Algorithms and Data Structures

CS-CO-412

David Vernon
Professor of Informatics
University of Skövde
Sweden

david@vernon.eu
www.vernon.eu

Course Outline

- Motivation & Preview
  - The importance of algorithms & data structures

- Complexity of algorithms
  - Performance of algorithms
  - Time and space tradeoff
  - Worst case and average case performance
  - Big O notation
  - Recurrence relationships
  - Analysis of complexity of iterative and recursive algorithms
  - Tractable and intractable algorithmic complexity

Course Outline

- Simple searching algorithms
  - Linear search
  - Binary search
- Simple sorting algorithms
  - Bubblesort
  - Quicksort
- Abstract Data Types (ADTs)

Course Outline

- Lists, stacks, and queues
  - ADT specification
  - Array implementation
  - Linked-list implementations

- Trees
  - Binary trees
  - Binary search trees
  - Depth-first traversals
  - Applications of trees (e.g. Huffman coding)
  - Height-balanced trees (e.g. AVL Trees, Red-Black Trees)

Course Outline

- Priority queues
  - Priority queue data structure
  - Binary heap
  - Heapsort

- Graphs
  - Graph data structure
  - Breadth-first search traversal
  - Depth-first search traversal
  - Minimum spanning tree (e.g. Prim's and Kruskal's algorithms)
  - Shortest-path algorithms (e.g. Dijkstra's and Floyd's algorithms)
  - Topological sort

Course Outline

- Algorithmic strategies
  - Brute-force
  - Divide-and-conquer
  - Greedy
  - Dynamic programming
  - Combinatorial Search & Backtracking
    - Combinatorics: subsets and permutations
    - All paths in a graphs
  - Branch-and-bound

Motivation & Preview

The Importance of Algorithms & Data Structures

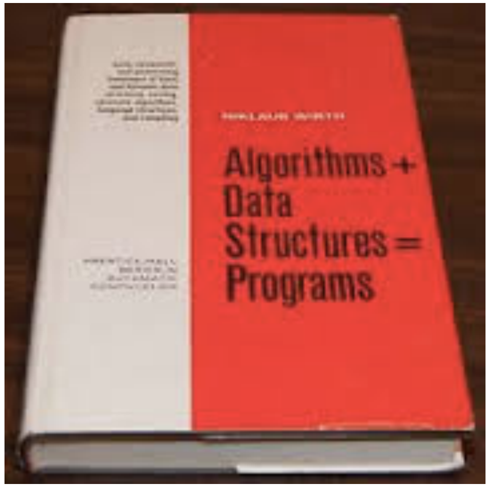
Muḥammad ibn Mūsā al-Khwārizmī

محمد بن موسى الخوارزمي

Born approximately 780, died between 835 and 850
Persian mathematician and astronomer
from the Khorasan province of present-day Uzbekistan

The word *algorithm* is derived from his name

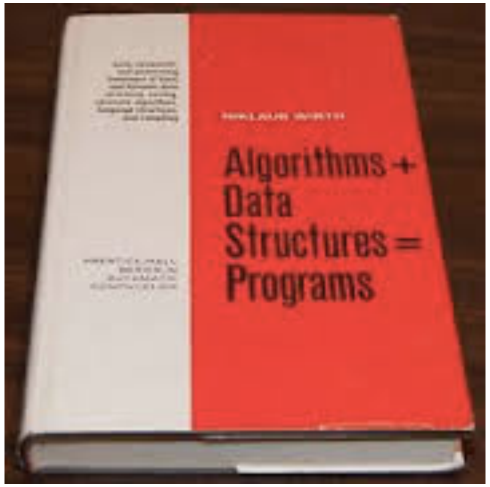
Algorithms + Data Structures = Programs



Niklaus Wirth, 1976

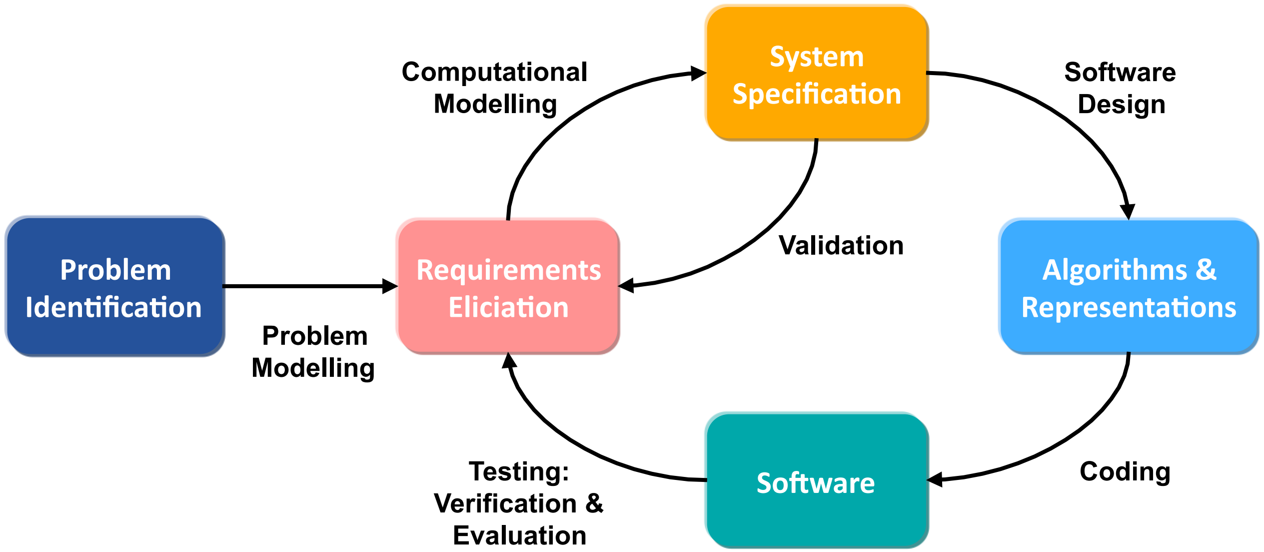
Inventor of Pascal and Modula
programming languages
Winner of Turing Award 1984

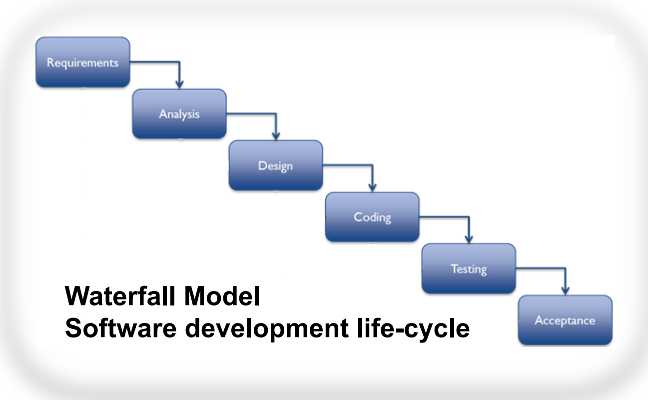

1969



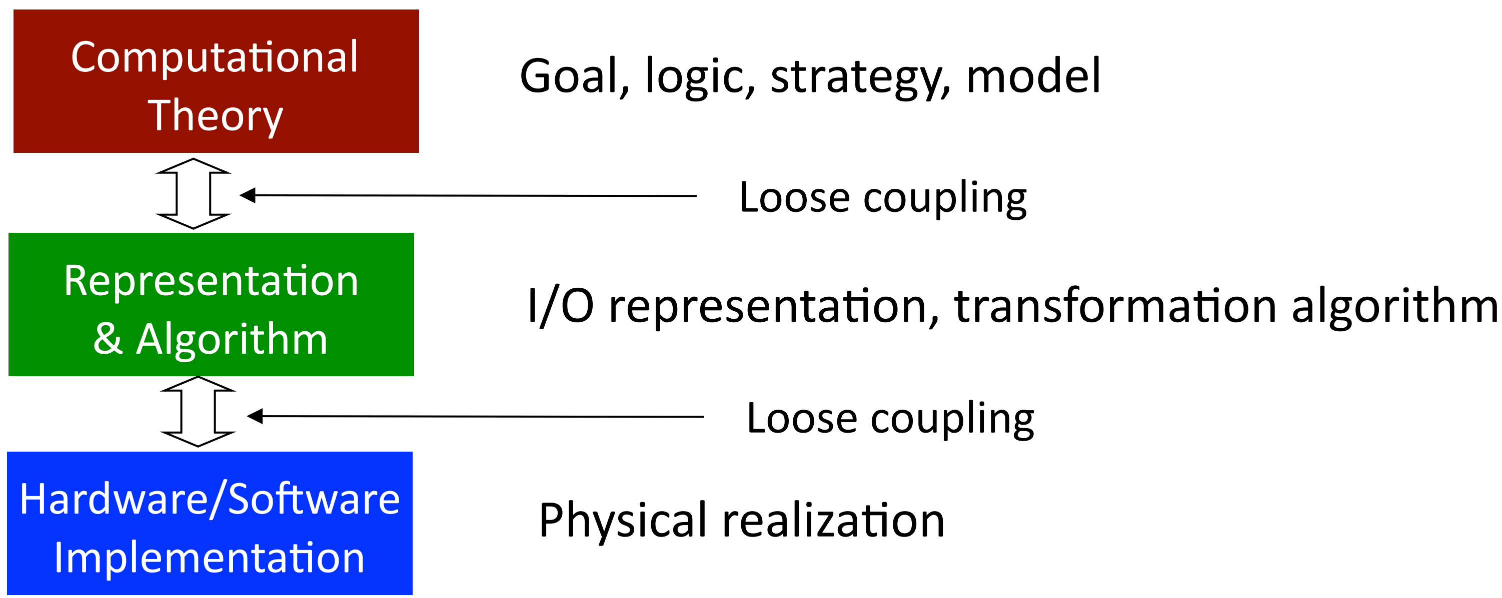
Information Processing:
Representation & Transformation

