integer to English words
   - Approach: Recursive breakdown

```
class Solution {
public:
    vector<string> below20 = {"", "One", "Two", "Three", "Four", "Five",
"Six", "Seven", "Eight", "Nine", "Ten", "Eleven", "Twelve", "Thirteen",
"Fourteen", "Fifteen", "Sixteen", "Seventeen", "Eighteen", "Nineteen"};
    vector<string> tens = {"", "", "Twenty", "Thirty", "Forty", "Fifty",
"Sixty", "Seventy", "Eighty", "Ninety"};
    vector<string> thousands = {"", "Thousand", "Million", "Billion"};
```

```
    string helper(int n) {
        if(n == 0) return "";
        if(n < 20) return below20[n] + " ";
        if(n < 100) return tens[n/10] + " " + helper(n%10);
        if(n < 1000) return below20[n/100] + " Hundred " + helper(n%100);
        for(int i = 1; i <= 3; i++) {
            if(n < pow(1000, i+1)) return helper(n / pow(1000, i)) +
thousands[i] + " " + helper(n % (int)pow(1000, i));
        }
        return "";
    }

    string numberToWords(int num) {
        if(num == 0) return "Zero";
        string res = helper(num);
        res.pop_back(); // remove trailing space
        return res;
    }
};
```

Example: Input 123, Output "One Hundred Twenty Three"

10. Wild Card Matching
    - Approach: DP

```
class Solution {
public:
    bool isMatch(string s, string p) {
        int m = s.length(), n = p.length();
        vector<vector<bool>> dp(m+1, vector<bool>(n+1, false));
        dp[0][0] = true;
        for(int j = 1; j <= n; j++) if(p[j-1] == '*') dp[0][j] = dp[0][j-1];
        for(int i = 1; i <= m; i++) {
            for(int j = 1; j <= n; j++) {
                if(p[j-1] == '*' ) dp[i][j] = dp[i][j-1] || dp[i-1][j];
                else if(p[j-1] == '?' || s[i-1] == p[j-1]) dp[i][j] = dp[i-1][j-
1];
            }
        }
        return dp[m][n];
    }
};
```

11. Perfect Square

- Approach: Binary search

```
class Solution {
public:
    int numSquares(int n) {
        vector<int> dp(n+1, INT_MAX);
        dp[0] = 0;
        for(int i = 1; i <= n; i++) {
            for(int j = 1; j*j <= i; j++) {
                dp[i] = min(dp[i], dp[i - j*j] + 1);
            }
        }
        return dp[n];
    }
};
```

12. Minimum Cost for Tickets

- Approach: DP

```
class Solution {
public:
    int mincostTickets(vector<int>& days, vector<int>& costs) {
        unordered_set<int> travel(days.begin(), days.end());
        vector<int> dp(366, 0);
        for(int i = 1; i < 366; i++) {
            if(travel.find(i) == travel.end()) dp[i] = dp[i-1];
            else dp[i] = min({dp[i-1] + costs[0], dp[max(0, i-7)] + costs[1],
dp[max(0, i-30)] + costs[2]});
        }
        return dp[365];
    }
};
```

13. Number of Dice Roll with Target Sum

- Approach: DP

```
class Solution {
public:
    int numRollsToTarget(int n, int k, int target) {
        const int MOD = 1000000007;
        vector<vector<int>> dp(n+1, vector<int>(target+1, 0));
        dp[0][0] = 1;
        for(int i = 1; i <= n; i++) {
            for(int j = 1; j <= target; j++) {
                for(int f = 1; f <= k; f++) {
```

```
                    if(j - f >= 0) dp[i][j] = (dp[i][j] + dp[i-1][j-f]) % MOD;
                }
            }
        }
        return dp[n][target];
    }
};
```

WEEK 08 TOPIC: Backtracking & Divide and conquer - Class 01

CLASS NOTES: 22

2. Merge sort

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

void merge(vector<int>& arr, int low, int mid, int high) {
    vector<int> temp;
    int i = low, j = mid+1;
    while(i <= mid && j <= high) {
        if(arr[i] <= arr[j]) temp.push_back(arr[i++]);
        else temp.push_back(arr[j++]);
    }
    while(i <= mid) temp.push_back(arr[i++]);
    while(j <= high) temp.push_back(arr[j++]);
    for(int k = low; k <= high; k++) arr[k] = temp[k - low];
}

void mergeSort(vector<int>& arr, int low, int high) {
    if(low >= high) return;
    int mid = low + (high - low) / 2;
    mergeSort(arr, low, mid);
    mergeSort(arr, mid+1, high);
    merge(arr, low, mid, high);
}

int main() {
    vector<int> arr = {5,4,3,2,1};
    mergeSort(arr, 0, 4);
    for(int num : arr) cout << num << " ";
    return 0;
}
```

Output: `1 2 3 4 5`

3. Merge two sorted array

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

vector<int> mergeTwoSorted(vector<int> a, vector<int> b) {
    vector<int> merged;
    int i = 0, j = 0;
    while(i < a.size() && j < b.size()) {
        if(a[i] <= b[j]) merged.push_back(a[i++]);
        else merged.push_back(b[j++]);
    }
    while(i < a.size()) merged.push_back(a[i++]);
    while(j < b.size()) merged.push_back(b[j++]);
    return merged;
}

int main() {
    vector<int> a = {1,3,5}, b = {2,4,6};
    auto res = mergeTwoSorted(a, b);
    for(int num : res) cout << num << " ";
    return 0;
}
```

Output: `1 2 3 4 5 6`

4. Quick sort algorithm

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

int partition(vector<int>& arr, int low, int high) {
    int pivot = arr[low];
    int i = low, j = high;
    while(i < j) {
        while(arr[i] <= pivot && i <= high) i++;
        while(arr[j] > pivot && j >= low) j--;
        if(i < j) swap(arr[i], arr[j]);
    }
    swap(arr[low], arr[j]);
    return j;
}

void quickSort(vector<int>& arr, int low, int high) {
    if(low >= high) return;
```

```
    int p = partition(arr, low, high);
    quickSort(arr, low, p-1);
    quickSort(arr, p+1, high);
}

int main() {
    vector<int> arr = {5,4,3,2,1};
    quickSort(arr, 0, 4);
    for(int num : arr) cout << num << " ";
    return 0;
}
```

Output: `1 2 3 4 5`

CLASS HOMEWORK: 22

1. Space complexity of merge sort
   - Explanation: O(n) for temp array.
2. Inversion count in Array using Merge Sort
   - Approach: Count during merge

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

long long mergeCount(vector<int>& arr, int low, int mid, int high) {
    vector<int> temp;
    int i = low, j = mid+1;
    long long inv = 0;
    while(i <= mid && j <= high) {
        if(arr[i] <= arr[j]) temp.push_back(arr[i++]);
        else {
            temp.push_back(arr[j++]);
            inv += mid - i + 1;
        }
    }
    while(i <= mid) temp.push_back(arr[i++]);
    while(j <= high) temp.push_back(arr[j++]);
    for(int k = low; k <= high; k++) arr[k] = temp[k - low];
    return inv;
}

long long inversionCount(vector<int>& arr, int low, int high) {
    if(low >= high) return 0;
    int mid = low + (high - low) / 2;
```

```
    long long inv = inversionCount(arr, low, mid);
    inv += inversionCount(arr, mid+1, high);
    inv += mergeCount(arr, low, mid, high);
    return inv;
}

int main() {
    vector<int> arr = {5,4,3,2,1};
    cout << inversionCount(arr, 0, 4);
    return 0;
}
```

Output: `10`

WEEK 08 TOPIC: Backtracking & Divide and conquer - Class 02

CLASS NOTES: 23

2. Permutation of string
    - Approach: Backtracking

```
class Solution {
public:
    void permuteHelper(string& s, int i, vector<string>& res) {
        if(i == s.length()) {
            res.push_back(s);
            return;
        }
        for(int j = i; j < s.length(); j++) {
            swap(s[i], s[j]);
            permuteHelper(s, i+1, res);
            swap(s[i], s[j]); // backtrack
        }
    }
    vector<string> find_permutation(string s) {
        vector<string> res;
        permuteHelper(s, 0, res);
        sort(res.begin(), res.end());
        return res;
    }
};
```

