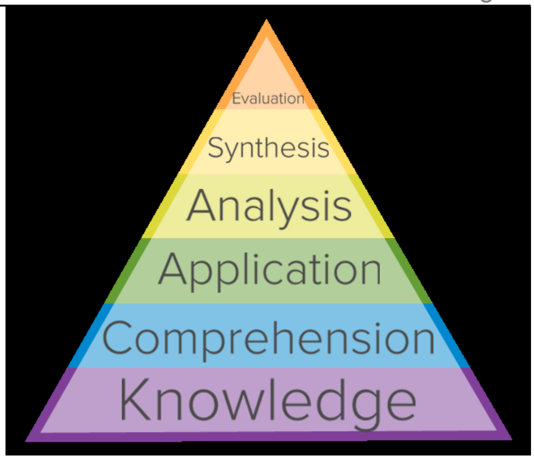
§1 Introduction

Bloom's Hierarchy
of cognitive learning



- *What is thought is not said*
- *What is said is not heard*
- *What is heard is not understood*
- *What is understood is not believed*
- *What is believed is not yet advocated*
- *What is advocated is not yet acted on*
- *What is acted on is not yet completed*

Konrad Lorenz (Nobel Prize 1973)

Syllabus

- "Virtues" of Computer *Science*
- Algorithms vs. Heuristic, Program
- Asymptotic Efficiency
- Example: Powering
- Example: Fibonacci Numbers
- Example: Polynomial Multiplication

Computer *Science*

CS500 M. Ziegler

"***Virtues***":

- problem specification
- formal semantics
- algorithm design
- and analysis (correctness, efficiency)
- proof of optimality (complexity, CS422)

≠program
≠heuristic

Algorithm ≠ Heuristic, Program

CS500 M. Ziegler

An algorithm is a

- finite sequence of
- primitive instructions that,

executed according to their

- well-specified semantics,

provide a *mechanical* solution to the *infinitely* many instances of a possibly *complex* mathematical problem.

1. fully specified (input/output)
2. guaranteed correct (no *heuristic/recipe*)
3. analysis of cost (runtime, memory, …)
4. optimality proof (wrt a model of computation)

Algorithm ≠ Heuristic, Program


```
        mov esi, offset list
top:    mov edi, esi
inner:  mov eax, [edi]
        mov edx, [edi+4]
        cmp eax, edx
        jle no_swap
        mov [edi+4], eax
        mov [edi], edx
no_swap: add edi, 4
        cmp edi, list_end - 4
        jb inner
        add esi, 4
        cmp esi, list_end - 4
        jb top
```

- primitive operations
- their semantics
- their costs

```
# vowels list
vowels = ['e', 'a', 'u', 'o', 'i']
# sort the vowels
vowels.sort()
# print vowels
print('Sorted list:', vowels)
```

Asymptotic Efficiency

n	$\log_2 n \cdot$10s	$n \cdot \log n$ sec	n^2 msec	n^3 μsec	2^n nsec
10	33sec	33sec	0.1sec	1msec	1msec
100	≈1min	11min	10sec	1sec	40 Mrd. Y
1000	≈1.5min	≈3h	17min	17min	
10 000	≈2min	1.5 days	≈1 day	11 days	
100 000	≈2.5min	19 days	4 months	32 years	

- Running times of some sorting algorithms
  - **BubbleSort**: $O(n^2)$ comparisons and copy instructions
  - **QuickSort**: typically $O(n \cdot \log n)$ steps
    but $O(n^2)$ in the worst-case
  - **HeapSort**: always at most $O(n \cdot \log n)$ operations
  - **BucketSort**: $O(n)$ operations
  - SORT primitive: $O(1)$
- **Worst-case** vs. **average**-case vs. **best** case
  w.r.t. input size =: $n \to \infty$

Example: Powering

Optimality?

Powering Problem: Given X and $n \in \mathbb{N}$.

