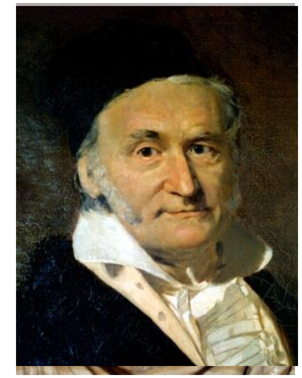
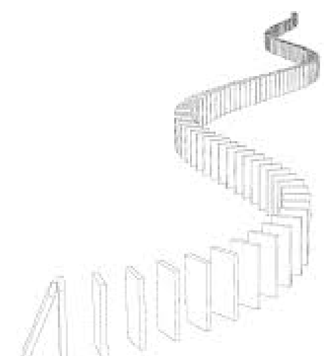


COMP 1002

Logic for Computer Scientists

Lecture 30



Tower of Hanoi game

- Rules of the game:
  - Start with all disks on the first peg.
  - At any step, can move a disk to another peg, as long as it is not placed on top of a smaller disk.
  - Goal: move the whole tower onto the second peg.
- Question: how many steps are needed to move the tower of 8 disks? How about n disks?

Recurrence relations

- **Recurrence**: an equation that defines an n^{th} element in a sequence in terms of one or more of previous terms.
    - Think of $F(n) = s_n$ for some sequence $\{s_n\}$
  - $H(n) = 2H(n-1) + 1$
  - $F(n) = F(n-1) + F(n-2)$

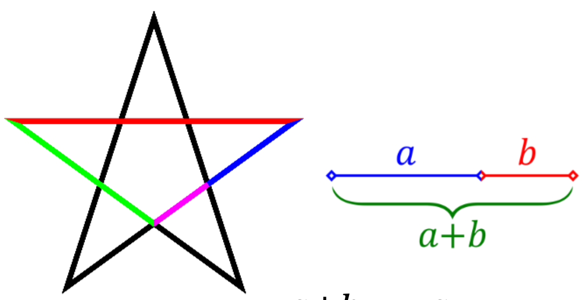
- A **closed form** of a recurrence relation is an expression that defines an n^{th} element in a sequence in terms of n directly.
  - Often use recurrence relations and their closed forms to describe performance of (especially recursive) algorithms.

Closed forms of some sequences

- Arithmetic progression:
  - Sequence: $c, c+d, c+2d, c+3d, \ldots, c+nd, \ldots$
  - **Closed form:** $s_n = c + nd$
    - Closed forms are very useful for analysis of recursive programs, etc.

- Geometric progression:
  - Sequence: $c, cr, cr^2, cr^3, \ldots, cr^n, \ldots$
  - **Closed form:** $s_n = c \cdot r^n$

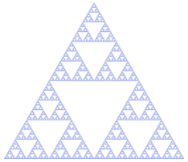
- Fibonacci sequence: F(n)=F(n-1)+F(n-2)
  - Sequence: 1,1,2,3,5,8,13, …
  - **Closed form:** $F_n = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}}$
    - Where φ *("phi")* is the "golden ratio": a ratio such that $\frac{a+b}{a} = \frac{a}{b}$
    - $\varphi = \frac{1+\sqrt{5}}{2}$



Tower of Hanoi game

- Rules of the game:
  - Start with all disks on the first peg.
  - At any step, can move a disk to another peg, as long as it is not placed on top of a smaller disk.
  - Goal: move the whole tower onto the second peg.
- Question: how many steps are needed to move the tower of 8 disks? How about n disks?

- Let us call the number of moves needed to transfer n disks H(n).
  - Names of pegs do not matter: from any peg i to any peg $j \neq i$ would take the same number of steps.
- Basis: only one disk can be transferred in one step.
  - So H(1) = 1
- Recursive step:
  - suppose we have n-1 disks. To transfer them all to peg 2, need $H(n-1)$ number of steps.
  - To transfer the remaining disk to peg 3, 1 step.
  - To transfer n-1 disks from peg 2 to peg 3 need H(n-1) steps again.
  - So H(n) = 2H(n-1)+1 (recurrence).
- Closed form: H(n) = $2^n - 1$.

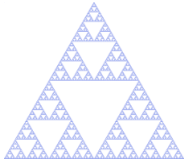


Closed form for Tower of Hanoi

- Solving a recurrence: finding a closed form.
  - Solving the recurrence H(n)=2H(n-1)+1
    - H(n) $= 2 \cdot H(n-1) + 1$

$$= 2(2H(n-2)+1)+1 = 2^2 H(n-2) + 2 + 1$$
$$= 2^3 H(n-3) + 2^2 + 2 + 1$$
$$= 2^4\, H(n-4) + 2^3 + 2^2 + 2 + 1 \ldots$$

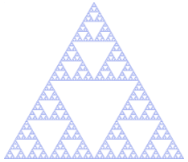
  - Closed form: $H(n) = \sum_{i=0}^{n-1} 2^i = 2^n - 1$
    - Proof by induction
    - Or by noticing that a binary number 111...1 plus 1 gives a binary number 10000...0



Solving recurrences

- So adding one more disk doubles the number of steps.
  - We say that the function defined by H(n) **grows exponentially**
  - $H(n) \in O(2^n)$ (and nothing slower-growing).
    - To say "nothing slower-growing", use symbol Ω (uppercase omega): $H(n) \in \Omega(2^n)$
    - To say "grows exactly like 2^n, use symbol Θ (uppercase theta): $H(n) \in \Omega(2^n)$

- Solving recurrences in general might be tricky.
  - When the recurrence is of the form T(n)=a T(n/b)+f(n), there is a general method to estimate the growth rate of a function defined by the recurrence
    - Called the Master Theorem for recurrences.



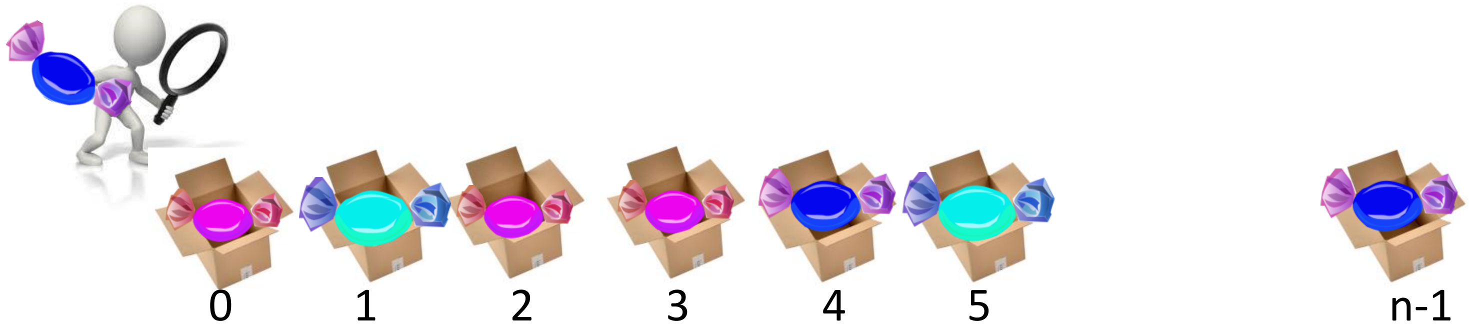
Master theorem for solving recurrences

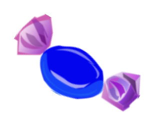
- Let $a, b, c, d \in \mathbb{R}$ such that $a \geq 1,\ b \geq 2, c > 0$ $,\ d\ \geq 0,$ and let f(n) $\in \Theta(n^c)$
- Let T(n) be the following recurrence relation:
  - Base: T(1) = d
  - Recurrence: $T(n) = a\,T\left(\left\lceil \frac{n}{b} \right\rceil\right) + f(n)$
- Then the growth rate of T(n) is:
  - If $\log_b a < c$ then T$(n) \in \Theta\big(f(n)\big)$
  - If $\log_b a = c$ then T$(n) \in \Theta(f(n)\log n\ )$
  - If $\log_b a > c$ then T$(n) \in \Theta\big(n^{\log_b a}\ \big)$

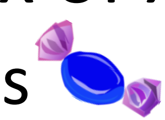
Analysis of algorithms

- Putting it all together:
  - Using **logic** to describe what an algorithm is doing
  - and **induction** to show that it does that correctly
  - Using **recurrence** relations to see how long it takes in the worst case.
    - With **O-notation** to talk about the time.
  - and **probabilities/expectation** to try to see how long it might take on average.

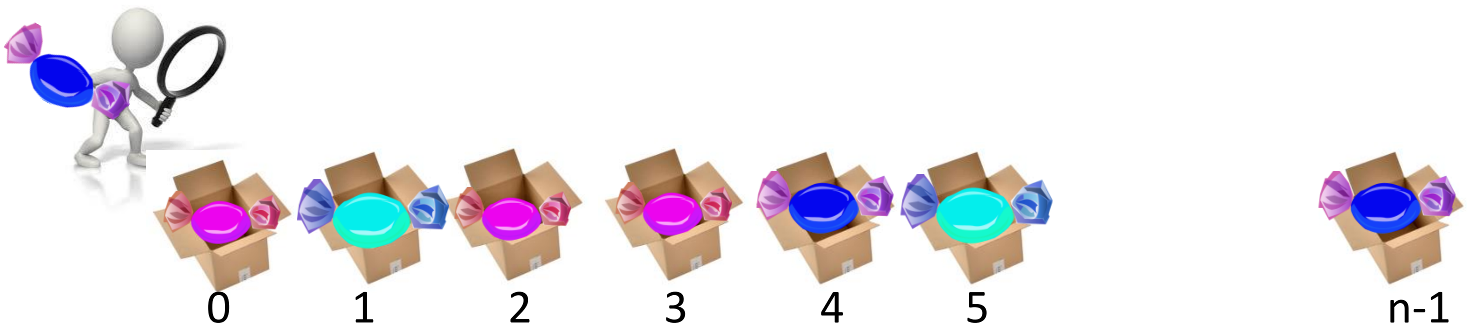
Example: search in an array

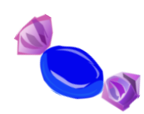


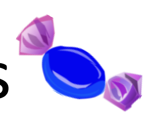
- Given:
  - an array A containing n elements, 
  - and a specific item x 

- Goal: find the index of x in A, if x is in A.
  - Which box contains  ? Box 4. 

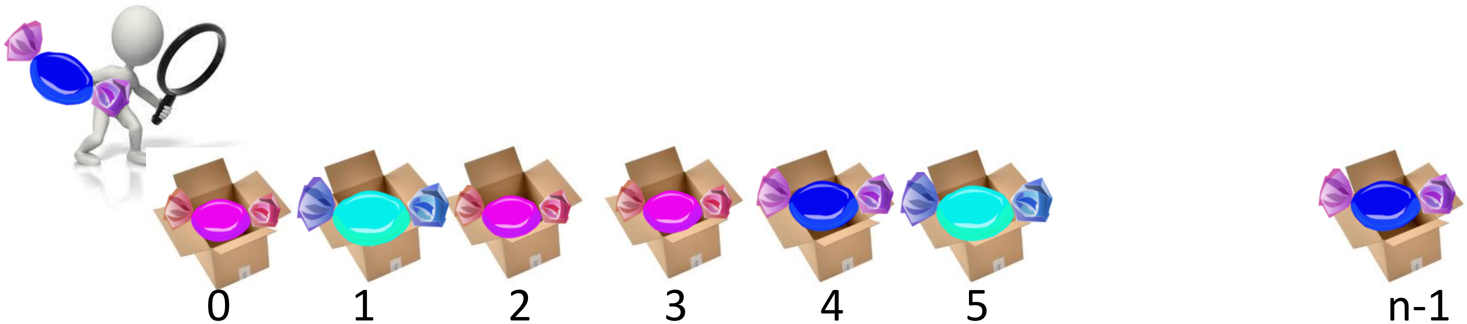
Example: search in an array



- Given:
  - an array A containing n elements, 
  - and a specific item x 

- Goal: find the index of x in A, if x is in A.
  - Which box contains ? Box 4. 

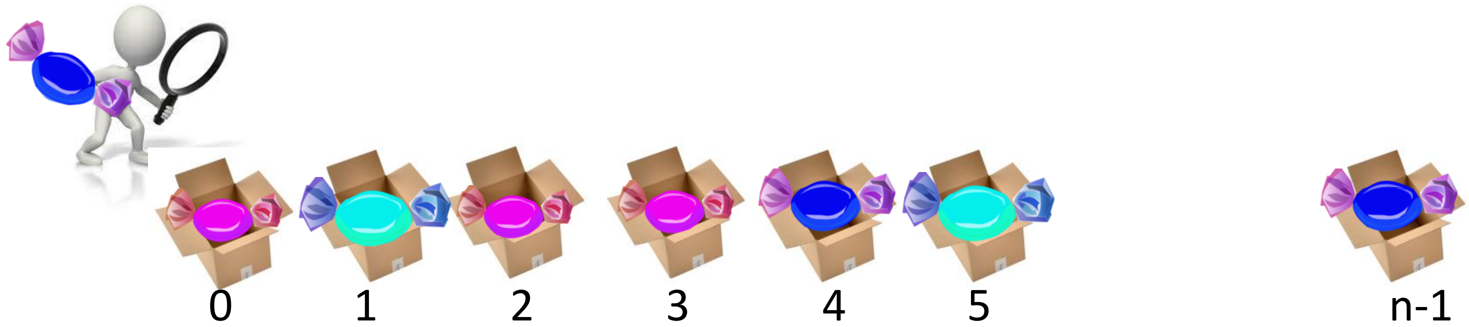
Example: search in an array



- *Precondition*: what should be true before a piece of code (or the whole algorithm) starts
  - E.g.: A is an array of numbers and A is not empty and x is a number.

- *Postcondition*: what should be true after a program (piece of code) finished.
  - E.g. If the program returned value k, then A[k]=x
    - or k=-1, if x is not in A.

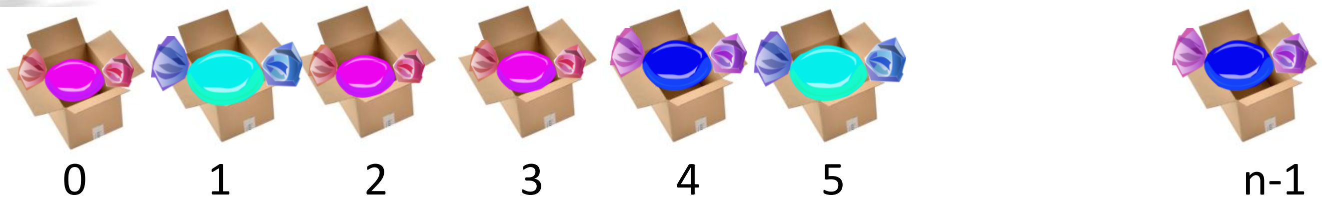
Example: search in an array



- *Precondition*: A is an array containing x

- *Postcondition*: Returned k such that A[k]=x

Example: search in an array



- *Precondition*: A is an array containing x

```
Algorithm arraySearch(A, x)
Input array A of n integers, number x
Output k such that A[k]=x

i = 0
out = -1

while  out < 0 do
        if A[i]  =  x  then
                out  = i
        i = i+1

return out
```

- *Postcondition*: Returned k such that A[k]=x

arraySearch algorithm

Algorithm ***arraySearch*(*A*, *x*)**
Input array ***A*** of ***n*** integers, number x
Output k such that A[k]=x

$\exists i \in \{0 \ldots n-1\}\ A[i] = x$

i = 0
out = -1

$\exists i \in \{0 \ldots n-1\}\ A[i] = x\ \wedge i = 0 \wedge out = -1$

while ***out*** < ***0*** **do**
  if *A*[*i*] = x **then**
    out = ***i***
  i = ***i*+1**

$A[out] = x$

return ***out***

Program returned k such that A[k]=x

- *A = [5,10,8,7]*
- *x = 8*
- *out = 2*

Loop invariant

- **Loop invariant:** a condition that is true on each iteration of the loop
  - Implied by loop precondition
  - Implies the loop postcondition
  - Implies next loop iteration is correct

- I(k): $i = k \ \wedge ((out = i \ \wedge A[out] = x) \vee (\exists j > i \ \ A[j] = x))$

- Guard condition: condition in the while loop
  - G= “out <0”

- Loop is correct when:
  - precondition $\rightarrow$ I(0)
  - for all k, $\mathrm{G} \wedge \mathrm{I(k)} \rightarrow \mathrm{I(k+1)}$
  - If $\mathrm{k_0}$ is the smallest number such that $\neg G$, then $\neg G \wedge I(k_0) \rightarrow$ postcondition

- **Termination**: proof that $\exists\ \mathrm{k_0}$ such that after $\mathrm{k_0}$ iterations G becomes false

$\exists i \in \{0 \ldots n-1\} \ \ A[i] = x \ \wedge$
$\wedge \ \ i = 0 \wedge out = -1$

while *out < 0* **do**
    if $A[i] = \mathrm{x}$ **then**
        out = *i*
    i = *i*+1

$A[out] = x$

Proving the loop invariant

- By induction on i:
- Base case: I(0)
  - $\exists i \in \{0 \dots n-1\}\ A[i] = x \land i = 0 \land$
    $\land\, out = -1$
    Implies I(0)
  - $i = 0\ \land ((out = 0\ \land A[out] = x) \lor (\exists j > i\ \ A[j] = x))$
- Assume I(k): $i = k\ \land ((out = i\ \land A[out] = x) \lor (\exists j > i\ \ A[j] = x))$
- Show: if G, then I(k+1): $i = k + 1\ \land ((out = i\ \land A[out] = x) \lor (\exists j > i\ \ A[j] = x))$
  - i=k+1 because of "i=i+1" statement
  - If A[i]=x, then $(out = i\ \land A[out] = x)$ holds
  - Otherwise, $(\exists j > i\ \ A[j] = x)$ holds.
- Otherwise, if $\neg G$, postcondition holds:
  - in this case, $(out = i\ \land A[out] = x)$ should have been true in I(k), for i=k.
  - So A[out]=x

$\exists i \in \{0 \dots n-1\}\ A[i] = x\ \land$
$\land\ i = 0 \land out = -1$

while *out* < *0* **do**
  if $A[i]$ = x **then**
    out = *i*
  i = *i+1*

$A[out] = x$