COP 3502 9/29/25

2 WEEK: Algorithm Analysis

Build tools that will help us analyze the efficiency of algorithms.

Sorted list matching problem

Input: 2 sorted lists

Output: ~~Merge them~~ Output all items that appear in both lists.

list1: 2, 12, 13, 20, 27 — n items

list2: 3, 8, 9, 12, 16, 18, 19, 20, 23, 24, 25 — m items

SOL #1

Nested loop structure:

```
for (int i = 0; i < n; i++) {
    int found = 0;                       // 1 stmt
    for (int j = 0; j < m; j++)          // m stmts
        if (list1[i] == list2[j])
            found = 1;
    if (found) printf("%d ", list1[i]);  // 1 stmt
}
```

m # times when i = 0.

$O(m)$, repeated n times $\Longrightarrow O(nm)$

Sol #2

```
for (int i = 0; i<n; i++)
    if (binsearch(list2, 0, m-1, list1[i]))
        printf("%d ", list1[i]);
```

Runs binary search for list1[i] in list2.

	low	high	diff
1	0	m-1	m
2			m/2
3			m/4
			⋮
			$m/2^k$

$m/2^k \rightarrow$ in k steps search space is $\frac{m}{2^k}$.

$$\frac{m}{2^k} = 1$$

$$m = 2^k$$

$$k = \log_2 m = O(\lg m)$$

this takes $O(n \lg m)$ time.

$n = 10^6, m = 10 \rightarrow O(n \lg m) \sim 10^6 \times 3$ (3mill)

$n = 10, m = 10^6 \rightarrow O(n \lg m) \sim 10 \times 20$ (200)

SOL # 3

list1: 2, 12, 13, 20, 27

list2: 3, 8, 9, 12, 16, 18, 19, 20, 23, 24, 25   12

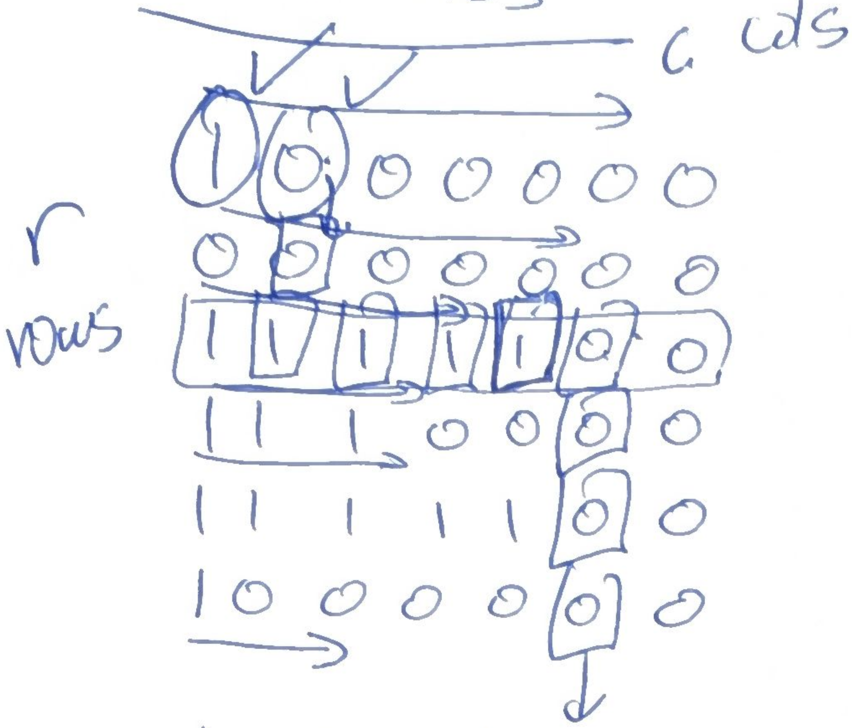
i ~~0~~ ~~1~~ ~~2~~ ~~3~~ 4

j ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~ 11

```
i=0, j=0
while (i<n && j<m) {
    if (list1[i] < list2[j]) i++;
    else if (list2[j] < list1[i]) j++;
    else {
        printf("%d", list1[i]);
        i++; j++;
    }
}
```

$O(n+m)$

Max Ones



Alg #1

for each row → $O(r)$
- Start on left end
- loop until a 0
- If this #1s > best
  - update best

→ $O(c)$

total time is $O(rc)$

Key observation is that we only change our answer is a new row has even more 1s. So don't go back to the beginning to check

```
int best = 0;
int i = 0, j = 0;
while (i < r) {
    while (j<c && arr[i][j] == 1) j++;
    if (j > best) best = j;
    i++;
}
```

rows

// j = 0; ADDING THIS Δ runtime $O(rc)$.

$O(r+c)$

Big-Oh Notation

We say that $f(n) = O(g(n))$
$\in O(g(n))$ if there

exists some constant c such that

$f(n) \le c\,g(n)$ for all $n > n_0$, n_0 is

also a constant.

Intuitively, we don't care about constants ~~any~~ and we only care about ~~that~~ the largest term in a sum of functions.

$$f(n) = 3n^2 + 7n - \log_2 n = O(n^2)$$

For any polynomial, to get its big-oh find the largest exponent term, drop constant

$$f(n) = 12n^5 - 8n^4 + 20000n^3 = O(n^5)$$

Technically $f(n) = 3n^2 = O(n^{10000})$, but this is NOT a tight upper bound.

for logs base doesn't matter.

$O(\lg n)$ $\qquad \log_a n = O(\log_b n)$, $a, b > 1$.

$$\log_a n = \frac{\log_b n}{\log_b a}, \text{ log base change rule}$$
$$= (\text{CONST}) \cdot \log_b n.$$

$O(\log(n^k))$, where k is positive constant grows slower than $O(n^{k2})$, where $k2$ is a positive const

$$f(n) = n^{0.001} + \log(n^{100}) = O(n^{0.001})$$

Any exponential function grows faster than any polynomial function. $O(n^{10000})$ is smaller than $O(2^n)$

$$f(n) = 100 \cdot n^{10000} + \frac{1}{3} \times 2^n = O(2^n)$$

Chart of Ordering

$O(1)$ const

$O(\lg(\lg n))$

$O(\lg n)$

$O(\lg^2 n) \rightarrow (\lg n) \times (\lg n)$ Diff than $O(\lg n^2)$

$O(n^k)$, $k \in$ Const

$O(2^n)$

$O(3^n)$

$O(4^n)$

$O(n^n) \rightarrow O(n!)$

$O(\lg n^2) = O(\underline{\underline{2}} \lg n)$
$= O(\lg n)$

$\sqrt{n} = n^{\frac{1}{2}}$

Poly

$O(\sqrt{n})$

$O(n) \rightarrow O(n \lg n)$

$O(n\sqrt{n})$

etc.