3. Rat in a maze
    - Approach: Backtracking with directions

```
class Solution {
public:
    void solve(vector<vector<int>>& m, int n, int x, int y, string path,
vector<string>& res, vector<vector<int>>& visited) {
        if(x < 0 || y < 0 || x >= n || y >= n || m[x][y] == 0 || visited[x]
[y]) return;
        if(x == n-1 && y == n-1) {
            res.push_back(path);
            return;
        }
        visited[x][y] = 1;
        solve(m, n, x-1, y, path + 'U', res, visited);
        solve(m, n, x+1, y, path + 'D', res, visited);
        solve(m, n, x, y-1, path + 'L', res, visited);
        solve(m, n, x, y+1, path + 'R', res, visited);
        visited[x][y] = 0;
    }
    vector<string> findPath(vector<vector<int>> &m, int n) {
        vector<string> res;
        vector<vector<int>> visited(n, vector<int>(n, 0));
        solve(m, n, 0, 0, "", res, visited);
        sort(res.begin(), res.end());
        return res;
    }
};
```

CLASS HOMEWORK: 23

1. Count inversion
   - Already in previous homework.
2. In-place merge sort
   - Standard merge sort is not in-place, but can optimize with extra space O(n).
3. Maximum Subarray
   - Approach: Kadane's

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int maxSum = INT_MIN, curr = 0;
        for(int num : nums) {
            curr = max(num, curr + num);
            maxSum = max(maxSum, curr);
```

```
        }
        return maxSum;
    }
};
```

4. Combination sum problem
    - Approach: Backtracking

```
class Solution {
public:
    void comboHelper(vector<int>& candidates, int target, int i, vector<int>
current, vector<vector<int>>& res) {
        if(target == 0) {
            res.push_back(current);
            return;
        }
        if(target < 0 || i == candidates.length()) return;
        comboHelper(candidates, target, i+1, current, res); // exclude
        current.push_back(candidates[i]);
        comboHelper(candidates, target - candidates[i], i, current, res); //
include
        current.pop_back();
    }
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        vector<vector<int>> res;
        vector<int> current;
        comboHelper(candidates, target, 0, current, res);
        return res;
    }
};
```

5. Combination sum problem - II
    - Similar, but skip duplicates

```
class Solution {
public:
    void comboHelper2(vector<int>& candidates, int target, int i, vector<int>
current, vector<vector<int>>& res) {
        if(target == 0) {
            res.push_back(current);
            return;
        }
        if(target < 0) return;
        for(int j = i; j < candidates.size(); j++) {
```

```
            if(j > i && candidates[j] == candidates[j-1]) continue;
            current.push_back(candidates[j]);
            comboHelper2(candidates, target - candidates[j], j+1, current,
res);
            current.pop_back();
        }
    }
    vector<vector<int>> combinationSum2(vector<int>& candidates, int target)
{
        vector<vector<int>> res;
        vector<int> current;
        sort(candidates.begin(), candidates.end());
        comboHelper2(candidates, target, 0, current, res);
        return res;
    }
};
```

6. Permutation - II

- Approach: Backtracking with used array

```
class Solution {
public:
    void permuteUniqueHelper(vector<int>& nums, vector<bool> used,
vector<int> current, vector<vector<int>>& res) {
        if(current.size() == nums.size()) {
            res.push_back(current);
            return;
        }
        for(int i = 0; i < nums.size(); i++) {
            if(used[i]) continue;
            if(i > 0 && nums[i] == nums[i-1] && !used[i-1]) continue;
            used[i] = true;
            current.push_back(nums[i]);
            permuteUniqueHelper(nums, used, current, res);
            current.pop_back();
            used[i] = false;
        }
    }
    vector<vector<int>> permuteUnique(vector<int>& nums) {
        vector<vector<int>> res;
        vector<int> current;
        vector<bool> used(nums.size(), false);
        sort(nums.begin(), nums.end());
        permuteUniqueHelper(nums, used, current, res);
```

```
        return res;
    }
};
```

7. Beautiful Arrangement

- Approach: Backtracking

```
class Solution {
public:
    int count = 0;
    void helper(int n, int pos, vector<bool>& used) {
        if(pos > n) {
            count++;
            return;
        }
        for(int i = 1; i <= n; i++) {
            if(!used[i] && (i % pos == 0 || pos % i == 0)) {
                used[i] = true;
                helper(n, pos+1, used);
                used[i] = false;
            }
        }
    }
    int countArrangement(int n) {
        vector<bool> used(n+1, false);
        helper(n, 1, used);
        return count;
    }
};
```

8. Distribute Repeating Integers

- Approach: Greedy frequency

```
class Solution {
public:
    bool canDistribute(vector<int>& nums, vector<int>& quantity) {
        unordered_map<int, int> freq;
        for(int num : nums) freq[num]++;
        vector<int> counts;
        for(auto p : freq) counts.push_back(p.second);
        sort(counts.rbegin(), counts.rend());
        sort(quantity.rbegin(), quantity.rend());
        return dfs(counts, quantity, 0);
    }
```

```
    bool dfs(vector<int>& counts, vector<int>& quantity, int idx) {
        if(idx == quantity.size()) return true;
        for(int &c : counts) {
            if(c >= quantity[idx]) {
                c -= quantity[idx];
                if(dfs(counts, quantity, idx+1)) return true;
                c += quantity[idx];
            }
        }
        return false;
    }
};
```

WEEK 08 TOPIC: Recursion marathon - Extra class

CLASS NOTES: 24

1. Maximum sum of non - adjacent element - House Robber (Leetcode-198)
    - Already.
2. House Robber II (Leetcode-213)
    - Approach: DP twice, with/without first

```
class Solution {
public:
    int robHelper(vector<int> nums) {
        int n = nums.size();
        if(n == 0) return 0;
        vector<int> dp(n, 0);
        dp[0] = nums[0];
        for(int i = 1; i < n; i++) {
            int take = nums[i] + (i > 1 ? dp[i-2] : 0);
            int notTake = dp[i-1];
            dp[i] = max(take, notTake);
        }
        return dp[n-1];
    }
    int rob(vector<int>& nums) {
        int n = nums.size();
        if(n == 1) return nums[0];
        vector<int> withoutFirst(nums.begin() + 1, nums.end());
        vector<int> withoutLast(nums.begin(), nums.end() - 1);
        return max(robHelper(withoutFirst), robHelper(withoutLast));
```

```
    }
};
```

3. Count Derangements (GFG)

- Approach: Recursion with formula

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

long long countDerangements(int n) {
    if(n == 1) return 0;
    if(n == 2) return 1;
    return (n-1) * (countDerangements(n-1) + countDerangements(n-2));
}

int main() {
    cout << countDerangements(3);
    return 0;
}
```

Output: `2`

4. Painting Fence Algorithm (GFG)

- Approach: DP

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

long long countWays(int n, int k) {
    const int MOD = 1000000007;
    if(n == 1) return k;
    long long same = k, diff = k * (k - 1);
    for(int i = 3; i <= n; i++) {
        long long newSame = diff;
        long long newDiff = (same + diff) * (k - 1) % MOD;
        same = newSame % MOD;
        diff = newDiff % MOD;
    }
    return (same + diff) % MOD;
}

int main() {
    cout << countWays(3, 2);
    return 0;
}
```

Output: `6`

5. Edit distance (Leetcode-72)
   - Approach: Recursion

```
class Solution {
public:
    int minDistance(string word1, string word2) {
        int m = word1.length(), n = word2.length();
        if(m == 0) return n;
        if(n == 0) return m;
        if(word1[m-1] == word2[n-1]) return minDistance(word1.substr(0, m-1),
word2.substr(0, n-1));
        int insert = minDistance(word1, word2.substr(0, n-1)) + 1;
        int del = minDistance(word1.substr(0, m-1), word2) + 1;
        int replace = minDistance(word1.substr(0, m-1), word2.substr(0, n-1))
+ 1;
        return min({insert, del, replace});
    }
};
```

6. Maximal Square (Leetcode-221)
   - Approach: DP

```
class Solution {
public:
    int maximalSquare(vector<vector<char>>& matrix) {
        int m = matrix.size(), n = matrix[0].size();
        vector<vector<int>> dp(m+1, vector<int>(n+1, 0));
        int maxSide = 0;
        for(int i = 1; i <= m; i++) {
            for(int j = 1; j <= n; j++) {
                if(matrix[i-1][j-1] == '1') {
                    dp[i][j] = min({dp[i-1][j], dp[i][j-1], dp[i-1][j-1]}) +
1;
                    maxSide = max(maxSide, dp[i][j]);
                }
            }
        }
        return maxSide * maxSide;
    }
};
```

CLASS HOMEWORK: 24

1. 0/1 Knapsack Problem (GFG)
    - Approach: Recursion

```
class Solution {
public:
    int knapSack(int W, vector<int> wt, vector<int> val, int n) {
        if(n == 0 || W == 0) return 0;
        if(wt[n-1] > W) return knapSack(W, wt, val, n-1);
        int take = val[n-1] + knapSack(W - wt[n-1], wt, val, n-1);
        int notTake = knapSack(W, wt, val, n-1);
        return max(take, notTake);
    }
};
```

2. Minimum Score Triangulation of Polygon (Leetcode-1039)
    - Approach: DP

```
class Solution {
public:
    int minScoreTriangulation(vector<int>& values) {
        int n = values.size();
        vector<vector<int>> dp(n, vector<int>(n, 0));
        for(int len = 3; len <= n; len++) {
            for(int i = 0; i + len - 1 < n; i++) {
                int j = i + len - 1;
                dp[i][j] = INT_MAX;
                for(int k = i+1; k < j; k++) {
                    dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j] + values[i]
* values[j] * values[k]);
                }
            }
        }
        return dp[0][n-1];
    }
};
```

3. Number of Dice Rolls With Target Sum (Leetcode-1155)
    - Already in DP.

WEEK 09 TOPIC: Object Oriented Programming - Class 01

CLASS NOTES: 25

Class, object, etc., no specific programs.

CLASS HOMEWORK: 25

1. Const keyword
    - Example: const int x = 10; // constant
2. Default argument
    - Example: int add(int a, int b = 0) { return a + b; }
3. Initialization list
    - Example: Class() : member(0) {}
4. MACROS
    - Example: #define PI 3.14
5. Static keyword in class
    - Example: static int count; static void func() {}

WEEK 09 TOPIC: Object Oriented Programming - Class 02

CLASS NOTES: 26

Copy ctor, destructor, getter setter, inheritance, polymorphism.

CLASS HOMEWORK: 26

1. Shallow vs deep copy
    - Shallow: Copy pointer, deep: Copy data.
2. Can constructor be made private
    - Yes, for singleton.
3. Friend keyword in C++
    - Example: friend class B; for access.

WEEK 09 TOPIC: Object Oriented Programming - Class 03

CLASS NOTES: 27

Run time polymorphism (virtual functions).

CLASS HOMEWORK: 27

1. Virtual CTOR vs virtual DTOR
    - Virtual ctor not possible, virtual dtor for base delete.
2. Abstraction in C++
    - Pure virtual for abstract class.

3. Inline function
    - inline int add(int a, int b) { return a + b; }

WEEK 10 TOPIC: Linked List - Class 01

CLASS NOTES: 28

4. Types of linked list
    - SLL, DLL, CLL
5. Create a linked list and a node

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

class Node {
public:
    int data;
    Node* next;
    Node(int d) : data(d), next(nullptr) {}
};

int main() {
    Node* head = new Node(1);
    return 0;
}
```

6. Print linked list

```
void printLL(Node* head) {
    while(head) {
        cout << head->data << " ";
        head = head->next;
    }
}
```

7. Print the length of the linked list "Number of nodes"

```
int lengthLL(Node* head) {
    int len = 0;
    while(head) {
        len++;
        head = head->next;
    }
    return len;
}
```

8. Insertion operations of SLL
    - Insert head

```
void insertHead(Node*& head, int d) {
    Node* newNode = new Node(d);
    newNode->next = head;
    head = newNode;
}
```

    - Insert tail

```
void insertTail(Node* &head, int d) {
    if(!head) {
        head = new Node(d);
        return;
    }
    Node* temp = head;
    while(temp->next) temp = temp->next;
    temp->next = new Node(d);
}
```

    - Insert at any position

```
void insertPosition(Node* &head, int d, int pos) {
    if(pos == 1) {
        insertHead(head, d);
        return;
    }
    Node* temp = head;
    for(int i = 1; i < pos-1; i++) temp = temp->next;
    Node* newNode = new Node(d);
    newNode->next = temp->next;
    temp->next = newNode;
}
```

9. Create a tail
    - Node* tail = head;
      while(tail->next) tail = tail->next;

CLASS HOMEWORK: 28

1. Insert at any position with the help of only one pointer "PREVIOUS"
    - Similar to above, use prev pointer.

WEEK 10 TOPIC: Linked List - Class 02

CLASS NOTES: 29

1. Deletion operations of SLL
    - Delete head

```
void deleteHead(Node*& head) {
    if(!head) return;
    Node* temp = head;
    head = head->next;
    delete temp;
}
```

    - Delete tail

```
void deleteTail(Node* &head) {
    if(!head) return;
    if(!head->next) {
        delete head;
        head = nullptr;
        return;
    }
    Node* temp = head;
    while(temp->next->next) temp = temp->next;
    delete temp->next;
    temp->next = nullptr;
}
```

    - Delete any position

```
void deletePosition(Node* &head, int pos) {
    if(pos == 1) {
        deleteHead(head);
        return;
    }
    Node* temp = head;
    for(int i = 1; i < pos-1; i++) temp = temp->next;
    Node* toDel = temp->next;
    temp->next = toDel->next;
    delete toDel;
}
```

2. Double linked list
    - Class with prev next.
3. Deletion operations of DLL

- Similar, update prev.

CLASS HOMEWORK: 29

1. Circular linked list
   - Class with next pointing to head for tail.

WEEK 10 TOPIC: Linked List - Class 03

CLASS NOTES: 30

1. Reverse Linked List (Leetcode-206)
   - Approach 1: Iterative

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* prev = nullptr, *curr = head, *next;
        while(curr) {
            next = curr->next;
            curr->next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }
};
```

   - Approach 2: Recursive

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        if(!head || !head->next) return head;
        ListNode* newHead = reverseList(head->next);
        head->next->next = head;
        head->next = nullptr;
        return newHead;
    }
};
```

2. Middle of the Linked List (Leetcode-876)
   - Approach 1: getLength and getMid

```
class Solution {
public:
```

```
    ListNode* middleNode(ListNode* head) {
        int len = 0;
        ListNode* temp = head;
        while(temp) {
            len++;
            temp = temp->next;
        }
        temp = head;
        for(int i = 0; i < len/2; i++) temp = temp->next;
        return temp;
    }
};
```

- Approach 2: Slow and fast pointer "Hare & Tortoise" algorithm

```
class Solution {
public:
    ListNode* middleNode(ListNode* head) {
        ListNode* slow = head, *fast = head;
        while(fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }
        return slow;
    }
};
```

3. Palindrome Linked List (Leetcode-234)

- Approach 1: Optimal (reverse second half)

```
class Solution {
public:
    bool isPalindrome(ListNode* head) {
        if(!head || !head->next) return true;
        ListNode* slow = head, *fast = head;
        while(fast->next && fast->next->next) {
            slow = slow->next;
            fast = fast->next->next;
        }
        ListNode* second = reverse(slow->next);
        slow->next = nullptr;
        ListNode* first = head;
        while(second) {
            if(first->val != second->val) return false;
            first = first->next;
```

```
            second = second->next;
        }
        return true;
    }
    ListNode* reverse(ListNode* head) {
        ListNode* prev = nullptr, *curr = head, *next;
        while(curr) {
            next = curr->next;
            curr->next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }
};
```

- Approach 2: Brute force (stack)
  Stack with all, then compare.