Compute X^n with few(est number of) multiplications

- $X^n = X \cdot X \cdot \ldots \cdot X$: n-1 multiplications
- Let $k := \lfloor n/2 \rfloor$ and recursively compute X^k,
  then compute $X^n = (X^k)^2$ or $X^n = (X^k)^2 \cdot X$
- #multiplications $T(n) \leq T(n/2) + 2$, $T(n) \leq 2 \cdot \log_2(n)$
- Asympt. optimality: Each multiplication at most doubles the degree of the intermediate results;
  so computing x^n requires at least $\log_2 n$ of them.

Example: Fibonacci Numbers


```
FibIter(n)
if n=0 return 0;
fib := 1;  fibL := 0;
while n>1 do
    tmp:=fibL;
    fibL := fib;
    fib := fibL⊕tmp;
    n := n-1  ;  end
return fib;
```

$$F_0=0, \quad F_1=1, \quad F_n = F_{n-1} + F_{n-2}$$

```
FibRek(n)
if n=0 return 0;  if n=1 return 0;
return FibRek(n-1)⊕FibRek(n-2);
```

$$F_n = (\varphi^n - (-1/\varphi)^n) \,/\, \sqrt{5}$$

$$\varphi := (1+\sqrt{5})/2$$

$$\begin{vmatrix} F_n \\ F_{n-1} \end{vmatrix} = \begin{vmatrix} 1 & 1 \\ 1 & 0 \end{vmatrix} \cdot \begin{vmatrix} F_{n-1} \\ F_{n-2} \end{vmatrix} = \begin{vmatrix} 1 & 1 \\ 1 & 0 \end{vmatrix}^k \cdot \begin{vmatrix} F_{n-k} \\ F_{n-k-1} \end{vmatrix} \qquad k := n-1$$

Example: Polynomial Multiplication

Long Multiplication

Input: coefficients of polynomials

$A(x)=a_0+a_1x+a_2x^2+\ldots+a_{N-1}x^{N-1}$ and $B(x)$ of degree $<N$.

Output: coefficients of polynomial

w.l.o.g. $2|N$

$C(x) := A(x) \cdot B(x)$ of degree $\leq K:=2N-2$.

e.g. $\times(-5)$ or $+$

$T(N)$:= #arithmetic operations (multiplications, linear combinat.s)

Recursive algorithm, Distributive law: $T(N) = 4\cdot T(N/2) + O(N)$

"Naîve" $c_k = \sum_j a_j \cdot b_{k-j}$

$$(A_0(x)+A_1(x)\cdot x^{N/2}) \cdot (B_0(x)+B_1(x)\cdot x^{N/2}) = C_0(x) + C_1(x)\cdot x^{N/2} + C_2(x)\cdot x^N$$

$C_0(x)=A_0(x)\cdot B_0(x)$ $C_1(x)=A_0(x)\cdot B_1(x)+A_1(x)\cdot B_0(x)$ $C_2(x)=A_1(x)\cdot B_1(x)$

$C(x)=$ | $c_0 ,\ldots, c_{N/2-1}$ | $c_{N/2} ,\ldots, c_{N-1}$ | $c_{N-1} ,\ldots, c_{3/2\cdot N-1}$ | $c_{3/2\cdot N-1} ,\ldots, c_{2N-2}$ |

Example: Polynomial Multiplication

Karatsuba

Input: coefficients of polynomials

$A(x)=a_0+a_1x+a_2x^2+\ldots+a_{N-1}x^{N-1}$ and $B(x)$ of degree $<N$.