The Software Development Process





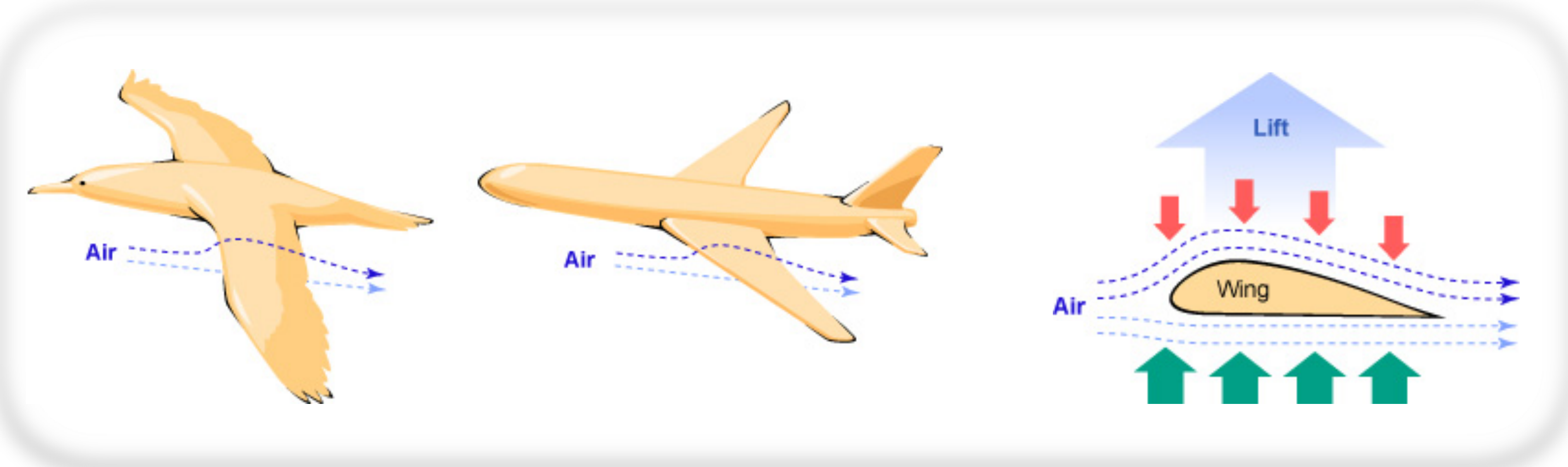
Marr's Hierarchy of Abstraction / Levels of Understanding Framework

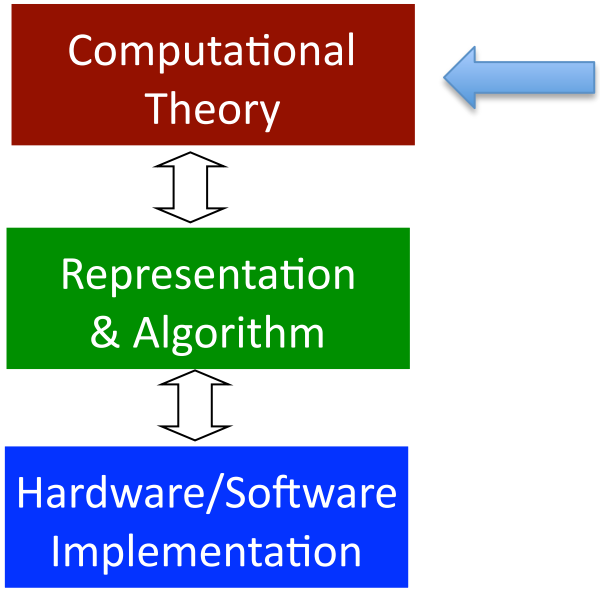


Marr's Hierarchy of Abstraction / Levels of Understanding Framework

"Trying to understand perception by studying only neurons is like trying to understand bird flight by studying only feathers: it just cannot be done. In order to understand bird flight, we have to understand aerodynamics; only then do the structure of feathers and the different shapes of birds' wings make sense"

Marr, D. *Vision*, Freeman, 1982.

