Looking Ahead

- Recursion
- Analysis of algorithms/Complexity
- Abstract datatypes

Recursion

- **Ex:** (i) $a+b = \begin{cases} a, & \text{if } b=0 \\ (a+1)+(b-1), & \text{otherwise} \end{cases}$

"Reducing '+' to '+1' and '-1'."

(ii) $a*b = \begin{cases} 0, & \text{if } b=0 \\ a+a*(b-1), & \text{otherwise} \end{cases}$

"Reducing '*' to '+'."

$$\begin{aligned} 4*2 &= 4+4*1 \\ &= 4+4+4*0 \\ &= 4+4+0 \\ &= \underline{8} \end{aligned}$$

(iii) $a^n = \begin{cases} 1, & \text{if } n=0 \\ a*a^{n-1}, & \text{otherwise} \end{cases}$

(iv) $n! = \begin{cases} 1 & , \text{ if } n=0 \\ n*(n-1)! & , \text{ otherwise} \end{cases}$

$$\begin{aligned} 3! &= 3 * 2! \\ &= 3 * 2 * 1! \\ &= 3 * 2 * 1 * 0! \\ &= 3 * 2 * 1 \times 1 \\ &= \underline{6} \end{aligned}$$

"Reducing '!' to '*'."

• C function:

```
static int Fact (int n)
{ if (n==0)
        return (1);
    else
        return (n* Fact(n-1));
}
```

Fact(3)
→ return(3*Fact(2))
6
→ return(2*Fact(1))
2
→ return(1*Fact(0))
1
→ return(1)

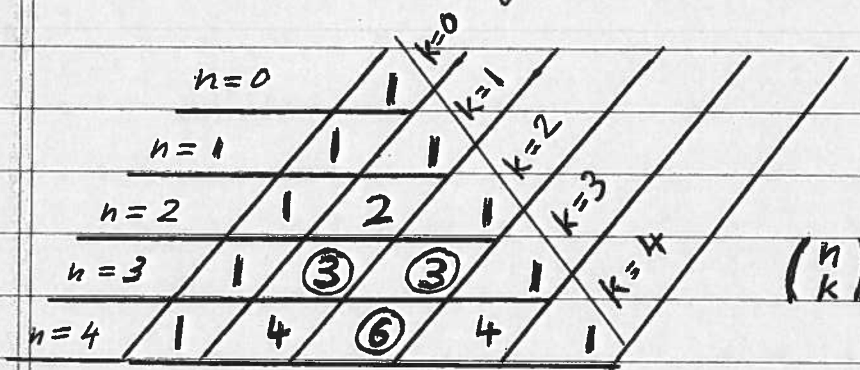
- **"STACK":**

Fact(0): n = 0 → 1	
Fact(1): n = 1 → 1 * Fact(0)	1
Fact(2): n = 2 → 2 * Fact(1)	1
Fact(3): n = 3 → 3 * Fact(2)	2
	6

(V) General recursion:

```
if (simple case(s))
    return (solution for simple case(s));
else
    return (recursive solution involving
            call(s) of the same function);
```

(VI) Pascal's triangle / Binomial coefficients



$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

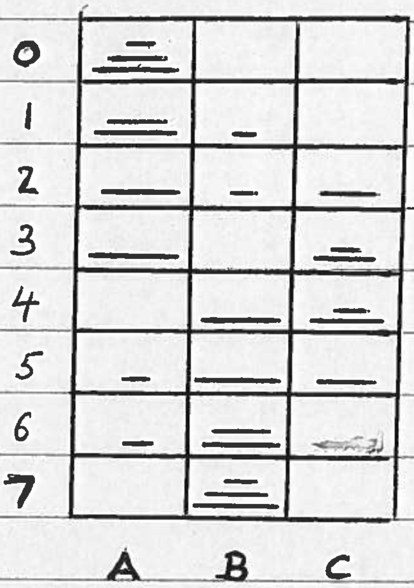
- Pascal's: $\underbrace{\binom{n-1}{k-1} \oplus \binom{n-1}{k}}_{\binom{n}{k}}$

$$\Rightarrow \binom{n}{k} = \begin{cases} 1 & \text{, if } k=0 \text{ or } n=k \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{, otherwise} \end{cases}$$

• **Note:** $(x+y)^3 = \underline{1}x^3 + \underline{3}x^2y + \underline{3}xy^2 + \underline{1}y^3$

$\Rightarrow (1, 3, 3, 1) =$ row $\underline{n=3}$ in Pascal's triangle

➡ Recursive function for "Towers of Hanoi":



• "Move tower from A to B.
Move one disk at a time.
A larger disk must not be on top of a smaller disk."

• n disks $\Rightarrow 2^n - 1$ moves.

• $n = 25$; 1 move: 1 sec
$\Rightarrow$ 1 year to move tower

```
if (tower of 1 disk)                    /* n: number of disks */
    • move tower from A to B;
else
{   • move tower of (n-1) top disks to C;
    • move Largest/lowest disk n to B;
    • move tower of (n-1) disks from C to B;
}
```

Analyses of algorithms

> "Study of computational complexity/ efficiency of algorithms depending on problem/data size"

$\mathcal{O}(1)$ – constant time

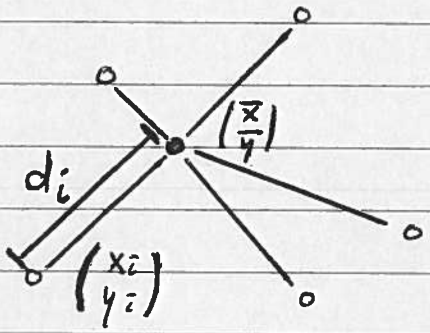
$\mathcal{O}(n)$ – linear time

→ doubling data → double time

$\mathcal{O}(n^2)$ – quadratic time

(n = no. of data) → doubling data → 4 * time

• **Ex:** • "Finding closest point"



– Given $\binom{\bar{x}}{\bar{y}}$ and points $\binom{x_i}{y_i}$, $i = 1 \ldots n$, which point is closest to $\binom{\bar{x}}{\bar{y}}$?

<u>$\mathcal{O}(n)$</u>

– Compute n squared distance values $d_i^2 = (\bar{x} - x_i)^2 + (\bar{y} - y_i)^2$ to determine closest point

• "Finding point pair $\binom{x_i}{y_i}$, $\binom{x_j}{y_j}$ with minimal distance"

<u>$\mathcal{O}(n^2)$</u>

– Compute values $d_{i,j}^2 = (x_i - x_j)^2 + (y_i - y_j)^2$, $i, j = 1 \ldots n$.

- "Selection sort – counting no. of comparisons"

$(n-1)$ {

input: | 4 | 3 | 2 | 1 | — 3 comparisons

| 1 | 3 | 2 | 4 | — 2 "

| 1 | 2 | 3 | 4 | — 1 "

} 2 comparisons on average $(2 = \frac{n}{2})$

output: | 1 | 2 | 3 | 4 | ✓ — (0) ".

$n = 4$

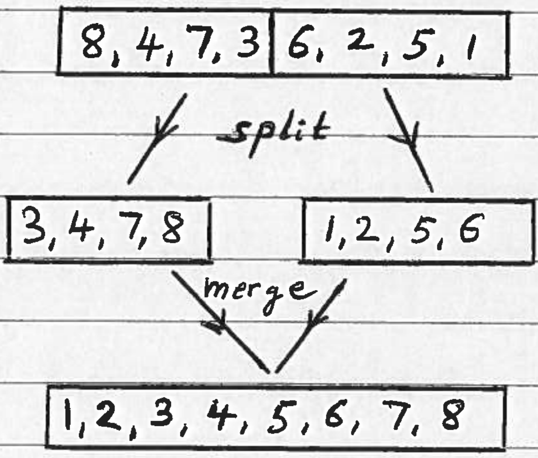
$\Rightarrow (n-1) * (\frac{1}{2} * n) = \frac{1}{2}(n^2 - n)$ comparisons

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

! ■ **DIVIDE-AND-CONQUER** !!!

- **Idea:** "Divide a big problem into two sub-problems (of the same type) of 'nearly equal size' and solve the two sub-problems independently; then 'merge' the results of the sub-problems to determine the result of the big problem."