Output: coefficients of polynomial

w.l.o.g. $2|N$

$C(x) := A(x) \cdot B(x)$ of degree $\leq K:=2N-2$.

e.g. $\times(-5)$ or $+$

$T(N)$:= #arithmetic operations (multiplications, linear combinat.s)

Recursive algorithm, Distributive law: $T(N) = 4\cdot T(N/2) + O(N)$

based on: „Karatsuba law“: $T(N) = 3\cdot T(N/2) + O(N)$

$$(A_0(x)+A_1(x)\cdot x^{N/2}) \cdot (B_0(x)+B_1(x)\cdot x^{N/2}) = C_0(x) + C_1(x)\cdot x^{N/2} + C_2(x)\cdot x^N$$

$C_0(x) := A_0(x)\cdot B_0(x)$, $C_2(x) := A_1(x)\cdot B_1(x)$

$$C_1(x) := (A_0(x)+A_1(x))\cdot(B_0(x)+B_1(x)) - C_0(x) - C_2(x)$$

Example: Polynomial Multiplication

Toom

$$T_1 := (A_0+2A_1+4A_2) \odot (B_0+2B_1+4B_2)$$
$$T_2 := (A_0+A_1+A_2) \odot (B_0+B_1+B_2)$$
$$T_3 := (4A_0+2A_1+A_2) \odot (4B_0+2B_1+B_2)$$

w.l.o.g. $3|N$

$$C_3 = -C_0 + \tfrac{1}{3}T_1 - 2T_2 + \tfrac{1}{6}T_3 - 3\tfrac{1}{2}C_4$$
$$C_2 = 3\tfrac{1}{2}C_0 - \tfrac{1}{2}T_1 + 5T_2 - \tfrac{1}{2}T_3 + 3\tfrac{1}{2}C_4$$
$$C_1 = -3\tfrac{1}{2}C_0 + \tfrac{1}{6}T_1 - 2T_2 + \tfrac{1}{3}T_3 - C_4$$
$$C_0 = A_0 \odot B_0,$$
$$C_4 = A_2 \odot B_2$$

Distributive law: $T(N) = 4\cdot T(N/2) + O(N)$

„Karatsuba law": $T(N) = 3\cdot T(N/2) + O(N)$

„Toom's law": $T(N) = \mathbf{5}\cdot T(N/\mathbf{3}) + O(N)$

$$(A_0(x)+A_1(x)\cdot x^{N/3} + A_2(x)\cdot x^{2N/3}) \times (B_0(x)+B_1(x)\cdot x^{N/3} + B_2(x)\cdot x^{2N/3})$$
$$= C_0(x) + C_1(x)\cdot x^{N/3} + C_2(x)\cdot x^{2N/3} + C_3(x)\cdot x^{3N/3} + C_4(x)\cdot x^{4N/3}$$

Example: Polynomial Multiplication

Toom-Cook

Input: coeff. of $A(x)=a_0+a_1x+a_2x^2+\ldots$ of $\deg<N$, $B(x)$ of $\deg<M$

Output: coeffients $c_0, \ldots c_K$ of $C(x) := A(x)\cdot B(x)$, $\deg(C)\le K := N+M-2$

Long Multiplication: $N\cdot M$ products & linear combinations

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^K \\ 1 & x_1 & x_1^2 & \ldots & x_1^K \\ & & \vdots & & \\ 1 & x_K & x_K^2 & \ldots & x_K^K \end{pmatrix}^{-1} \bullet \begin{pmatrix} A(x_0)\cdot B(x_0) \\ A(x_1)\cdot B(x_1) \\ \vdots \\ A(x_K)\cdot B(x_K) \end{pmatrix} = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_K \end{pmatrix}$$

Vandermonde Matrix invertible for any distinct fixed $x_0, \ldots x_K$

Evaluation/Interpolation: $K+1$ products, $O(N^2+M^2)$ linear comb.s

$$T(N,M) = (n+m-1)\cdot T(N/n, M/m) + O((N+M)\cdot(n+m))$$

$$(A_0(x) + A_1(x)\cdot x^{N/n} + A_2(x)\cdot x^{2N/n} + \ldots + A_{n-1}(x)\cdot x^{(n-1)\cdot N/n})$$
$$\times\ (B_0(x) + B_1(x)\cdot x^{M/m} + B_2(x)\cdot x^{2M/m} + \ldots + B_{m-1}(x)\cdot x^{(m-1)\cdot M/m})$$