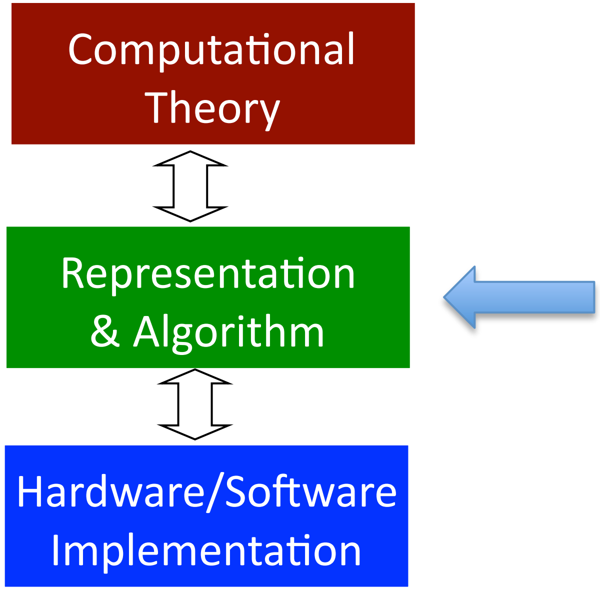




Sorting a List

Given a sequence of n keys $a_1, \ldots, a_n$

Find the permutation (reordering) such that $a_i \leq a_j$
$1 \leq i, j \leq n$



Sorting a List

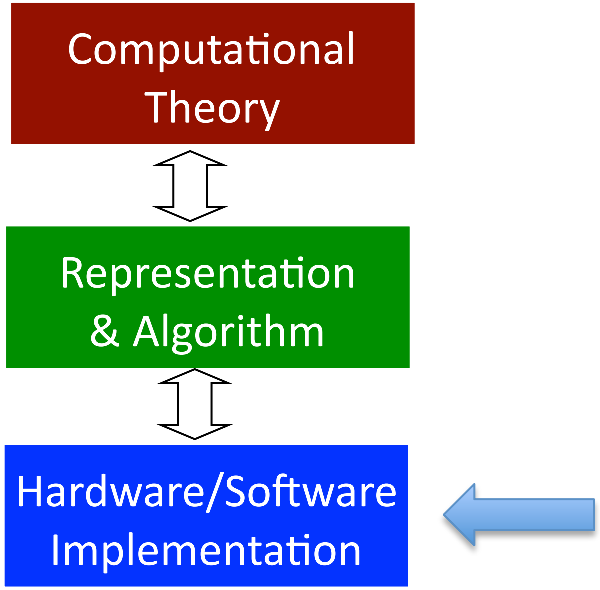
Bubble Sort

Insertion Sort

Quick Sort

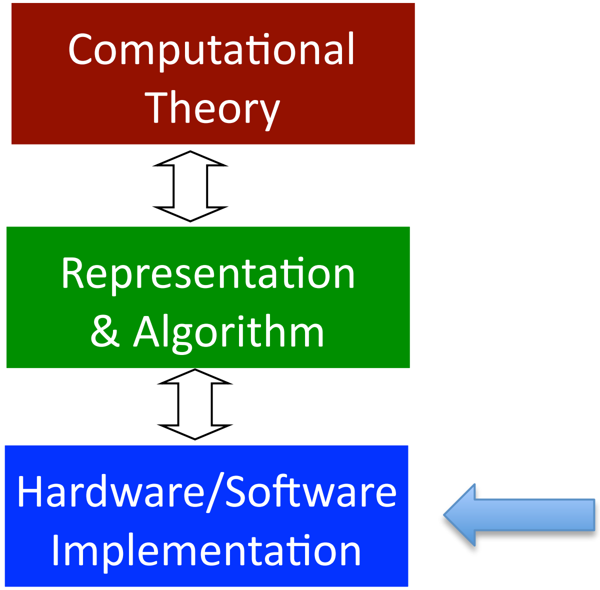
Merge Sort, …

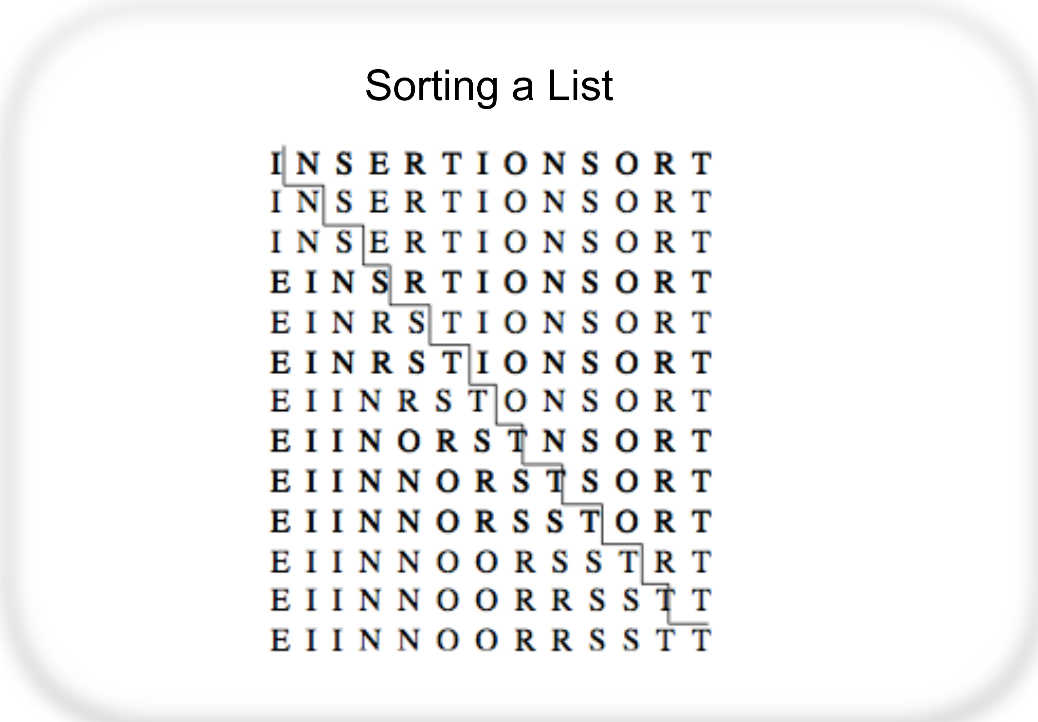
Key point: different computational efficiency

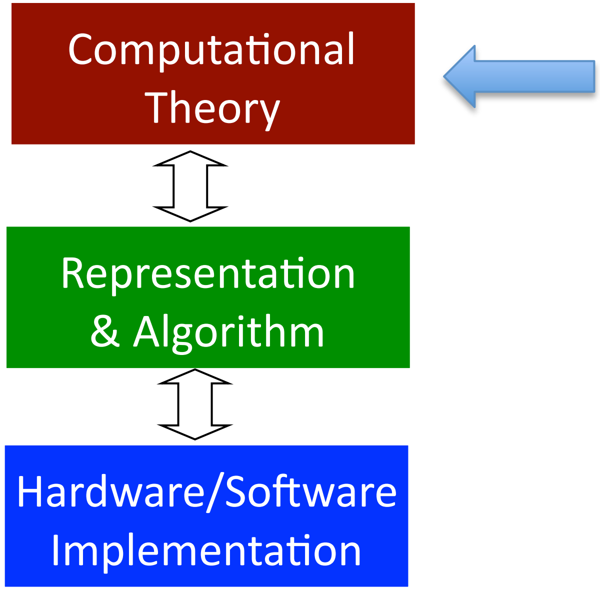


Sorting a List

```
insertion_sort(item s[], int n)
{
        int i,j;                /* counters */

        for (i=1; i<n; i++) {
                j=i;
                while ((j>0) && (s[j] < s[j-1])) {
                        swap(&s[j],&s[j-1]);
                        j = j-1;
                }
        }
}
```


Computational Theory
Representation & Algorithm
Hardware/Software Implementation

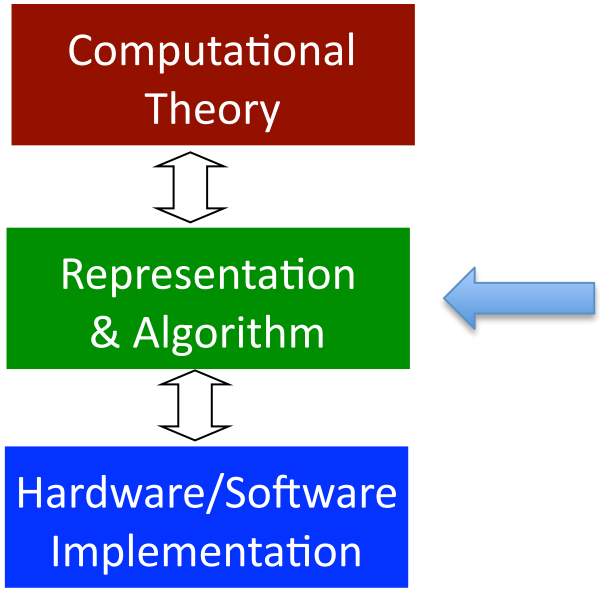

Sorting a List
I N S E R T I O N S O R T
I N S E R T I O N S O R T
I N S E R T I O N S O R T
E I N S R T I O N S O R T
E I N R S T I O N S O R T
E I N R S T I O N S O R T
E I I N R S T O N S O R T
E I I N O R S T N S O R T
E I I N N O R S T S O R T
E I I N N O R S S T O R T
E I I N N O O R S S T R T
E I I N N O O R R S S T T
E I I N N O O R R S S T T



Fourier Transform

$$
\begin{aligned}
\mathcal{F}\left(f(x,y)\right) &= \mathsf{F}(\omega_x, \omega_y) \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-i(\omega_x x + \omega_y y)} \mathrm{d}x\mathrm{d}y
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{F}\left(f(x,y)\right) &= \mathsf{F}(\omega_x, \omega_y) \\
&= \mathsf{F}(\omega_x \Delta_{\omega_x}, \omega_y \Delta_{\omega_y}) \\
&= \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-i(\frac{\omega_x x}{M} + \frac{\omega_y y}{N})}
\end{aligned}
$$



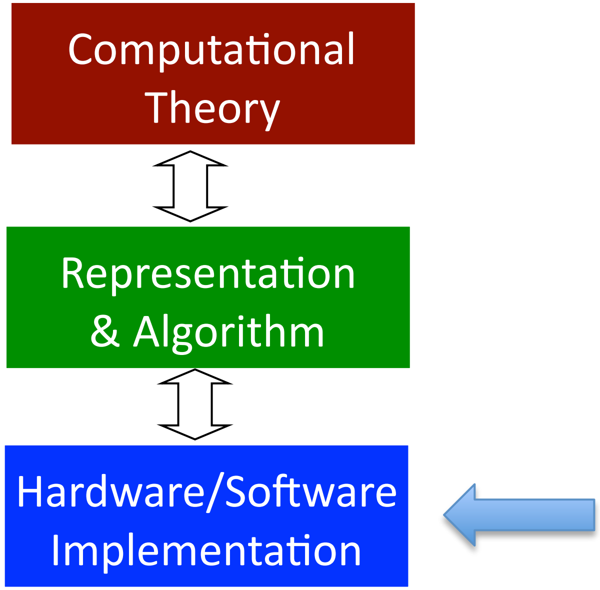
Fourier Transform

DFT: Discrete Fourier Transform

FFT: Fast Fourier Transform

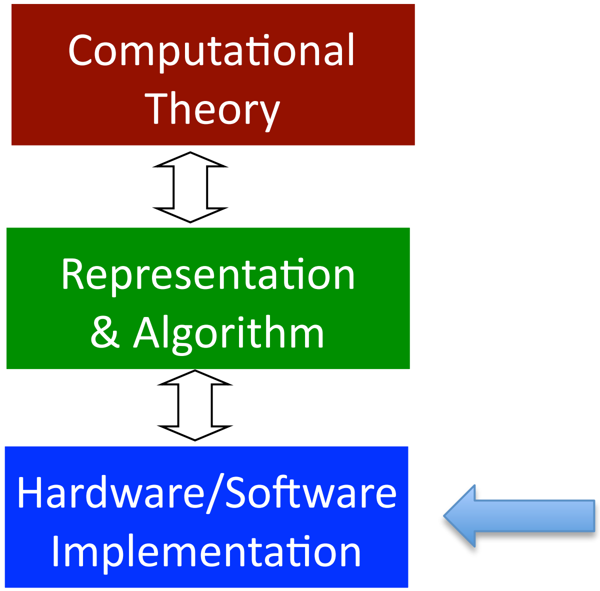
FFTW: Fasted Fourier Transform in the West

Key point: different computational efficiency

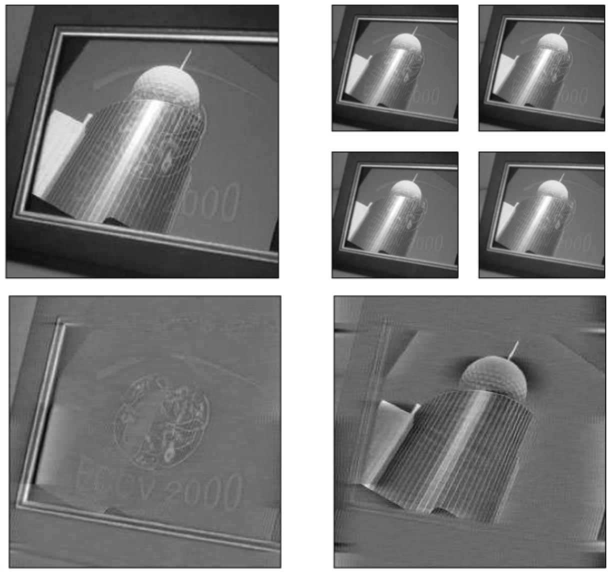


Fourier Transform

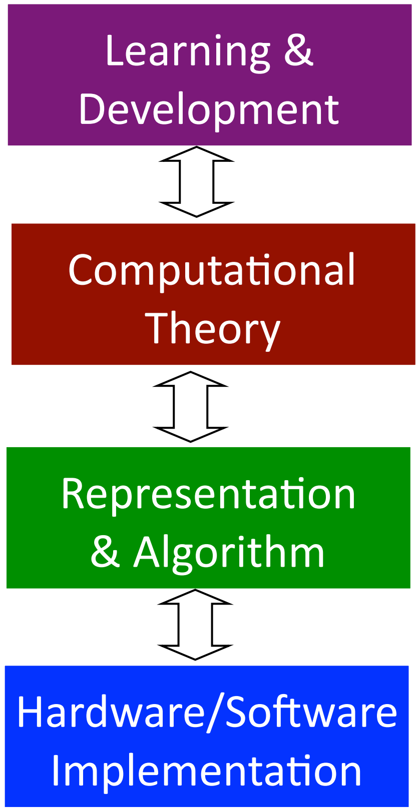
```
main()
{
        unsigned long i;
        int isign;
        float *data1,*data2,*fft1,*fft2;

        data1=vector(1,N);
        data2=vector(1,N);
        fft1=vector(1,N2);
        fft2=vector(1,N2);
        for (i=1;i<=N;i++) {
                data1[i]=floor(0.5+cos(i*2.0*PI/PER));
                data2[i]=floor(0.5+sin(i*2.0*PI/PER));
        }
        twofft(data1,data2,fft1,fft2,N);
        printf("Fourier transform of first function:\n");
        prntft(fft1,N);
        printf("Fourier transform of second function:\n");
        prntft(fft2,N);
        /* Invert transform */
        isign = -1;
        four1(fft1,N,isign);
        printf("inverted transform  =  first function:\n");
        prntft(fft1,N);
        four1(fft2,N,isign);
        printf("inverted transform  =  second function:\n");
        prntft(fft2,N);
        free_vector(fft2,1,N2);
        free_vector(fft1,1,N2);
        free_vector(data2,1,N);
        free_vector(data1,1,N);
        return 0;
}
```



Fourier Transform



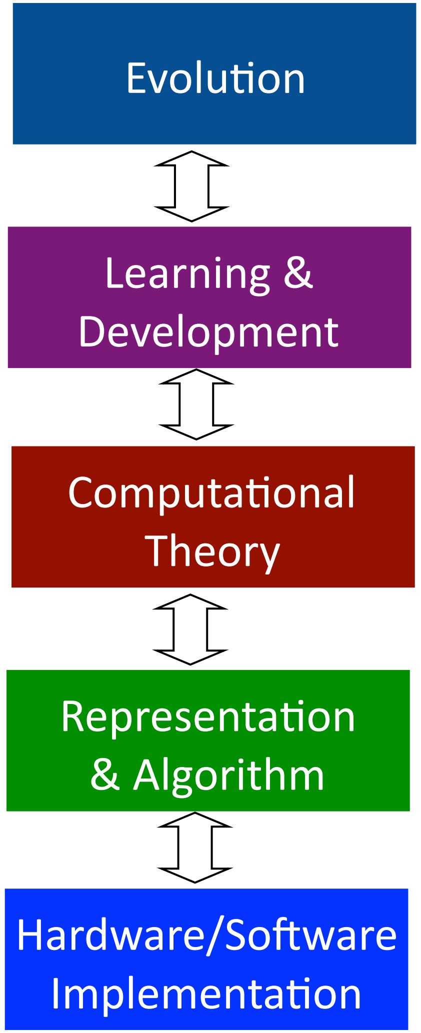
Marr's Levels of Understanding Framework updated 2012 by T. Poggio



Calibrating & improving the model

CS-AI-421 Artificial Cognitive Systems
CS-AI-422 Machine Learning

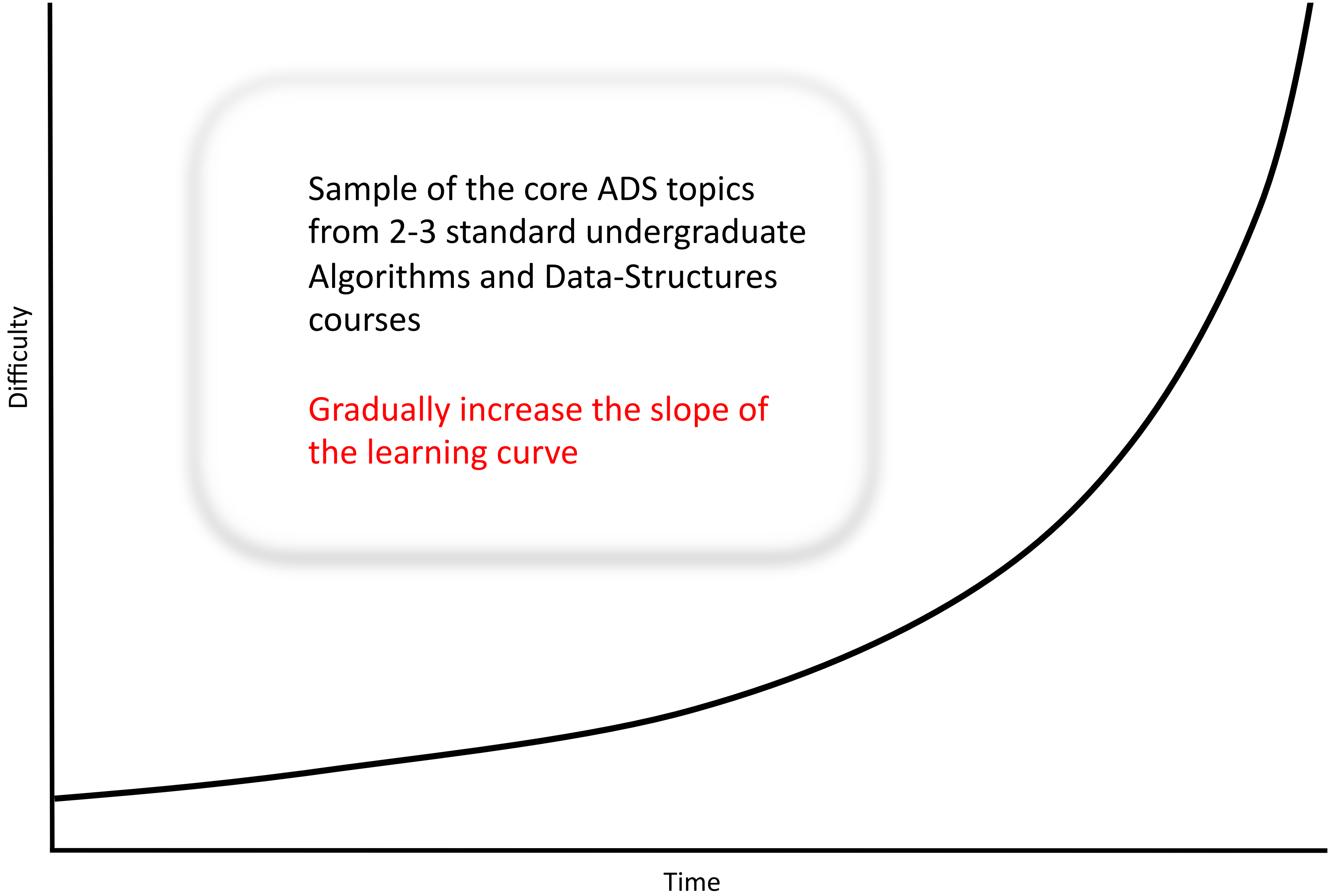
Marr's Levels of Understanding Framework updated 2012 by T. Poggio



