COL1000: Introduction to Programming

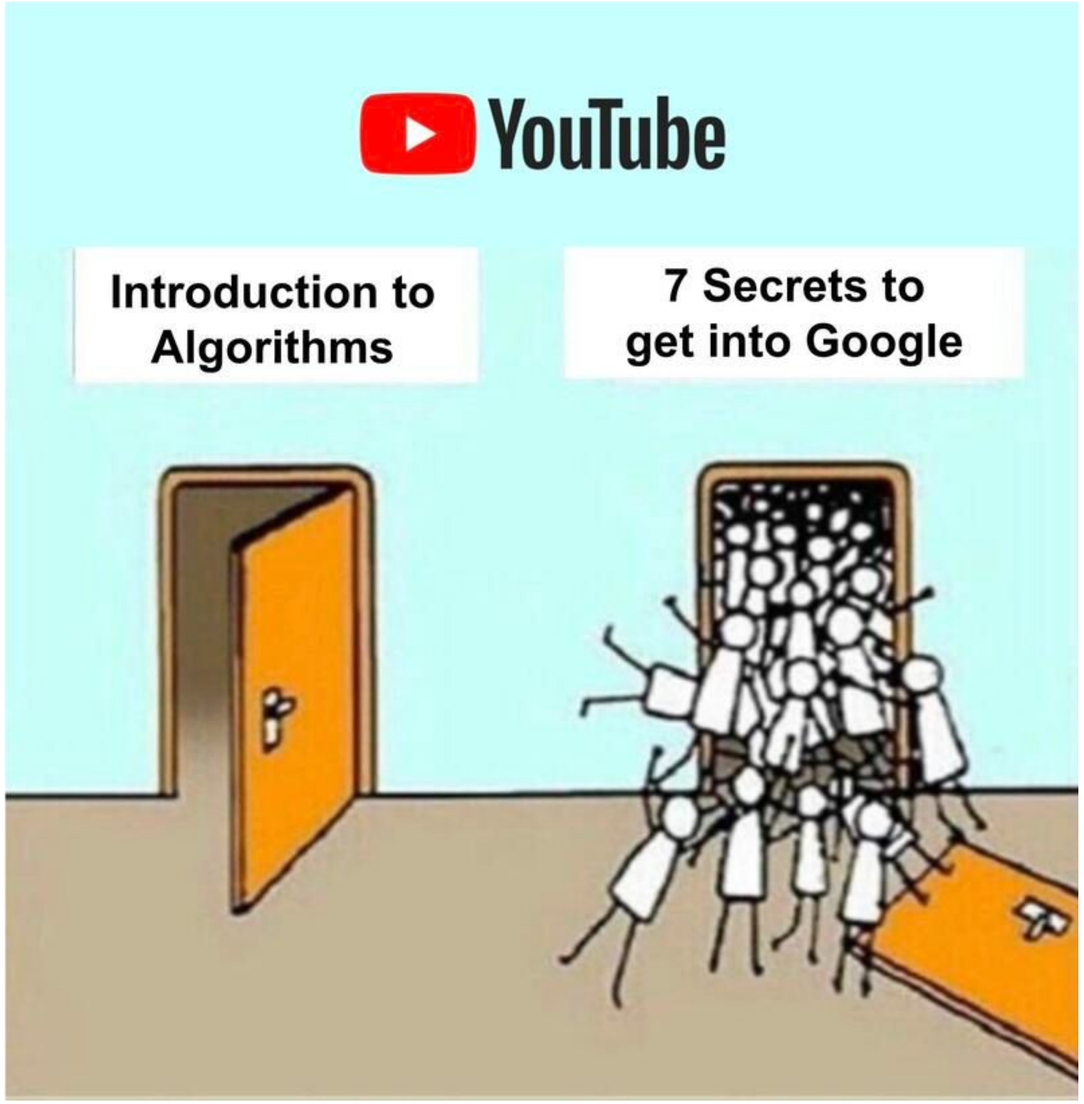
Algorithmic Thinking (Merge & Merge Sort)

Subodh Sharma | Lec 33 | Nov 4

Other Examples of Greedy

- Classroom Scheduling
- Huffman Encoding
- Minimum Spanning Tree (on graphs)
- Load Balancer
- Set Cover
- 0/1 Knapsack
- ...

Divide and Conquer



Divide and Conquer: Merge Sort

- **Divide:** Split the problem into smaller subproblems
- **Conquer:** The subproblems are solved (usually recursively) by applying the same solution
- **Combine:** Fusing the results together of the subproblems.

- **Problem:** sort A = [8, 3, 1, 7, 0, 10, 2] using MergeSort

Divide and Conquer: Merge Sort

- **Problem:** sort A = [8, 3, 1, 7, 0, 10, 2] using MergeSort
- **Algorithm Run (Trace):**
  - **Recursively split the problem to smaller problems**
    - Left: [8,3,1] Right:[7,0,10,2]
    - **Sort Left:** [8,3,1]
      - **Split again:** Left [8] Right[3,1]
      - **Sort Left —> sorted —> [8] Sort Right:** [3,1]
        - **Split again:** Left [3] Right[1]
        - **Sort left —> sorted —> [3]; Sort right —> sorted —> [1]**
        - **Combine (Merge) —> [1,3]**
      - **Combine (Merge) —> [1,3,8]**

Divide and Conquer: Merge Sort

- `Merge(A, B):`
  - `Base case:`
    - `if not A: return B`
    - `if not B: return A`
  - `Induction step:`
    - `If A[0] <= B[0]: return [A[0]] + Merge(A[1:], B)`
    - `return [B[0]] + Merge(A, B[1:])`

- **Problem:** sort A = [8, 3, 1, 7, 0, 10, 2] using MergeSort
- **Algorithm:**
  - `MergeSort(A):`
    - `Base case: if len(A) <= 1: return A`
    - `Induction Step:`
      - `Mid = len(A)//2;`
      - `LeftLst = MergeSort(A[0:mid]) //LeftLst is sorted`
      - `RightLst = MergeSort(A[mid:]) //RightLst is sorted`
      - `return Merge(LeftLst, RightLst)`

Exercise: Write the imperative (non-recursive) version of Merge and MergeSort

Other Examples of Divide-Conquer

- **Binary Search:** Finding a value in a sorted array
- **Maximum Subarray Sum:** Contiguous subarray with max sum
- **Power Function** (x^n): Recursive doubling
- **Closest Pair of Points:** Closest pair in the 2D space
- **Count Inversions in an Array**
- **...**

Count Inversions

Brute Force

- Given a list (or array), A, find the number of pairs (i, j) where:
  - $i < j$, and
  - A[i] > A[j]
- **Brute Force:**
  - **Check all pairs**


```
def cntInversions(A):
    n = len(A)
    for i in range(n):
        for j in range(i + 1, n):
            if A[i] > A[j]:
                count += 1
```

Count Inversions — Divide & Conquer

- Given a list (or array), A, find the number of pairs (i, j) where:
  - $i < j$, and
  - A[i] > A[j]
- **Algorithm: A = [2, 4, 1, 3, 5]**
  - Split A in 2 halves; Count recursively the inversions in the left half and right half and sum them up
    - L = [2,4], R = [1,3,5]
  - Count the inversions across Left half and Right half (**basically, Merge**)
    - 2 > 1 —> everything after 2 is also greater than 1 => I discovered **len(L) - i** many inversions in one shot!
  - **Observation: Similar to Merge Sort** —> Why not sort the array and count the inversions!

Count Inversions — Divide & Conquer

- Code: