📚 Day 13 Study Notes

1. Aptitude: Pipes & Cisterns

Goal: Solve rate-based problems involving filling and emptying tanks.

Key Concepts

This topic is conceptually identical to **Time & Work**, but with a twist: "Negative Work" (leaks/outlets).

- **Inlet Pipe:** Adds water. If it fills a tank in x hours, work done in 1 hour = $+\frac{1}{x}$.
- **Outlet Pipe (Leak):** Removes water. If it empties a tank in y hours, work done in 1 hour = $-\frac{1}{y}$.

Formula

If an inlet fills in x hours and an outlet empties in y hours, the net part filled in 1 hour is:

$$\text{Net Work} = \frac{1}{x} - \frac{1}{y}$$

- If the result is positive, the tank gets filled.
- If negative, the tank gets emptied.

Example Problem

Question: Pipe A fills a tank in 4 hours, Pipe B fills it in 6 hours. If both are open, how long to fill?
Solution:

$$\frac{1}{T} = \frac{1}{4} + \frac{1}{6} = \frac{3+2}{12} = \frac{5}{12}$$

$$T = \frac{12}{5} = 2.4 \text{ hours (or 2 hours 24 mins)}$$

2. Programming: Merge Two Sorted Lists

Goal: Combine two already sorted arrays into a single sorted array efficiently ($O(N+M)$).

Logic (Two Pointer Approach)

Since the input lists are sorted, we don't need to re-sort the final list.

1. Initialize pointers `i = 0` (for list A) and `j = 0` (for list B).
2. Compare `A[i]` and `B[j]` .

  - If `A[i] < B[j]` : Append `A[i]` to result, increment `i` .
  - Else: Append `B[j]` to result, increment `j` .
3. Repeat until one list is exhausted.
4. Append the remaining elements of the non-exhausted list.

Code Snippet (Python)

```python
def merge_sorted_lists(list1, list2):
    merged = []
    i, j = 0, 0

    # Compare elements while both lists have items
    while i < len(list1) and j < len(list2):
        if list1[i] < list2[j]:
            merged.append(list1[i])
            i += 1
        else:
            merged.append(list2[j])
            j += 1

    # Add remaining elements from either list
    merged.extend(list1[i:])
    merged.extend(list2[j:])

    return merged

l1 = [1, 3, 5]
l2 = [2, 4, 6, 8]
print(merge_sorted_lists(l1, l2)) # [1, 2, 3, 4, 5, 6, 8]
```

3. Concept: Python Exception Handling

Goal: Prevent program crashes when runtime errors occur (e.g., division by zero, file not found).

Keywords

- `try` : Block of code to test for errors.
- `except` : Block of code to handle the error.
- `else` : Runs if no errors occurred.
- `finally` : Runs regardless of the result (good for cleanup/closing files).

Example

```python
try:
    num = int(input("Enter a number: "))
    result = 10 / num
except ZeroDivisionError:
    print("Error: Cannot divide by zero.")
except ValueError:
    print("Error: Please enter a valid integer.")
```

```
else:
    print(f"Result is {result}")
finally:
    print("Execution complete.")
```

4. C/C++ Concept: Error Handling

Goal: Manage errors in lower-level environments.

C Style (Return Codes & `errno`)

C does not have built-in exceptions. It uses return values (e.g., returning `-1` or `NULL`) and the global `errno` variable.

```
FILE *f = fopen("nonexistent.txt", "r");
if (f == NULL) {
    perror("Error opening file"); // Prints descriptive error based on errno
}
```

C++ Style (`try-catch`)

C++ supports structured exception handling similar to Python/Java.

```
try {
    int age = 15;
    if (age < 18) {
        throw "Access denied - You must be at least 18 years old.";
    }
}
catch (const char* msg) {
    cerr << "Error: " << msg << endl;
}
```

5. SQL: IS NULL Operator

Goal: Correctly filter for missing or undefined values.

The NULL Trap

You **cannot** use standard comparison operators (`=` , `!=`) with `NULL` .

- Wrong: `WHERE Email = NULL` (Always returns False/Unknown).
- Correct: `WHERE Email IS NULL` .

Syntax

- **Find missing values:**

```
SELECT * FROM Employees WHERE PhoneNumber IS NULL;
```

- **Find non-missing values:**

```
SELECT * FROM Employees WHERE PhoneNumber IS NOT NULL;
```

Handling NULL in Results

Use `COALESCE()` or `IFNULL()` to replace NULL with a default value for display.

```
SELECT Name, COALESCE(PhoneNumber, 'No Phone Provided')
FROM Employees;
```