4. Linked List Cycle (Leetcode-141)
   - Approach 1: STL map

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
        unordered_map<ListNode*, bool> mp;
        while(head) {
            if(mp[head]) return true;
            mp[head] = true;
            head = head->next;
        }
        return false;
    }
};
```

WEEK 10 TOPIC: Linked List - Class 04

CLASS NOTES: 31

1. Linked List Cycle (Leetcode-141)
   - Approach 1: Fast and slow

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
        ListNode* slow = head, *fast = head;
```

```
        while(fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
            if(slow == fast) return true;
        }
        return false;
    }
};
```

2. Starting point of loop (Leetcode-142)
    - Approach: Fast and slow, reset slow to head

```
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        ListNode* slow = head, *fast = head;
        while(fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
            if(slow == fast) {
                slow = head;
                while(slow != fast) {
                    slow = slow->next;
                    fast = fast->next;
                }
                return slow;
            }
        }
        return nullptr;
    }
};
```

3. Remove loop (GFG)
    - Approach: Find loop start, break last next

```
void removeLoop(Node* head) {
    Node* slow = head, *fast = head;
    while(fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;
        if(slow == fast) {
            slow = head;
            while(slow->next != fast->next) {
                slow = slow->next;
```

```
                fast = fast->next;
            }
            fast->next = nullptr;
        }
    }
}
```

4. Add 1 to a linked list (GFG)
    - Approach: Reverse, add, reverse

```
class Solution {
public:
    Node* addOne(Node *head) {
        head = reverse(head);
        Node* temp = head;
        int carry = 1;
        while(temp) {
            int sum = temp->data + carry;
            temp->data = sum % 10;
            carry = sum / 10;
            if(!temp->next && carry) {
                temp->next = new Node(carry);
                carry = 0;
            }
            temp = temp->next;
        }
        return reverse(head);
    }
    Node* reverse(Node* head) {
        Node* prev = nullptr, *curr = head, *next;
        while(curr) {
            next = curr->next;
            curr->next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }
};
```

5. Reverse Nodes in k-Group (Leetcode-25)
    - Approach: Recursive

```cpp
class Solution {
public:
    ListNode* reverseKGroup(ListNode* head, int k) {
        ListNode* temp = head;
        int count = 0;
        while(temp && count < k) {
            temp = temp->next;
            count++;
        }
        if(count < k) return head;
        ListNode* prev = nullptr, *curr = head, *next;
        count = 0;
        while(curr && count < k) {
            next = curr->next;
            curr->next = prev;
            prev = curr;
            curr = next;
            count++;
        }
        if(next) head->next = reverseKGroup(next, k);
        return prev;
    }
};
```

6. Remove Duplicates from Sorted List (Leetcode-83)
    - Approach: Traverse and skip dups

```cpp
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* curr = head;
        while(curr && curr->next) {
            if(curr->val == curr->next->val) curr->next = curr->next->next;
            else curr = curr->next;
        }
        return head;
    }
};
```

CLASS HOMEWORK: 31

1. Merge Two Sorted Lists (Leetcode-21)
    - Approach: Dummy node

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        ListNode dummy(0);
        ListNode* tail = &dummy;
        while(list1 && list2) {
            if(list1->val <= list2->val) {
                tail->next = list1;
                list1 = list1->next;
            } else {
                tail->next = list2;
                list2 = list2->next;
            }
            tail = tail->next;
        }
        tail->next = list1 ? list1 : list2;
        return dummy.next;
    }
};
```

2. Sort Lists using Merge Sort (Leetcode-148)
    - Approach: Divide and merge

```
class Solution {
public:
    ListNode* sortList(ListNode* head) {
        if(!head || !head->next) return head;
        ListNode* slow = head, *fast = head->next;
        while(fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }
        ListNode* mid = slow->next;
        slow->next = nullptr;
        return merge(sortList(head), sortList(mid));
    }
    ListNode* merge(ListNode* a, ListNode* b) {
        ListNode dummy(0);
        ListNode* tail = &dummy;
        while(a && b) {
            if(a->val <= b->val) {
                tail->next = a;
                a = a->next;
            } else {
```

```
                tail->next = b;
                b = b->next;
            }
            tail = tail->next;
        }
        tail->next = a ? a : b;
        return dummy.next;
    }
};
```

3. Intersection of Two Linked Lists (Leetcode-160)
    - Approach: Two pointers cycle

```
class Solution {
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
        ListNode* a = headA, *b = headB;
        while(a != b) {
            a = a ? a->next : headB;
            b = b ? b->next : headA;
        }
        return a;
    }
};
```

4. Delete N Nodes after M Nodes (GFG)
    - Approach: Traverse and skip

```
void linkdelete(Node  *head, int M, int N) {
    Node* curr = head;
    while(curr) {
        for(int i = 1; i < M && curr; i++) curr = curr->next;
        if(!curr) return;
        Node* temp = curr->next;
        for(int i = 0; i < N && temp; i++) {
            Node* toDel = temp;
            temp = temp->next;
            delete toDel;
        }
        curr->next = temp;
        curr = temp;
    }
}
```

5. Print kth Node from the End (Hackerrank)

- Approach: Two pointers

```
int getNode(SinglyLinkedListNode* head, int positionFromTail) {
    SinglyLinkedListNode* fast = head, *slow = head;
    for(int i = 0; i < positionFromTail; i++) fast = fast->next;
    while(fast->next) {
        slow = slow->next;
        fast = fast->next;
    }
    return slow->data;
}
```

6. Flatten Linked List (GFG)
   - Approach: Merge k lists
     - Use priority queue or recursive merge.
7. Copy List with Random Pointer (Leetcode-138)
   - Approach: Map or interleave

```
class Solution {
public:
    Node* copyRandomList(Node* head) {
        if(!head) return nullptr;
        Node* curr = head;
        while(curr) {
            Node* newNode = new Node(curr->val);
            newNode->next = curr->next;
            curr->next = newNode;
            curr = newNode->next;
        }
        curr = head;
        while(curr) {
            if(curr->random) curr->next->random = curr->random->next;
            curr = curr->next->next;
        }
        curr = head;
        Node* newHead = head->next;
        Node* newCurr = newHead;
        while(curr) {
            curr->next = curr->next->next;
            curr = curr->next;
            if(curr) newCurr->next = curr->next;
            newCurr = newCurr->next;
        }
```

```
        return newHead;
    }
};
```

8. Rotate List (Leetcode-61)
    - Approach: Find new tail

```
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if(!head) return head;
        int len = 1;
        ListNode* tail = head;
        while(tail->next) {
            len++;
            tail = tail->next;
        }
        k %= len;
        if(k == 0) return head;
        tail->next = head;
        ListNode* newTail = head;
        for(int i = 1; i < len - k; i++) newTail = newTail->next;
        ListNode* newHead = newTail->next;
        newTail->next = nullptr;
        return newHead;
    }
};
```

9. Odd Even Linked List (Leetcode-328)
    - Approach: Two lists, merge

```
class Solution {
public:
    ListNode* oddEvenList(ListNode* head) {
        if(!head) return head;
        ListNode odd(0), even(0);
        ListNode* oddTail = &odd, *evenTail = &even;
        bool isOdd = true;
        while(head) {
            if(isOdd) {
                oddTail->next = head;
                oddTail = oddTail->next;
            } else {
                evenTail->next = head;
```

```
                evenTail = evenTail->next;
            }
            isOdd = !isOdd;
            head = head->next;
        }
        oddTail->next = even.next;
        evenTail->next = nullptr;
        return odd.next;
    }
};
```

10. Find Minimum and Maximum Number of Nodes Between Critical Points (Leetcode-2048)
    - Approach: Traverse, find critical, calculate min max dist

```
class Solution {
public:
    vector<int> nodesBetweenCriticalPoints(ListNode* head) {
        vector<int> critical;
        ListNode* prev = head;
        head = head->next;
        int idx = 1;
        while(head->next) {
            if((prev->val < head->val && head->val > head->next->val) || (prev->val > head->val && head->val < head->next->val)) critical.push_back(idx);
            prev = head;
            head = head->next;
            idx++;
        }
        if(critical.size() < 2) return {-1,-1};
        int minDist = INT_MAX, maxDist = critical.back() - critical[0];
        for(int i = 1; i < critical.size(); i++) minDist = min(minDist, critical[i] - critical[i-1]);
        return {minDist, maxDist};
    }
};
```

11. Merge Nodes in between Zeros (Leetcode-2181)
    - Approach: Sum between zeros

```
class Solution {
public:
    ListNode* mergeNodes(ListNode* head) {
        ListNode dummy(0);
        ListNode* tail = &dummy;
```

```
        head = head->next;
        while(head) {
            int sum = 0;
            while(head && head->val != 0) {
                sum += head->val;
                head = head->next;
            }
            if(sum > 0) {
                tail->next = new ListNode(sum);
                tail = tail->next;
            }
            if(head) head = head->next;
        }
        return dummy.next;
    }
};
```

12. Add two linked list (Leetcode-445)
    - Approach: Reverse, add, reverse

```
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        l1 = reverse(l1);
        l2 = reverse(l2);
        ListNode dummy(0);
        ListNode* tail = &dummy;
        int carry = 0;
        while(l1 || l2 || carry) {
            int sum = carry;
            if(l1) sum += l1->val, l1 = l1->next;
            if(l2) sum += l2->val, l2 = l2->next;
            tail->next = new ListNode(sum % 10);
            tail = tail->next;
            carry = sum / 10;
        }
        return reverse(dummy.next);
    }
    ListNode* reverse(ListNode* head) {
        ListNode* prev = nullptr, *curr = head, *next;
        while(curr) {
            next = curr->next;
            curr->next = prev;
            prev = curr;
```

```
            curr = next;
        }
        return prev;
    }
};
```

13. Sort 0, 1, 2 in linked list (GFG)
    - Approach: Dutch flag

```
Node* segregate(Node *head) {
    Node zero(0), one(0), two(0);
    Node* z = &zero, *o = &one, *t = &two;
    Node* curr = head;
    while(curr) {
        if(curr->data == 0) z = z->next = curr;
        else if(curr->data == 1) o = o->next = curr;
        else t = t->next = curr;
        curr = curr->next;
    }
    z->next = one.next ? one.next : two.next;
    o->next = two.next;
    t->next = nullptr;
    return zero.next;
}
```

14. ARTICLE: Quick Sort Algorithm is best for array or linked list?
    - Explanation: Array, due to random access.
15. ARTICLE: Merge Sort Algorithm is best for array or linked list?
    - Explanation: Linked list, no extra space for merge.
16. Double a number represented as a linked list
    - Approach: Reverse, double with carry

```
class Solution {
public:
    ListNode* doubleIt(ListNode* head) {
        head = reverse(head);
        ListNode* curr = head;
        int carry = 0;
        while(curr) {
            int val = curr->val * 2 + carry;
            curr->val = val % 10;
            carry = val / 10;
            if(!curr->next && carry) curr->next = new ListNode(0);
```

```
            curr = curr->next;
        }
        return reverse(head);
    }
    ListNode* reverse(ListNode* head) {
        ListNode* prev = nullptr, *curr = head, *next;
        while(curr) {
            next = curr->next;
            curr->next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }
};
```

17. Swapping nodes in a linked list
    - Approach: Swap pairs

```
class Solution {
public:
    ListNode* swapPairs(ListNode* head) {
        if(!head || !head->next) return head;
        ListNode* next = head->next;
        head->next = swapPairs(next->next);
        next->next = head;
        return next;
    }
};
```

18. Remove zero sum consecutive nodes from linked list
    - Approach: Prefix sum map

```
class Solution {
public:
    ListNode* removeZeroSumSublists(ListNode* head) {
        ListNode dummy(0);
        dummy.next = head;
        unordered_map<int, ListNode*> mp;
        int prefix = 0;
        ListNode* curr = &dummy;
        while(curr) {
            prefix += curr->val;
            if(mp.count(prefix)) mp[prefix]->next = curr->next;
```

```
            else mp[prefix] = curr;
            curr = curr->next;
        }
        return dummy.next;
    }
};
```

WEEK 11 TOPIC: Stack - Class 01

CLASS NOTES: 32

2. Stack implementation using dynamic array
    - Push, pop, isEmpty, getTop, getSize

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

class Stack {
private:
    vector<int> data;
public:
    void push(int x) {
        data.push_back(x);
    }
    void pop() {
        if(!isEmpty()) data.pop_back();
    }
    bool isEmpty() {
        return data.empty();
    }
    int getTop() {
        if(isEmpty()) return -1;
        return data.back();
    }
    int getSize() {
        return data.size();
    }
};

int main() {
    Stack s;
    s.push(1);
    cout << s.getTop();
```

```
    return 0;
}
```

Output: `1`

3. Problem 1: Reverse string using stack

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

string reverseStringStack(string s) {
    stack<char> st;
    for(char c : s) st.push(c);
    string res = "";
    while(!st.empty()) {
        res += st.top();
        st.pop();
    }
    return res;
}

int main() {
    cout << reverseStringStack("hello");
    return 0;
}
```

Output: `olleh`

4. Problem 2: Middle element of a stack
    - Approach 1: Recursion and backtracking

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

int getMiddle(stack<int> st, int size, int curr) {
    if(curr == size / 2 + 1) return st.top();
    int top = st.top();
    st.pop();
    int mid = getMiddle(st, size, curr + 1);
    st.push(top);
    return mid;
}

int main() {
    stack<int> st;
    st.push(1); st.push(2); st.push(3);
```

```
    cout << getMiddle(st, 3, 1);
    return 0;
}
```

Output: `2`

5. Problem 3: Insert at bottom of a stack
    - Approach 1: Recursion and backtracking

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

void insertBottom(stack<int>& st, int x) {
    if(st.empty()) {
        st.push(x);
        return;
    }
    int top = st.top();
    st.pop();
    insertBottom(st, x);
    st.push(top);
}

int main() {
    stack<int> st;
    st.push(1); st.push(2);
    insertBottom(st, 3);
    while(!st.empty()) {
        cout << st.top() << " ";
        st.pop();
    }
    return 0;
}
```

Output: `2 1 3`

6. Problem 4: Reverse a stack
    - Approach 1: Recursion and backtracking

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

void reverseStack(stack<int>& st) {
    if(st.empty()) return;
    int top = st.top();
    st.pop();
```

```
    reverseStack(st);
    insertBottom(st, top); // from above
}

int main() {
    stack<int> st;
    st.push(1); st.push(2); st.push(3);
    reverseStack(st);
    while(!st.empty()) {
        cout << st.top() << " ";
        st.pop();
    }
    return 0;
}
```

Output: `1 2 3`

7. Problem 5: Insert in a sorted stack
    - Approach 1: Recursion and backtracking

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

void insertSorted(stack<int>& st, int x) {
    if(st.empty() || st.top() < x) {
        st.push(x);
        return;
    }
    int top = st.top();
    st.pop();
    insertSorted(st, x);
    st.push(top);
}

int main() {
    stack<int> st;
    st.push(1); st.push(3); st.push(4);
    insertSorted(st, 2);
    while(!st.empty()) {
        cout << st.top() << " ";
        st.pop();
    }
    return 0;
}
```

Output: `4 3 2 1`

8. Problem 6: Sort a stack
    - Approach 1: Recursion and backtracking

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

void sortStack(stack<int>& st) {
    if(st.empty()) return;
    int top = st.top();
    st.pop();
    sortStack(st);
    insertSorted(st, top); // from above
}

int main() {
    stack<int> st;
    st.push(4); st.push(2); st.push(3); st.push(1);
    sortStack(st);
    while(!st.empty()) {
        cout << st.top() << " ";
        st.pop();
    }
    return 0;
}
```

Output: `4 3 2 1`

WEEK 11 TOPIC: Stack - Class 02

CLASS NOTES: 33

1. Problem 1: Implementation of Two Stack in an Array

```
class twoStacks {
private:
    int* arr;
    int size;
    int top1, top2;
public:
    twoStacks(int n) {
        size = n;
        arr = new int[n];
        top1 = -1;
        top2 = size;
```

```
    }
    void push1(int x) {
        if(top1 + 1 == top2) return;
        arr[++top1] = x;
    }
    void push2(int x) {
        if(top2 - 1 == top1) return;
        arr[--top2] = x;
    }
    int pop1() {
        if(top1 < 0) return -1;
        return arr[top1--];
    }
    int pop2() {
        if(top2 >= size) return -1;
        return arr[top2++];
    }
};
```

2. Problem 2: Valid Parentheses (Leetcode-20)
    - Approach: Stack

```
class Solution {
public:
    bool isValid(string s) {
        stack<char> st;
        for(char c : s) {
            if(c == '(' || c == '{' || c == '[') st.push(c);
            else {
                if(st.empty()) return false;
                char top = st.top();
                st.pop();
                if((c == ')' && top != '(') || (c == '}' && top != '{') || (c
== ']' && top != '[')) return false;
            }
        }
        return st.empty();
    }
};
```

3. Problem 3: Remove Redundant Brackets
    - Approach: Stack for operators
        - Traverse, count operators between brackets.

WEEK 11 TOPIC: Stack - Class 03

CLASS NOTES: 34

Problem 1: Implement a minStack (Leetcode-155)

- Approach: Pair with min

```
class MinStack {
private:
    stack<pair<int, int>> st;
public:
    MinStack() {

    }

    void push(int val) {
        int mn = st.empty() ? val : min(val, st.top().second);
        st.push({val, mn});
    }

    void pop() {
        st.pop();
    }

    int top() {
        return st.top().first;
    }

    int getMin() {
        return st.top().second;
    }
};
```

Problem 2: Next smaller element

- Approach: Stack monotonic

```
vector<int> nextSmaller(vector<int> arr) {
    stack<int> st;
    st.push(-1);
    vector<int> res(arr.size());
    for(int i = arr.size() - 1; i >= 0; i--) {
        while(st.top() >= arr[i]) st.pop();
        res[i] = st.top();
```

```
            st.push(arr[i]);
        }
        return res;
    }
```

Problem 3: Prev smaller element

- Similar, reverse traversal.