Generating new models

Calibrating & improving the model

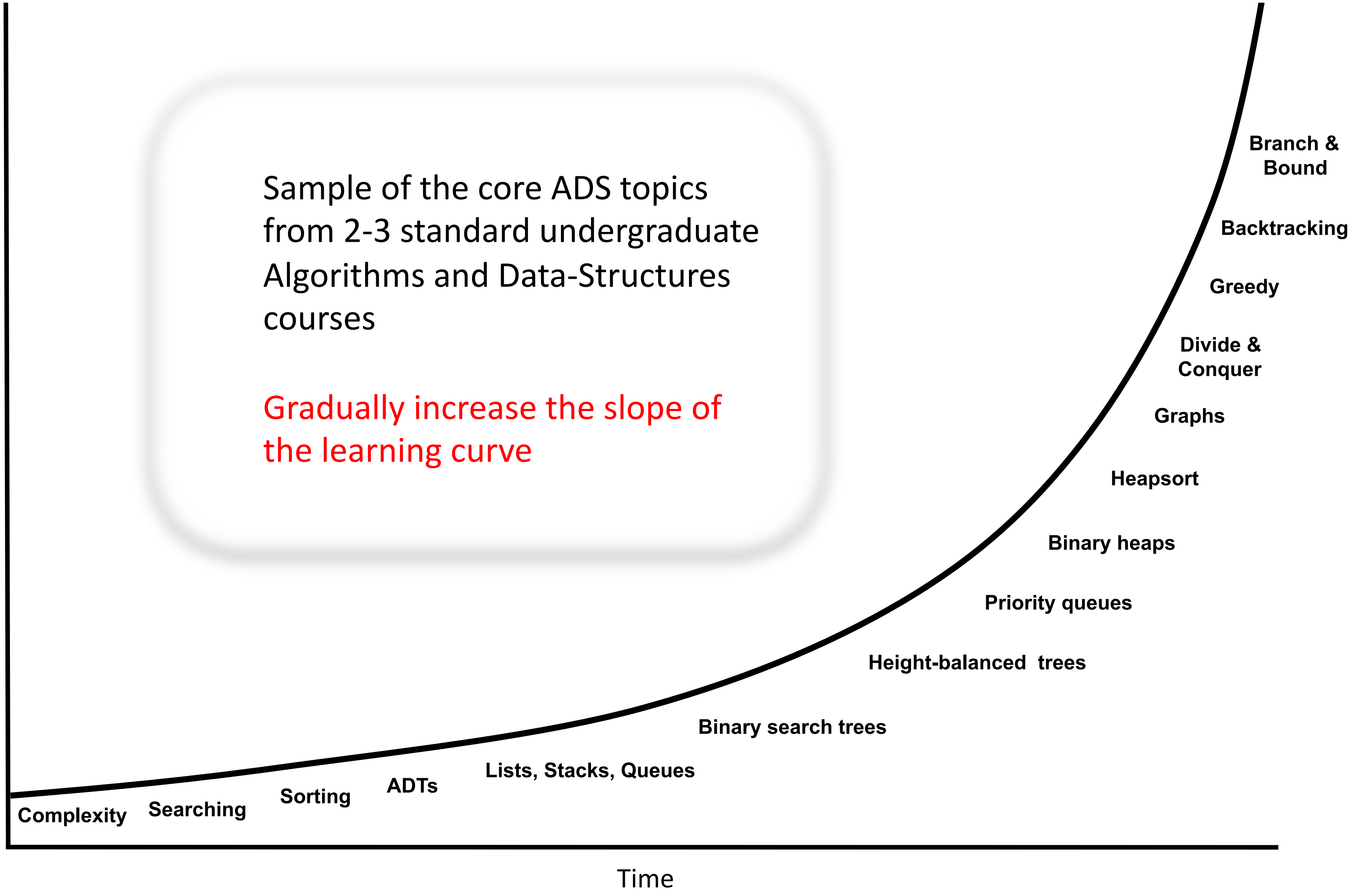
Algorithms and Data-Structures Strategy



Sample of the core ADS topics from 2-3 standard undergraduate Algorithms and Data-Structures courses

Gradually increase the slope of the learning curve

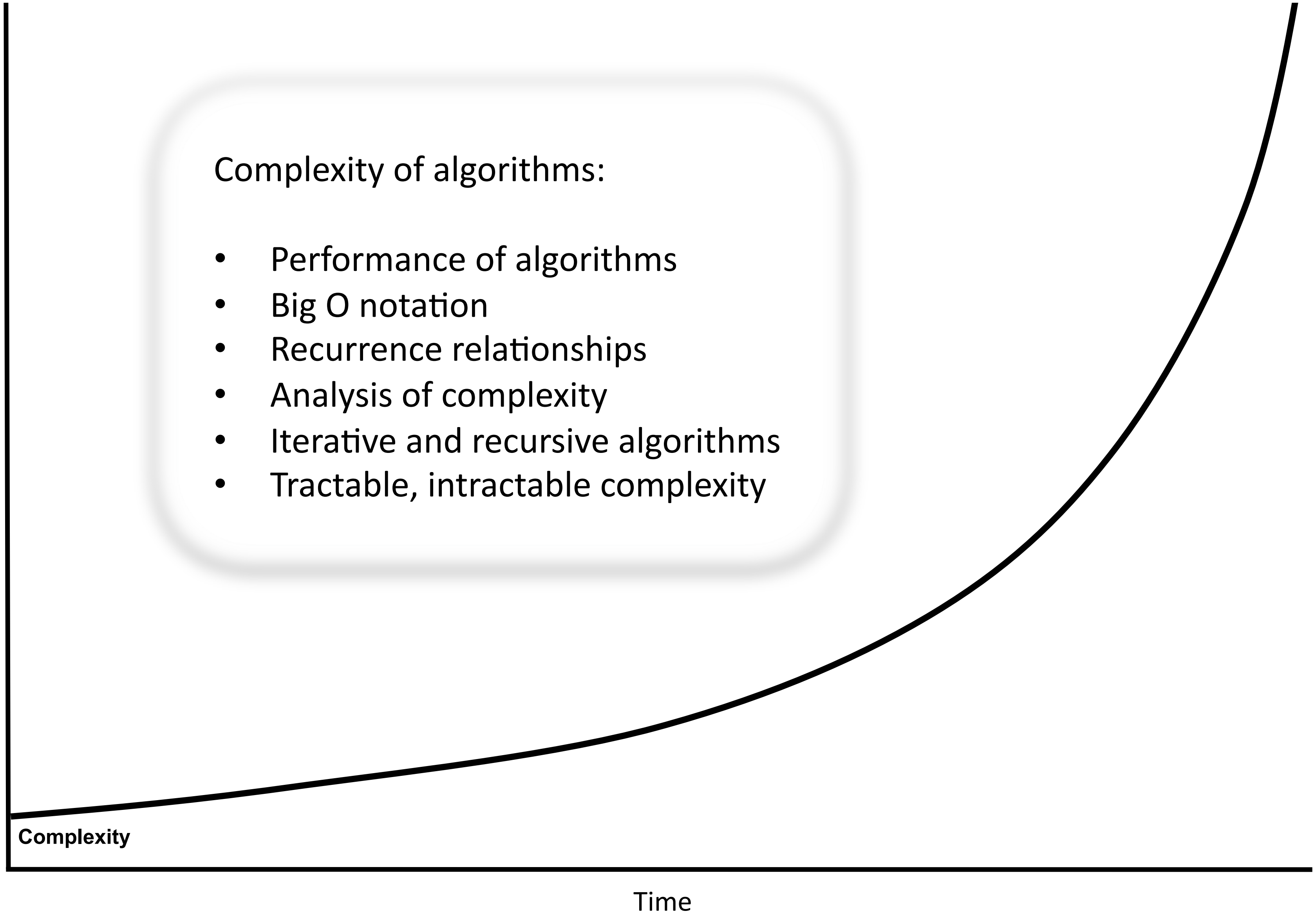
Algorithms and Data-Structures Topics

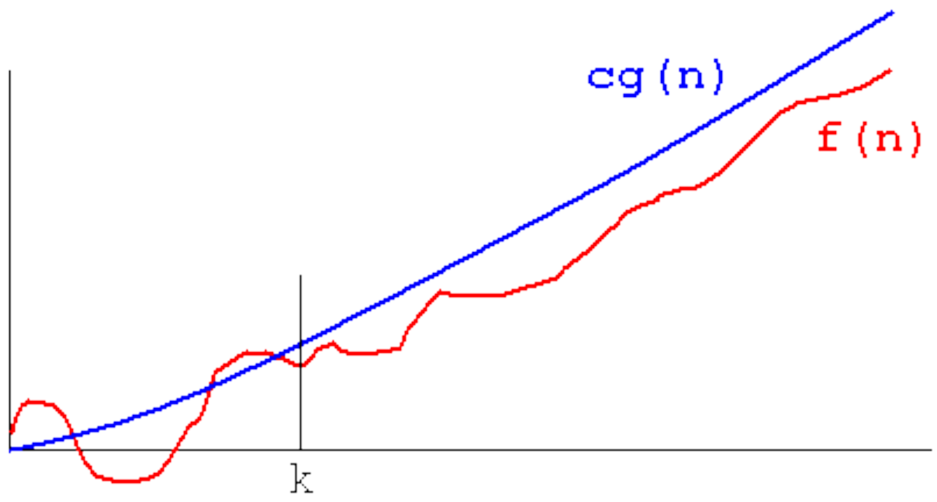


Algorithms and Data-Structures Topics

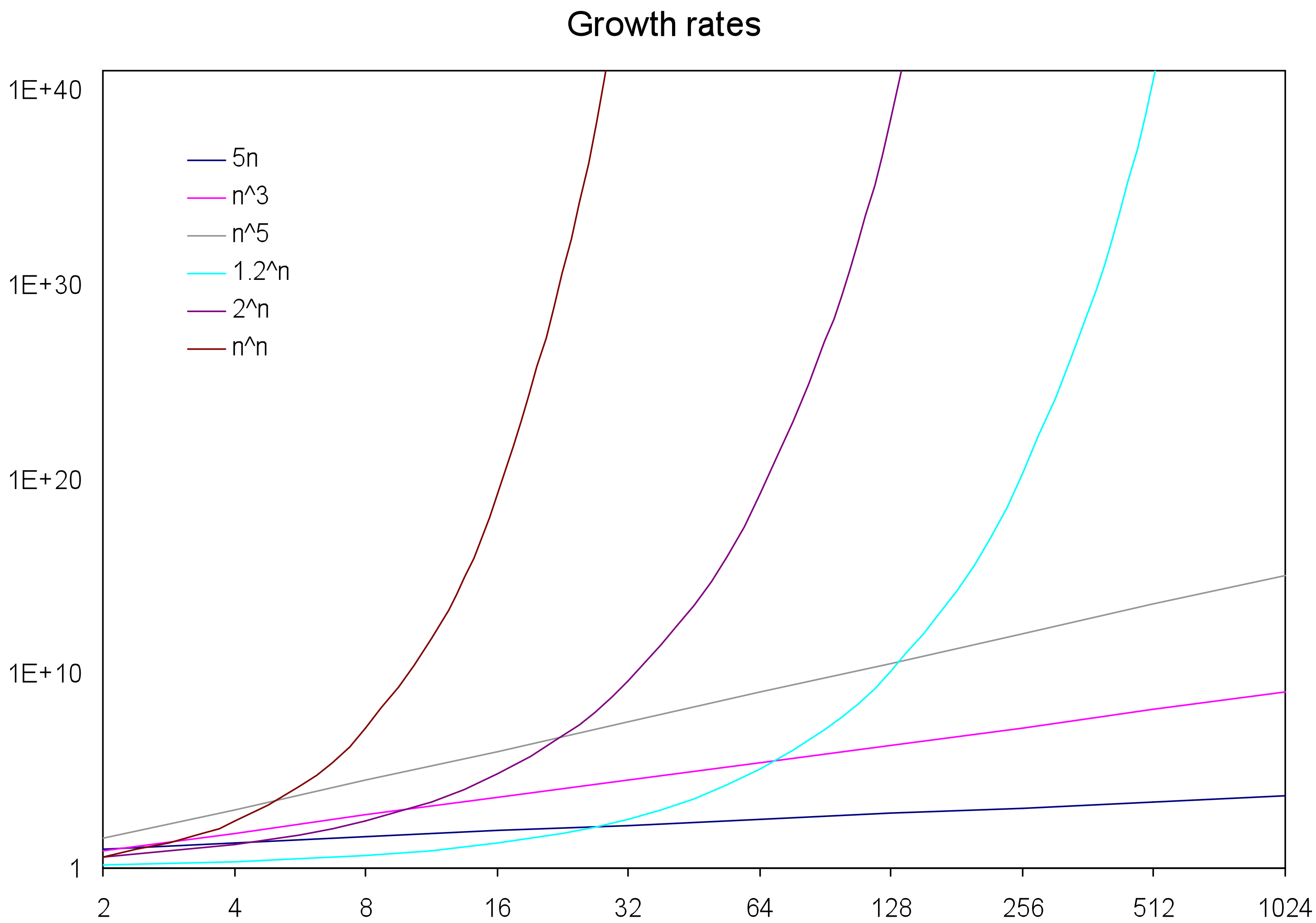
Complexity of algorithms:

- Performance of algorithms
- Big O notation
- Recurrence relationships
- Analysis of complexity
- Iterative and recursive algorithms
- Tractable, intractable complexity





Use Big O notation to determine an upper bound function g(n) that characterizes how the complexity grows with the size of the data set … $O(g(n))$ e.g. $O(n \log_2 n)$

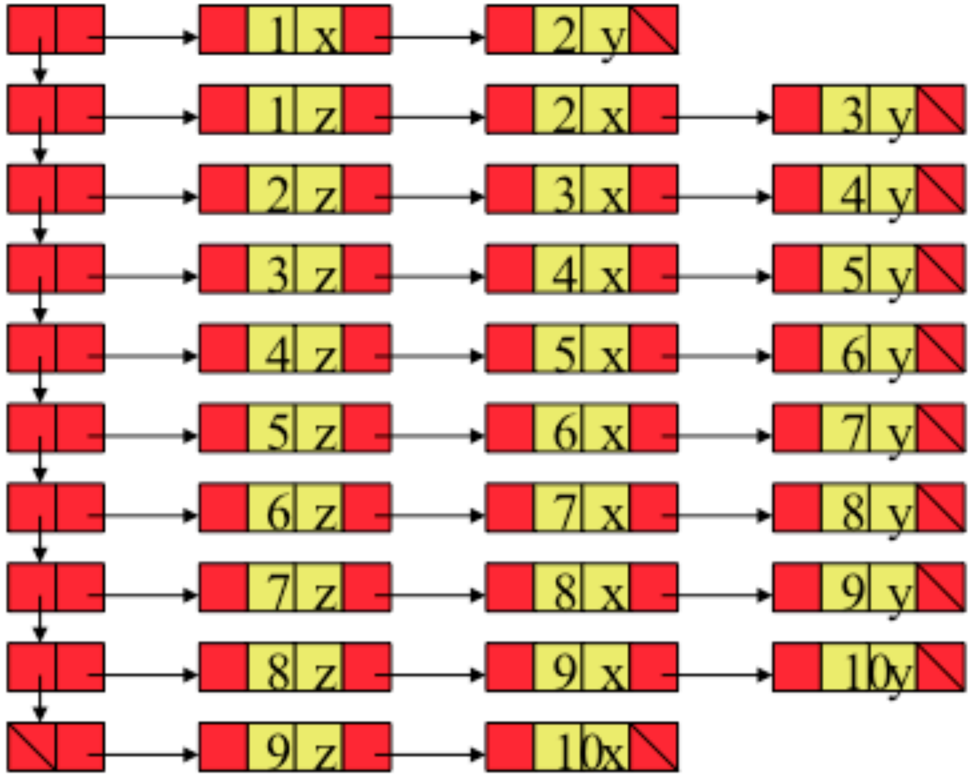

Growth rates
1E+40
1E+30
1E+20
1E+10
1
5n
n^3
n^5
1.2^n
2^n
n^n
2
4
8
16
32
64
128
256
512
1024

	function/ n	10	20	50	100	300
Polynomial	n^2	1/10,000 second	1/2,500 second	1/400 second	1/100 second	9/100 second
	n^5	1/10 second	3.2 seconds	5.2 minutes	2.8 hours	28.1 days
Exponential	2^n	1/1000 second	1 second	35.7 years	400 trillion centuries	a 75 digit-number of centuries
	n^n	2.8 hours	3.3 trillion years	a 70 digit-number of centuries	a 185 digit-number of centuries	a 728 digit-number of centuries

x	y	0	0	0	0	0	0	0	0
z	x	y	0	0	0	0	0	0	0
0	z	x	y	0	0	0	0	0	0
0	0	z	x	y	0	0	0	0	0
0	0	0	z	x	y	0	0	0	0
0	0	0	0	z	x	y	0	0	0
0	0	0	0	0	z	x	y	0	0
0	0	0	0	0	0	z	x	y	0
0	0	0	0	0	0	0	z	x	y
0	0	0	0	0	0	0	0	z	x

n x n matrix:

$O(n^2)$ space complexity



$2x(2 + 4 + 4) + (n-2)x(2 + 4 + 4 + 4)$

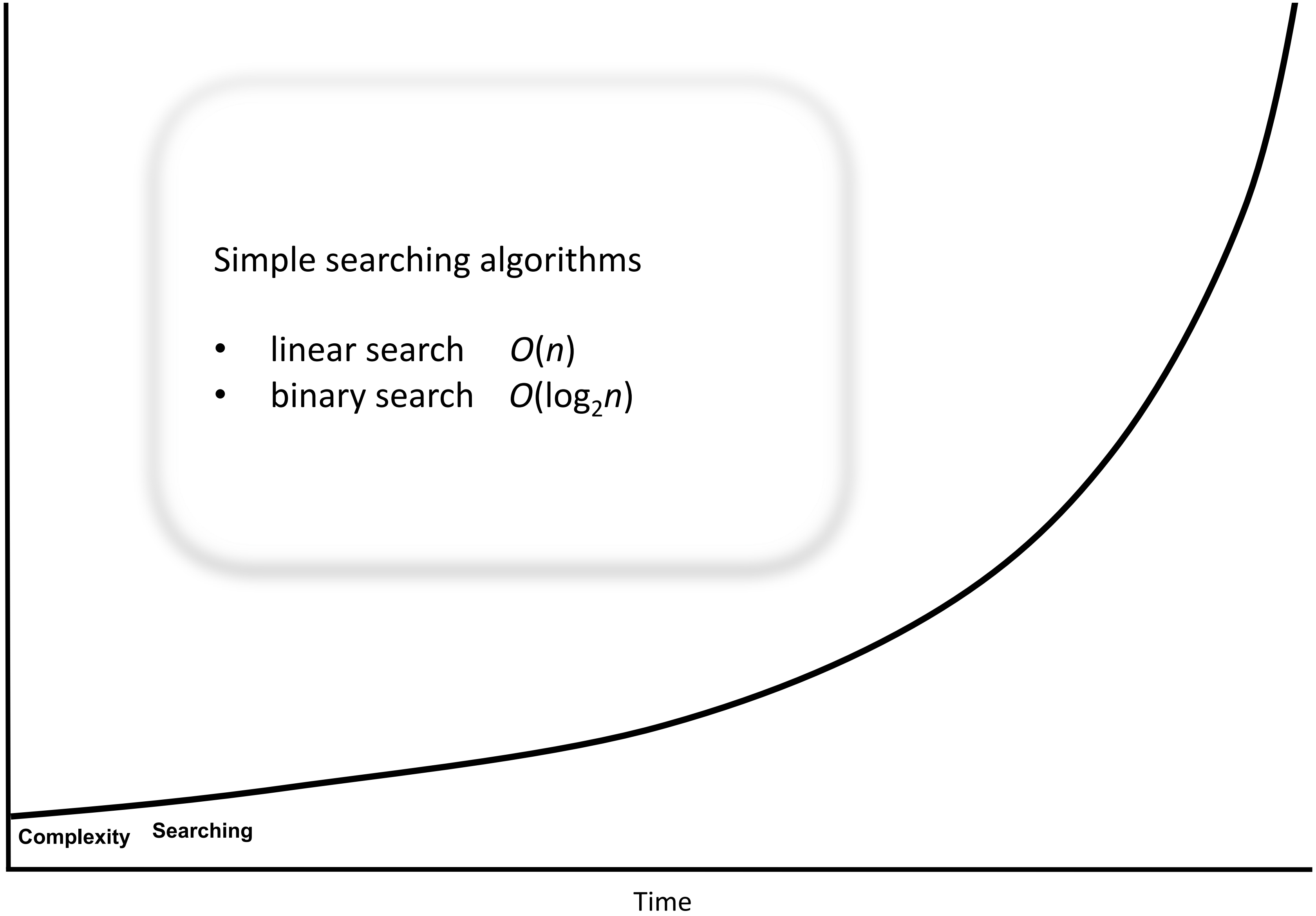
$= 20 + 14n - 28 = 14n - 8$:

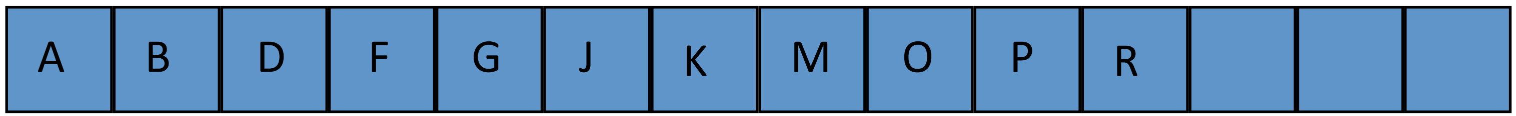
$O(n)$ space complexity

Algorithms and Data-Structures Topics

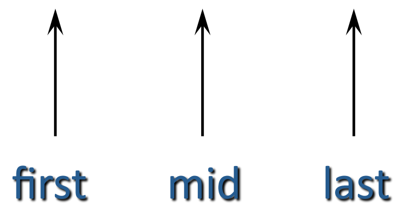
Simple searching algorithms

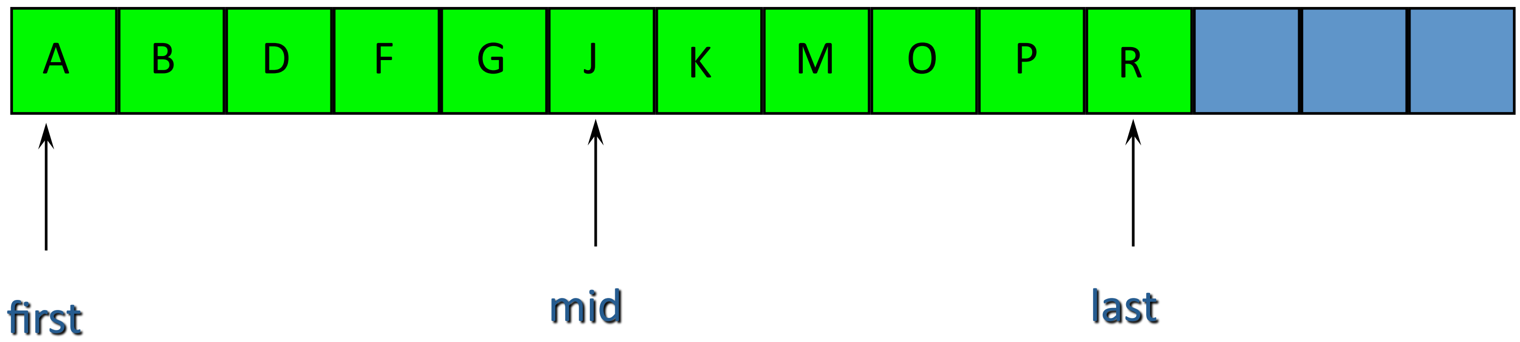
- linear search $O(n)$
- binary search $O(\log_2 n)$



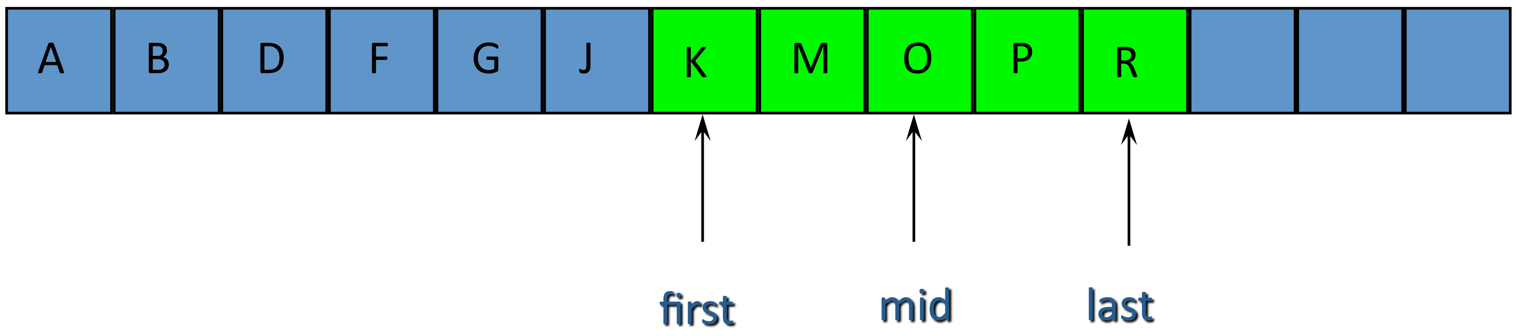



```
first:
last:
mid:
list[mid]:
key:        P
```

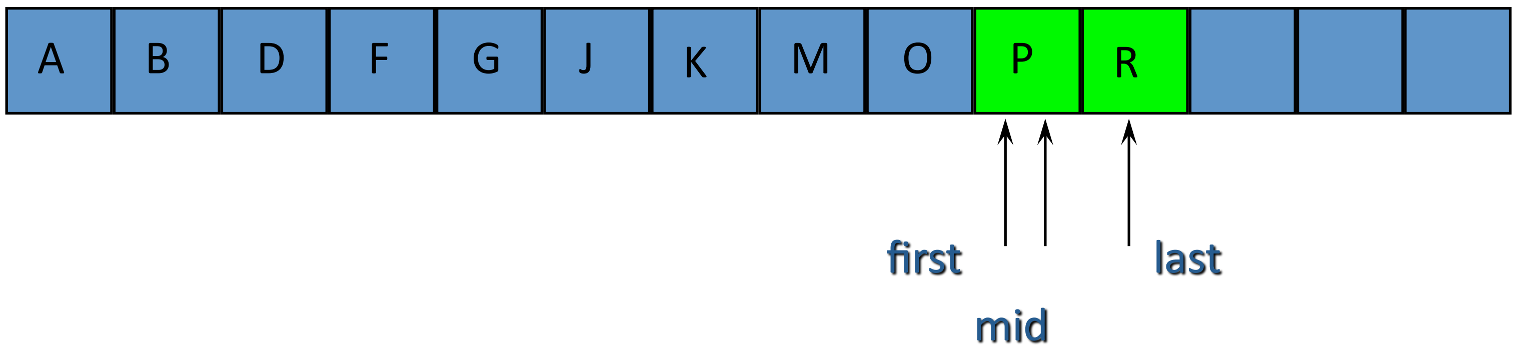





```
first:      1
last:      11
mid:        6
list[mid]: J
key:        P
```