• **Ex:** "Merge-sort" – Sorting an array of numbers

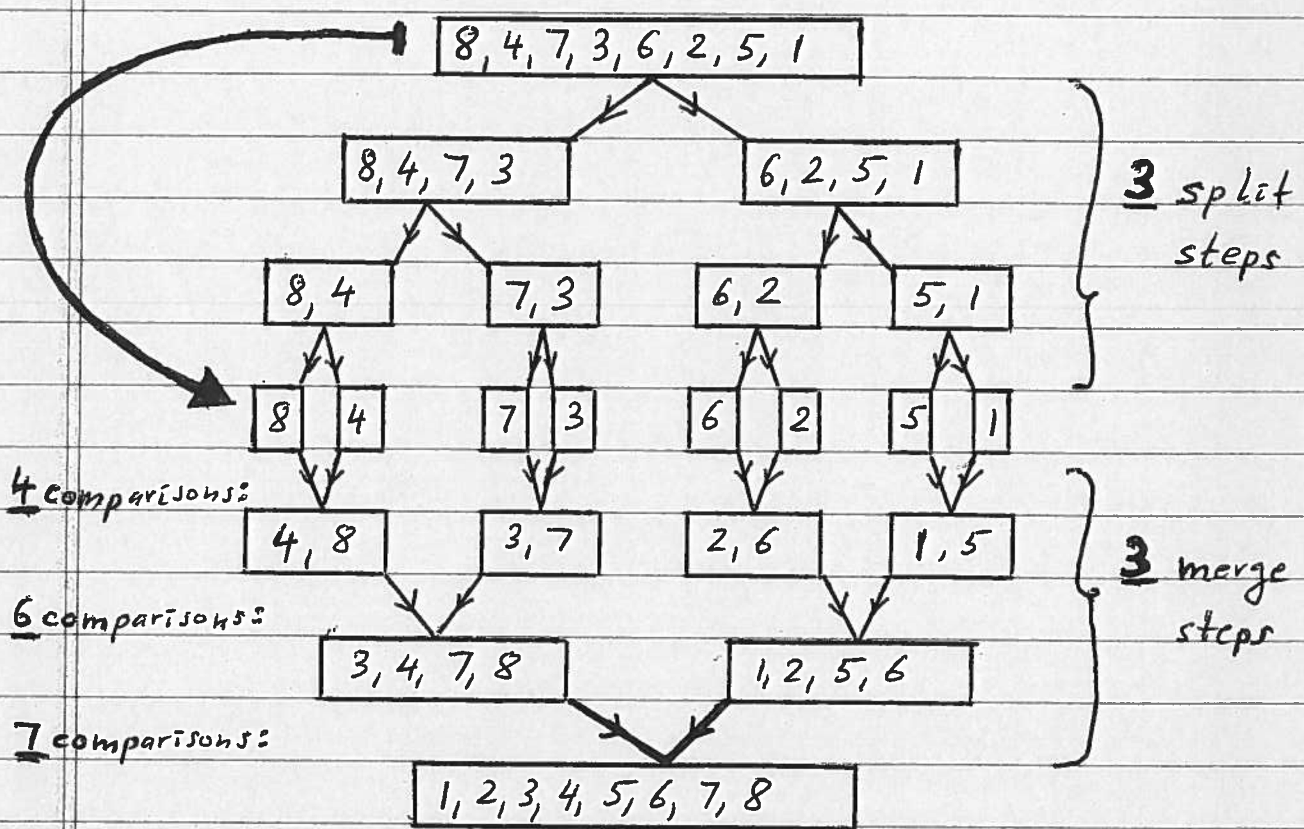


(i) divide given array into two sub-arrays (of equal size);

(ii) sort the two sub-arrays;

(iii) merge the sub-arrays;

→ Apply principle recursively: split sub-arrays until array size of one is reached, then merge!



• here: $\underline{n=8}$ → $\underline{\log_2 8} = \underline{3}$ split steps,
$\underline{3}$ merge "

➡ $\leq n$ $(=8)$ comparisons needed during each merge step

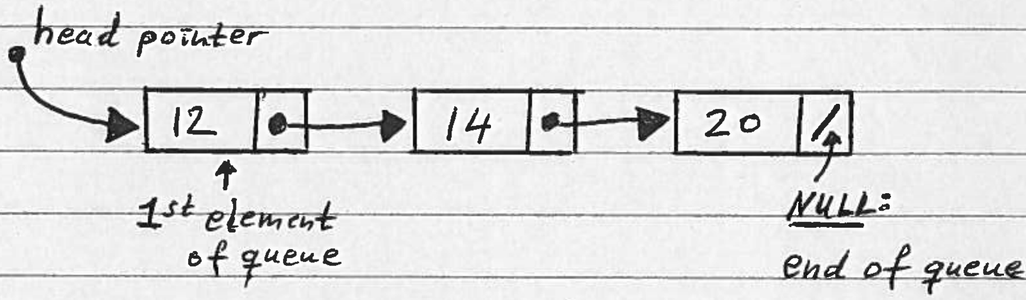
➡ No. of total comparisons $\leq n \cdot \log_2 n$

$O(n \log n)$

➡ Abstract data types

- **Idea:** "Defining one's own data structures/types and operations/functions for them"
- **Ex:** "**QUEUE**" – defined via a pointer-based abstraction (and implementation)

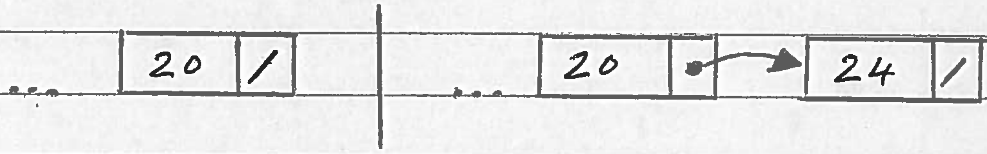
e.g., a queue of integers:



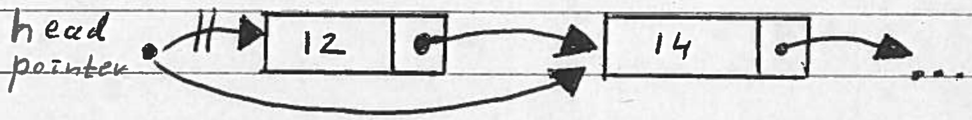
"structure"

→ desired algorithms for a queue:

- create queue: • → head pointer
- eliminate queue: ...
- add element at end of queue ("enqueue"):



- remove element from beginning ("dequeue"):

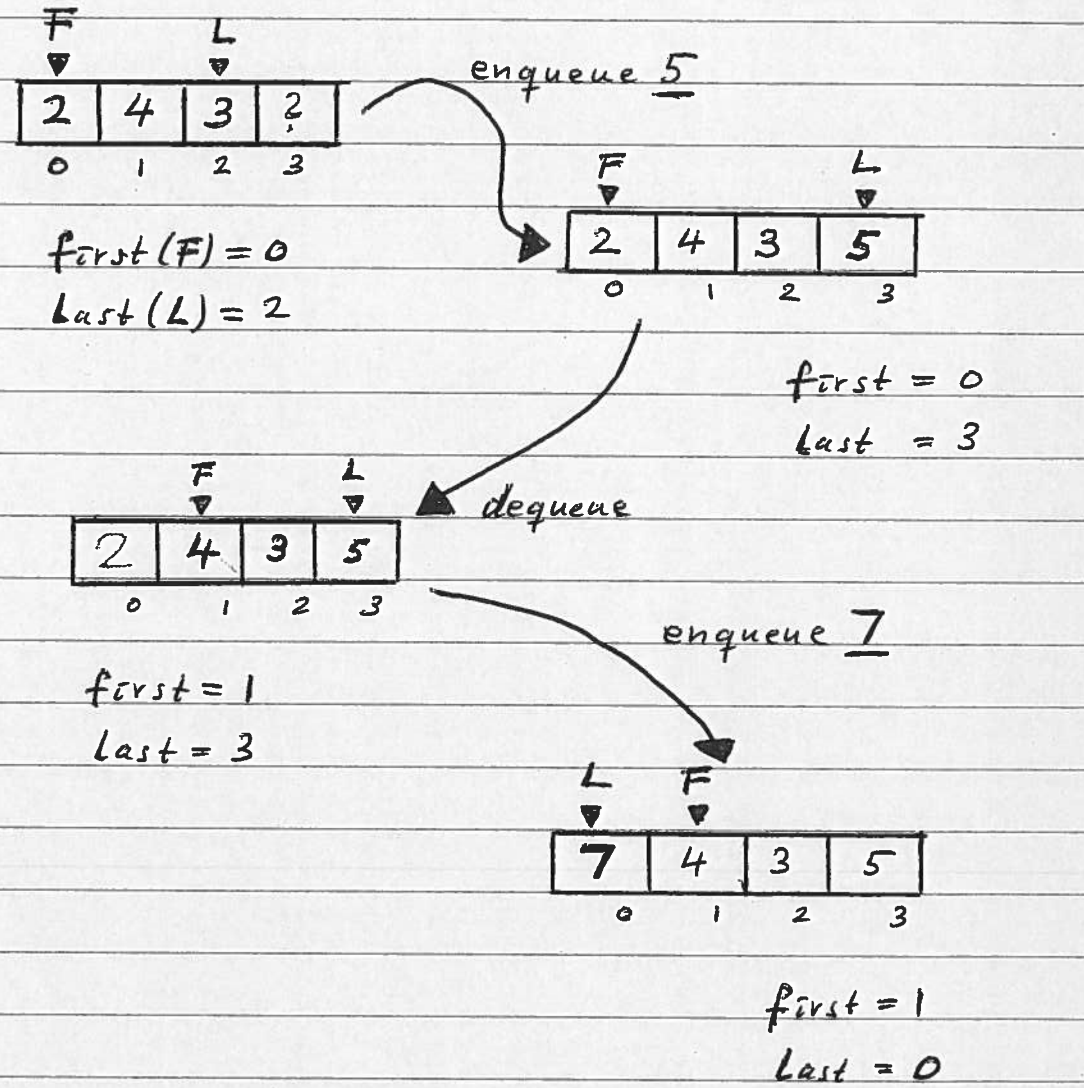


- count no. of queue elements: ...

• **Note:** - Queues (of processes) are important for the design of operating systems!

- Can design and implement your own "Queue Library" (queue.h, queue.c), see Chapter 17.

e.g., array-based implementation of a queue of integers, with maximal queue size/length being 4:



- array indices used in modulo fashion
- using first/last as 'head'/'tail' indices: no need to shift actual array elements!