Problem 4: Largest Rectangle Area in Histogram (Leectcode-84)

- Approach: Stack for left right smaller

```
class Solution {
public:
    int largestRectangleArea(vector<int>& heights) {
        int n = heights.size();
        vector<int> left(n), right(n);
        stack<int> st;
        for(int i = 0; i < n; i++) {
            while(!st.empty() && heights[st.top()] >= heights[i]) st.pop();
            left[i] = st.empty() ? -1 : st.top();
            st.push(i);
        }
        while(!st.empty()) st.pop();
        for(int i = n-1; i >= 0; i--) {
            while(!st.empty() && heights[st.top()] >= heights[i]) st.pop();
            right[i] = st.empty() ? n : st.top();
            st.push(i);
        }
        int maxArea = 0;
        for(int i = 0; i < n; i++) maxArea = max(maxArea, heights[i] * (right[i]
- left[i] - 1));
        return maxArea;
    }
};
```

CLASS HOMEWORK: 34

1. Minimum Bracket Reversal
    - Approach: Stack for unbalanced

```
int countBracketReversals(string s) {
    stack<char> st;
    for(char c : s) {
        if(c == '{') st.push(c);
```

```
        else if(!st.empty() && st.top() == '{') st.pop();
        else st.push(c);
    }
    if(st.size() % 2 != 0) return -1;
    int count = 0;
    while(!st.empty()) {
        char a = st.top(); st.pop();
        char b = st.top(); st.pop();
        if(a == b) count++;
        else count += 2;
    }
    return count;
}
```

2. Remove All Adjacent Duplicates In String
    - Already.
3. Celebrity Problem
    - Approach: Stack or two pass

```
int celebrity(vector<vector<int>>& M, int n) {
    int candidate = 0;
    for(int i = 1; i < n; i++) {
        if(M[candidate][i] == 1) candidate = i;
    }
    for(int i = 0; i < n; i++) {
        if(i != candidate && (M[candidate][i] == 1 || M[i][candidate] == 0))
return -1;
    }
    return candidate;
}
```

4. Next greater element in Linked List (Leetcode)
    - Approach: Stack, reverse traversal.
5. N Stacks in an Array
    - Approach: Array with top array and next array.
6. Online Stock Span (Leetcode)
    - Approach: Stack with pair index span.
7. Check If Word Is Valid After Substitutions (Leetcode)
    - Approach: Stack for "abc".
8. Decode Strings (Leetcode)

- Approach: Stack for [] and numbers.

9. Car Fleet I (Leetcode)
    - Approach: Sort, stack for fleets.
10. Car Fleet - II (Leetcode)
    - Approach: Stack for collision times.
11. Simplify Path (Leetcode)
    - Approach: Stack for directories.
12. Max rectangle in Binary Matrix with all 1s
    - Approach: Histogram for each row.
13. Daily Temperatures
    - Approach: Stack for next warmer.
14. Remove K Digits
    - Approach: Stack for monotonous increasing.
15. Minimum Add To Make Parentheses Valid
    - Approach: Count open close.
16. Longest Valid Parentheses
    - Approach: Stack for indices.
17. Asteroid Collision
    - Approach: Stack for left right.
18. Design Browser History
    - Approach: Two stacks or vector with curr.
19. Final Prices With a Special Discount in a Shop (Leetcode-1475)
    - Approach: Stack for next smaller.

WEEK 12 TOPIC: Queue - Class 01

CLASS NOTES: 35

1. What is Queue and STL Queue
    - queue q;
2. Queue Implementation using a Dynamic Array
    - Similar to stack, but front rear.
3. Circular Queue Implementation
    - Modulo for wrap.

4. Circular Double Ended Queue Implementation
    - Deque.

WEEK 12 TOPIC: Queue - Class 02

CLASS NOTES: 36

1. Reverse a queue
    - Approach: Stack

```
void reverseQueue(queue<int>& q) {
    stack<int> st;
    while(!q.empty()) {
        st.push(q.front());
        q.pop();
    }
    while(!st.empty()) {
        q.push(st.top());
        st.pop();
    }
}
```

2. Reverse 'k' element in a queue
    - Approach: Stack for k, then enqueue back.
3. Interleave first and second half of a queue
    - Approach: Two queues.
4. First negative integer in every window of 'k'
    - Approach: Deque for indices.

WEEK 12 TOPIC: Queue - Class 03

CLASS NOTES: 37

1. First Non Repeating/Unique Character in a String (Leetcode-387)
    - Approach: Queue and count.
2. Gas Station (Leetcode-134)
    - Approach: Total gas cost, find start.
3. Sliding Window Maximum (Leetcode-239)
    - Approach: Deque for decreasing.

HOMEWORK NOTES: 37

1. Implement Queue using Stack
    - Two stacks.
2. Implement Stacks using Queue
    - Two queues.
3. Implement "K" queues in an array
    - Similar to n stacks.
4. Sum of min/max element of all subarray of size "k"
    - Deque for min/max.
5. Number of Recent Calls
    - Queue for timestamps.
6. First Unique Character in a String
    - Already.
7. Number of People Aware of a Secret
    - DP or queue.
8. Maximum Sum Circular Subarray
    - Kadane twice.
9. Find the Winner of the Circular Game
    - Queue simulation.
10. Reveal Cards In Increasing Order
    - Deque simulation.
11. Product of the Last K Numbers
    - Vector of prefix products.
12. Longest Continuous Subarray With Absolute Diff Less Than or Equal to Limit
    - Two deques for max min.
13. Delivering Boxes from Storage to Ports
    - DP with optimization.

WEEK 13 TOPIC: Binary Tree - Class 01

CLASS NOTES: 38

3. Implementation of binary tree
    - Class TreeNode.
4. Three binary tree traversals

- Pre: root left right

```
void preOrder(TreeNode* root) {
    if(!root) return;
    cout << root->val << " ";
    preOrder(root->left);
    preOrder(root->right);
}
```

- In: left root right

```
void inOrder(TreeNode* root) {
    if(!root) return;
    inOrder(root->left);
    cout << root->val << " ";
    inOrder(root->right);
}
```

- Post: left right root

```
void postOrder(TreeNode* root) {
    if(!root) return;
    postOrder(root->left);
    postOrder(root->right);
    cout << root->val << " ";
}
```

5. Level order traversal in a line
    - Queue, process level.
6. Level order traversal in level wise
    - Queue, size for level.
7. Height of a binary tree (Leetcode-104)
    - Approach: Recursion

```
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if(!root) return 0;
        return 1 + max(maxDepth(root->left), maxDepth(root->right));
    }
};
```

8. Diameter of binary tree (Leetcode-543)
    - Approach: Max left + right height

```
class Solution {
public:
    int height(TreeNode* root) {
        if(!root) return 0;
        return 1 + max(height(root->left), height(root->right));
    }
    int diameterOfBinaryTree(TreeNode* root) {
        if(!root) return 0;
        int lh = height(root->left);
        int rh = height(root->right);
        int ld = diameterOfBinaryTree(root->left);
        int rd = diameterOfBinaryTree(root->right);
        return max(lh + rh, max(ld, rd));
    }
};
```

HOMEWORK NOTES: 38

1. Generics tree
    - Tree with variable children.
2. READ ARTICLE: Skew tree
    - Left or right only.
3. READ ARTICLE: BFS and DFS Algorithm
    - BFS queue, DFS recursion/stack.
4. READ ARTICLE: Complete and perfect binary tree
    - Complete: filled left to right, perfect: all levels full.

WEEK 13 TOPIC: Binary Tree - Class 02

CLASS NOTES: 39

1. Balanced Binary Tree (Leetcode-110)
    - Approach: Height diff <=1

```
class Solution {
public:
    int height(TreeNode* root) {
        if(!root) return 0;
        return 1 + max(height(root->left), height(root->right));
    }
    bool isBalanced(TreeNode* root) {
        if(!root) return true;
```

```
        int lh = height(root->left);
        int rh = height(root->right);
        if(abs(lh - rh) > 1) return false;
        return isBalanced(root->left) && isBalanced(root->right);
    }
};
```

2. Lowest Common Ancestor of a Binary Tree (Leetcode-236)

- Approach: Recursion

```
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q)
{
        if(!root || root == p || root == q) return root;
        TreeNode* left = lowestCommonAncestor(root->left, p, q);
        TreeNode* right = lowestCommonAncestor(root->right, p, q);
        if(!left) return right;
        if(!right) return left;
        return root;
    }
};
```

3. Path Sum (Leetcode-112)

- Approach: Recursion subtract

```
class Solution {
public:
    bool hasPathSum(TreeNode* root, int targetSum) {
        if(!root) return false;
        if(!root->left && !root->right) return root->val == targetSum;
        return hasPathSum(root->left, targetSum - root->val) ||
hasPathSum(root->right, targetSum - root->val);
    }
};
```

4. Path Sum II (Leetcode-113)

- Approach: Backtracking

```
class Solution {
public:
    void helper(TreeNode* root, int target, vector<int> current,
vector<vector<int>>& res) {
        if(!root) return;
        current.push_back(root->val);
```

```
        if(!root->left && !root->right && target == root->val)
res.push_back(current);
        helper(root->left, target - root->val, current, res);
        helper(root->right, target - root->val, current, res);
        current.pop_back();
    }
    vector<vector<int>> pathSum(TreeNode* root, int targetSum) {
        vector<vector<int>> res;
        vector<int> current;
        helper(root, targetSum, current, res);
        return res;
    }
};
```

5. K-th ancestor of a node in Binary Tree (GFG)
    - Approach: Recursion level
6. Construct Binary Tree from Inorder and Preorder Traversal (Leetcode-105)
    - Approach: Recursion with index

```
class Solution {
public:
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        unordered_map<int, int> mp;
        for(int i = 0; i < inorder.size(); i++) mp[inorder[i]] = i;
        int preIdx = 0;
        return helper(preorder, mp, preIdx, 0, inorder.size() - 1);
    }
    TreeNode* helper(vector<int>& pre, unordered_map<int, int>& mp, int&
preIdx, int inStart, int inEnd) {
        if(inStart > inEnd) return nullptr;
        TreeNode* root = new TreeNode(pre[preIdx++]);
        int inIdx = mp[root->val];
        root->left = helper(pre, mp, preIdx, inStart, inIdx - 1);
        root->right = helper(pre, mp, preIdx, inIdx + 1, inEnd);
        return root;
    }
};
```

7. Construct Binary Tree from Inorder and Postorder Traversal (Leetcode-106)
    - Similar, postIdx from end.

HOMEWORK NOTES: 39

1. K-th ancestor of a node in Binary Tree (GFG)

- Find path, k from end.

WEEK 13 TOPIC: Binary Tree - Class 03

CLASS NOTES: 40

1. Left view of binary tree
    - Approach: Level order, first per level.
2. Right view of binary tree
    - Last per level.
3. Top view of binary tree
    - Vertical order first.
4. Bottom view of binary tree
    - Vertical order last.
5. Boundary traversal of binary tree
    - Left boundary, leaves, right boundary reverse.

HOMEWORK NOTES: 40

1. FW to find diameter of binary tree
    - O(n) with height.
2. FW to find height balanced tree
    - Height diff in recursion.
3. Check two trees are identical
    - Recursion val left right.
4. Symmetric Tree (Mirror Subtrees)
    - Mirror left right.
5. Zig-zag traversal
    - Level order alternate reverse.
6. Transform to sum tree
    - Postorder sum children.
7. Diagonal Traversal
    - Queue with diagonal level.
8. Vertical Traversal
    - Map with level.

9. K-Sum Paths
    - Backtrack path sum.
10. Morris Traversal
    - Threaded for inorder.
11. Flatten a binary tree into LinkedList
    - Recursion stack or morris.
12. Sum of longest bloodline of tree

- Recursion max length sum.

13. Maximum sum of non-adjacent nodes

- DP pair include exclude.

14. Burning Tree

- BFS from target.

15. Find Duplicate Subtrees

- Serialize and map.

16. Left view of binary tree using level order traversal

- Already.

WEEK 14 TOPIC: Binary Search Tree - Class 01

CLASS NOTES: 41

2. Create binary search tree
    - Recursion insert.
3. Traversals of binary search tree
    - Inorder sorted.
4. Min and Max value in BST
    - Leftmost min, rightmost max.
5. Target value present or not in BST
    - Binary search.
6. Delete node from BST
    - Find, replace with inorder successor if both children.

WEEK 14 TOPIC: Binary Search Tree - Class 02

CLASS NOTES: 42

1. Construct BST from Inorder (GFG)
    - Recursion mid root.
2. Validate BST (Leetcode-98)
    - Inorder sorted check.
3. Lowest Common Ancestor of a BST (Leetcode-235)
    - Binary search.
4. Kth Smallest Element in a BST (Leetcode-230)
    - Inorder count.
5. Two Sum IV - Input is a BST (Leetcode-653)
    - Inorder to array, two sum.

WEEK 14 TOPIC: Binary Search Tree - Class 03

CLASS NOTES: 43

1. Convert BST into Sorted Double Linked List
    - Inorder, prev next.
2. Convert Sorted Double Linked List into BST
    - Mid root, recursion.

HOMEWORK NOTES: 43

1. Inorder Successor in BST
    - Right min or parent.
2. Inorder Predecessor in BST
    - Left max.
3. Build BST using Preorder Traversal
    - Recursion with upper lower.
4. Brothers from Different roots
    - Two BST, find pairs sum x.
5. Convert BST to a Balanced BST
    - Inorder to array, build balanced.
6. Find the Median of BST
    - Inorder count.

7. Check BST has Dead End
    - Leaf with no gap.
8. Count BST Nodes lying in a Range
    - Recursion count.
9. Flatten BST to Sorted LL
    - Already.
10. Replace elements with the least Greater elements to it Right
    - Reverse inorder.
11. Valid BST from Preorder
    - Stack monotonic.
12. Merge two BSTs
    - Inorder merge.

WEEK 15 TOPIC: Heap - Class 01

CLASS NOTES: 44

3. Insertion to heap
    - Push, heapify up.
4. Deletion from heap
    - Pop, heapify down.
5. Heapify using recursion
    - Bottom up.
6. Convert array to heap
    - Heapify from n/2.
7. Heap sort
    - Build, extract max.

WEEK 15 TOPIC: Heap - Class 02

CLASS NOTES: 45

1. C++ STL Priority Queue "MAX HEAP"
    - priority_queue pq;
2. C++ STL Priority Queue "MIN HEAP"
    - priority_queue<int, vector, greater> pq;

3. Kth Smallest Element in an Array using Max Heap (GFG)
    - Max heap size k.
4. Kth Largest Element in an Array using Min Heap (GFG)
    - Min heap size k.
5. Check if a given Complete Binary Tree is a Max Heap or not? (GFG)
    - Check complete and max property.
6. Check Whether a Binary Tree is a Complete Binary Tree or Not? (Leetcode-958)
    - Level order count.
7. Convert Given Combination of CBT and BST into a Valid Max Heap (GFG)
    - Heapify.

WEEK 15 TOPIC: Heap - Class 03

CLASS NOTES: 46

1. Merge K Sorted Arrays (GFG)
    - Priority queue with triple.
2. Merge K Sorted Linked Lists (Leetcode-23)
    - Priority queue with nodes.
3. Smallest Range in K Lists (Leetcode-632)
    - Priority queue min range.

WEEK 15 TOPIC: Heap - Class 04

CLASS NOTES: 47

1. Remove Stones to Minimize the Total (Leetcode-1962)
    - Max heap, reduce half.
2. Reorganize String (Leetcode-767)
    - Already.
3. Longest Happy String (Leetcode-1405)
    - Priority queue greedy.
4. Median in a Stream (CodingNinjas)
    - Two heaps max min.

HOMEWORK NOTES: 47

1. Check If Binary Tree is Heap

    - Complete and max.
2. Merge Two Binary Max Heap
    - Combine, heapify.
3. K-Closest points to the origin
    - Max heap distance.
4. Get Biggest Three Rhombus Sums In A Grid
    - Brute all rhombus.
5. Minimum Difference in Sums After Removal of Elements
    - Prefix min max.
6. Minimum Number of Refueling Stops
    - Priority queue greedy.
7. Sliding Window Maximum
    - Already.

WEEK 16 TOPIC: Hashmaps & Tries - Class 01

CLASS NOTES: 48

3. Implement C++ STL Unordered Map
    - Use unordered_map.

PROBLEM 1: Store all character frequency of a string

- Unordered map char int.

PROBLEM 2: Reorganize String (Leetcode-767)

- Already.

PROBLEM 3: Linked List Cycle (Leetcode-141)

- Already.

WEEK 16 TOPIC: Hashmaps & Tries - Class 02

CLASS NOTES: 49

2. Create Trie Node
    - Array 26 or map.
3. Insertion Method of Trie
4. Searching Method of Trie

5. Deletion Method of Trie

HOMEWORK NOTES: 49

1. Print All Words of Given Prefix String
    - Trie search, DFS for all.

WEEK 16 TOPIC: Hashmaps & Tries - Class 03

CLASS NOTES: 50

1. Print All Words of Given Prefix String - I
    - Trie.
2. Print All Words of Given Prefix String - II
    - Trie.
3. Longest Common Prefix (Leetcode-14)
    - Already.

HOMEWORK NOTES: 50

1. Array Subset of Another Array
    - Hash set.
2. Union of Two Linked Lists
    - Set.
3. Intersection of Two Linked Lists
    - Already.
4. Sum Equals To Sum
    - Pair sum.
5. Largest Subarray with 0 Sum
    - Prefix map.
6. Largest Subarray of 0’s and 1’s
    - Treat 0 as -1, prefix.
7. Valid Anagram
    - Already.
8. Replace Words
    - Trie for dictionary.
9. Top K Frequent Words

- Map, priority queue.

10. Camelcase Matching

- Two pointers.

11. Palindrome Pairs

- Trie for reverse.

WEEK 17 TOPIC: Dynamic Programming - Class 01

CLASS NOTES: 51

4. Fibonacci (Leetcode-509)
    - Memo

```
class Solution {
public:
    int fib(int n) {
        if(n <= 1) return n;
        vector<int> dp(n+1, -1);
        if(dp[n] != -1) return dp[n];
        return dp[n] = fib(n-1) + fib(n-2);
    }
};
```

    - Tabulation

```
class Solution {
public:
    int fib(int n) {
        if(n <= 1) return n;
        vector<int> dp(n+1);
        dp[0] = 0;
        dp[1] = 1;
        for(int i = 2; i <= n; i++) dp[i] = dp[i-1] + dp[i-2];
        return dp[n];
    }
};
```

    - Space opt
        - Two variables.
5. Cut Segment (GFG)
    - Max cuts.

WEEK 17 TOPIC: Dynamic Programming - Class 02

CLASS NOTES: 52

1. House Robber (Leetcode-198)
    - Already.
2. Coin Change (Leetcode-322)
    - DP unbounded knaps.

WEEK 17 TOPIC: Dynamic Programming - Class 03

CLASS NOTES: 53

1. Painting Fence Algorithm (GFG)
    - Already.
2. 0/1 Knapsack Problem (GFG)
    - Already.

WEEK 17 TOPIC: Dynamic Programming - Class 04

CLASS NOTES: 54

1. Longest Common Subsequence (Leetcode-1143)
    - DP 2D.
2. Longest Palindrome Subsequence (Leetcode-516)
    - LCS with reverse.
3. Edit Distance (Leetcode-72)
    - DP 2D.

WEEK 18 TOPIC: Dynamic Programming - Class 05

CLASS NOTES: 55

1. Longest Increasing Subsequence (Leetcode-300)
    - DP n^2 or binary.
2. Maximum Height by Stacking Cuboids (Leetcode-1691)
    - DP sort dimensions.
3. Russian Doll Envelopes (Leetcode-354)
    - LIS on height after sort width.
4. Longest Common Subsequence (Leetcode-1143)

- Already.

WEEK 18 TOPIC: Dynamic Programming - Class 06

CLASS NOTES: 56

1. Guess Number Higher or Lower II (Leetcode-375)
    - DP min max.
2. Minimum Cost Tree From Leaf Values (Leetcode-1130)
    - DP max product.

WEEK 18 TOPIC: Dynamic Programming - Class 07

CLASS NOTES: 57

1. Partition Equal Subset Sum (Leetcode-416)
    - DP subset sum.
2. Number of Dice Rolls With Target Sum (Leetcode-1155)
    - Already.

WEEK 18 TOPIC: Dynamic Programming - Assignments

All Leetcode DP problems, similar approaches.

WEEK 19 TOPIC: Graphs - Class 01

CLASS NOTES: 58

4. Graph Creation
    - Adj list vector<vector>.
5. Traverse the graph
    - BFS queue visited.
    - DFS recursion visited.

HOMEWORK NOTES: 58

Time space for graph, BFS DFS, components.

WEEK 19 TOPIC: Graphs - Class 02

CLASS NOTES: 59

1. Cycle undirected BFS
    - Parent check.

2. Cycle undirected DFS
    - Parent check.
3. Cycle directed DFS
    - Vis and path vis.

WEEK 19 TOPIC: Graphs - Class 03

CLASS NOTES: 60

1. Topo sort DFS
    - Stack post order.
2. Topo sort BFS
    - Kahn indegree.
3. Cycle directed BFS
    - Indegree 0 count != v.
4. Shortest path undirected BFS
    - Dist array.

WEEK 19 TOPIC: Graphs - Class 04

CLASS NOTES: 61

1. Shortest path weighted directed DFS
    - Topo + dist.
2. Shortest path weighted undirected Dijkstra
    - Priority queue.

WEEK 20 TOPIC: Graphs - Class 05

CLASS NOTES: 62

1. Course Schedule (Leetcode-207)
    - Cycle detection.
2. Course Schedule II (Leetcode-210)
    - Topo sort.
3. Path With Minimum Effort (Leetcode-1631)
    - Dijkstra with max effort.

WEEK 20 TOPIC: Graphs - Class 06

CLASS NOTES: 63

1. Number of Provinces (Leetcode-547)
    - DFS components.
2. Number of Islands (Leetcode-200)
    - DFS flood.
3. Flood Fill (Leetcode-733)
    - DFS color.
4. Rotting Oranges (Leetcode-994)
    - BFS multi source.

WEEK 20 TOPIC: Graphs - Class 07

CLASS NOTES: 64

1. Dijkstra
    - Already.
2. Bellman Ford
    - Relax v-1, negative cycle.
3. Floyd Warshall
    - All pair shortest.
4. Kosaraju
    - DFS transpose stack.
5. Tarjans Algorithm for Bridges (Leetcode-1192)
    - DFS disc low.

WEEK 20 TOPIC: Graphs - Assignments

Prim, Kruskal MST, other problems similar.

WEEK 21 TOPIC: Bit Manipulation

CLASS NOTES: 65

2. Check even or odd number
    - n & 1
3. Get Ith bit from right side
    - n & (1 << i)

4. Set Ith bit from right side
    - n | (1 << i)
5. Clear Ith bit from right side
    - n & ~(1 << i)
6. Update Ith bit from right side
    - Clear then set.
7. Single number (Leetcode-136)
    - XOR all.
8. Clear n bits from last
    - n & ~((1 << n) - 1)
9. Check power of two
    - n & (n-1) == 0
10. Count set bits
    - Brian Kernighan n & (n-1)
11. Clear bits in range
    - Mask.
12. Subsequence of a string "Include & Exclude" (Leetcode-78)
    - Bit mask for subsets.
13. Pow(x, n) (Leetcode-50)
    - Binary exp.
14. Single number II (Leetcode-137)
    - Bit count mod 3.
15. Single number III (Leetcode-260)
    - XOR, group by bit.
16. Binary to decimal
    - Already.
17. Decimal to binary
    - Bit shift.
18. Count total jump from source to destination
    - ? Assumption frog jump, DP.

WEEK 21 TOPIC: Sliding Window Technique

CLASS NOTES: 66

2. Fixed size window
    - Sliding Window Maximum (Leetcode-239)
      Already.
    - Max Sum Subarray of size K (GFG)
      Sliding sum.
3. Variable size window
    - Minimum Size Subarray Sum (Leetcode-209)
      Two pointers.
    - Binary Subarrays With Sum (Leetcode-930)
      Prefix at most.
4. Two pointer approach
    - Two Sum (Leetcode-1)
      Already.
5. Optimization approach
    - Minimum Window Substring (Leetcode-76)
      Sliding with count.