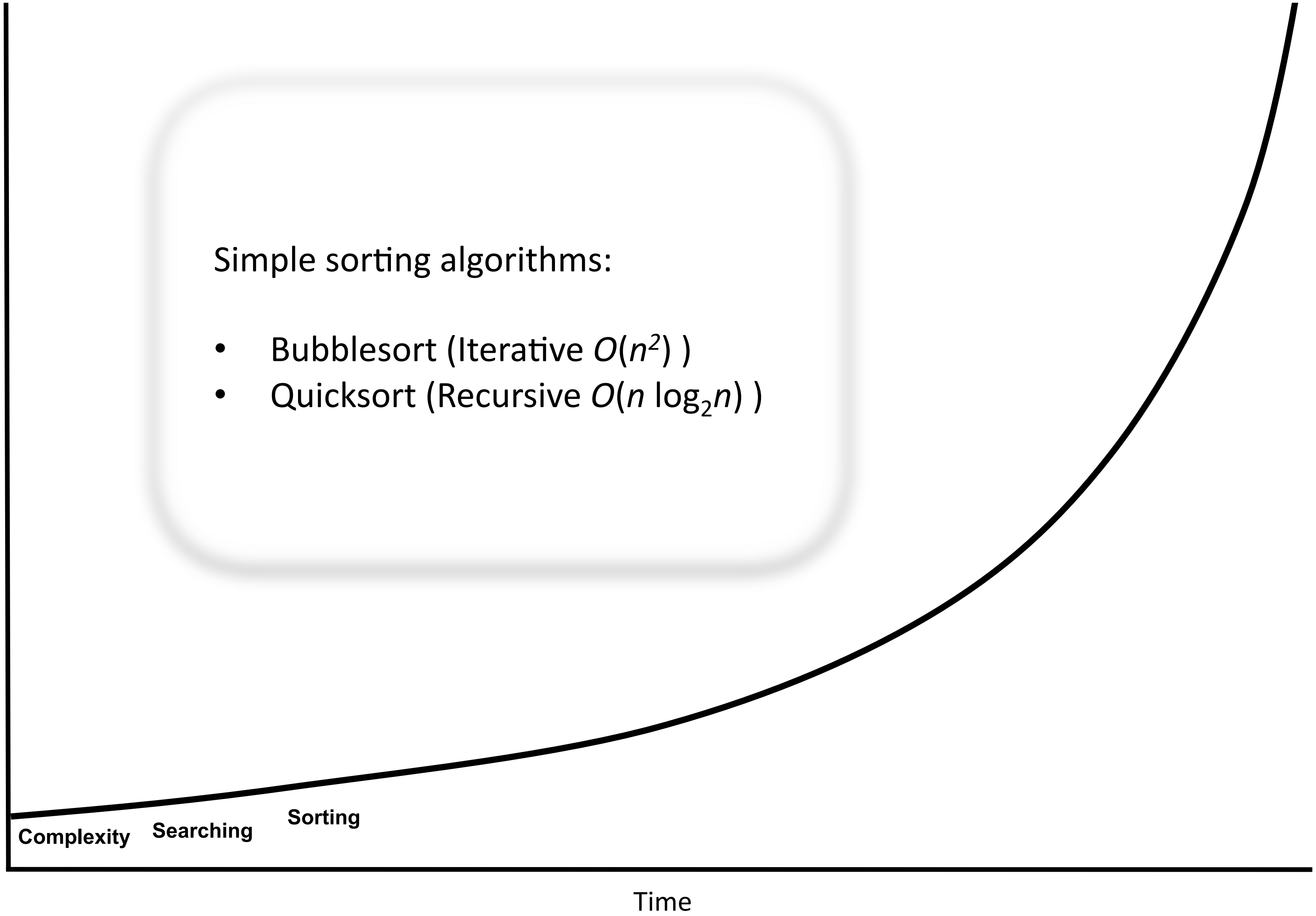

```
first:       1   7
last:       11  11
mid:         6   9
list[mid]: J   O
key:         P   P
```

first:	1	7	10
last:	11	11	11
mid:	6	9	10
list[mid]:	J	O	P
key:	P	P	P

← FOUND!

Algorithms and Data-Structures Topics



Simple sorting algorithms:

- Bubblesort (Iterative $O(n^2)$)
- Quicksort (Recursive $O(n \log_2 n)$)

```
/* Quicksort: assume A[R] contains a sentinel */

void quicksort (int A[], int L, int R)
{
   int i, j, pivot;
   if ( R > L) {
      i = L; j = R;
      pivot = A[i];
      do {
         while (A[i] <= pivot) i=i+1;
         while ((A[j] >= pivot) && (j>l)) j=j-1;
         if (i < j) {
            exchange(A[i],A[j]); /*between partitions*/
            i = i+1; j = j-1;
         }
      } while (i <= j);
      exchange(A[L], A[j]); /* reposition pivot */
      quicksort(A, L, j);
      quicksort(A, i, R);   /*includes sentinel*/
   }
}
```


4	9	8	10	11	99

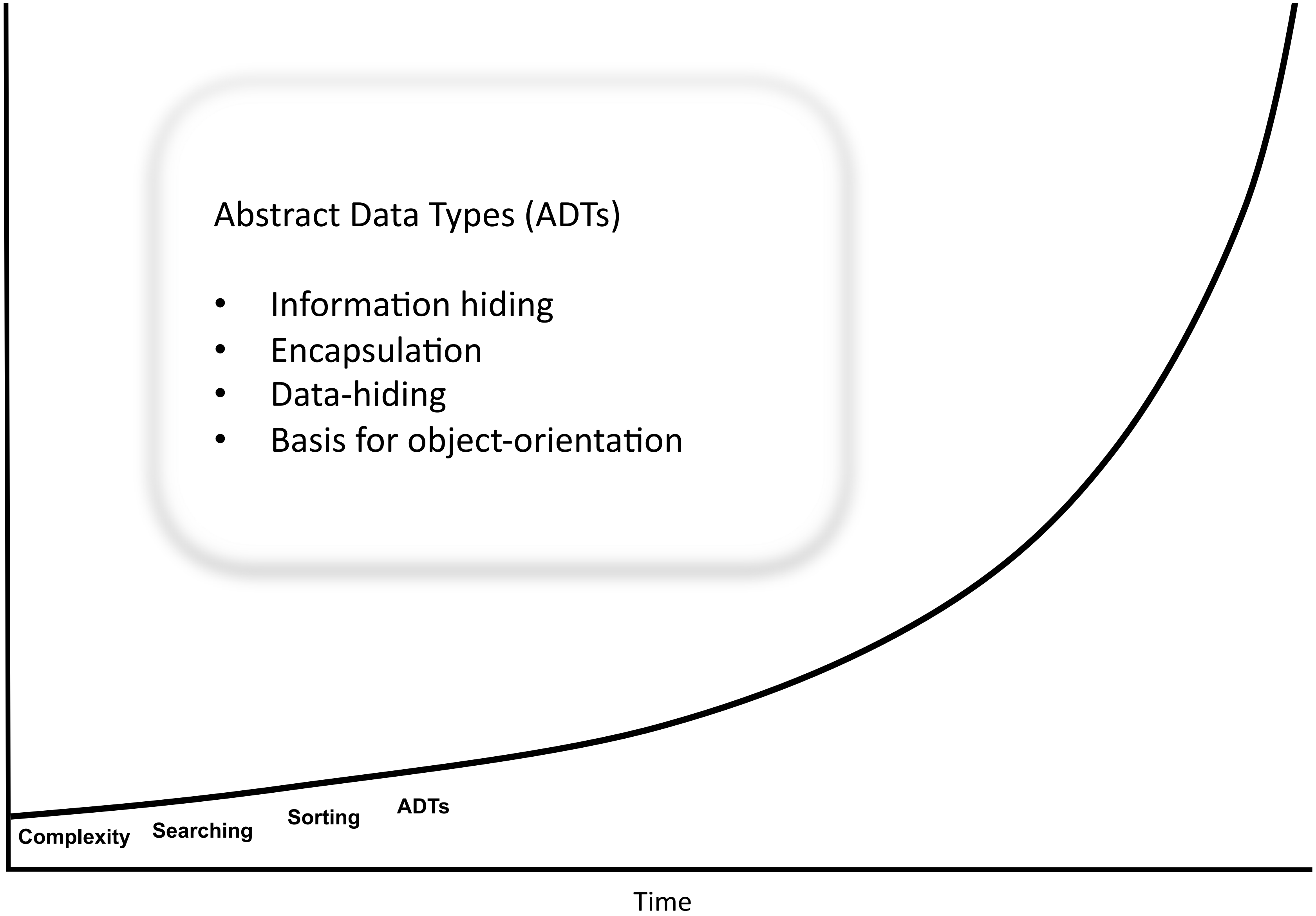
i j

QS(A,1,6)		QS(A,1,4)		QS(A,5,6)	
L:	1	L:	1	L:	5
R:	6	R:	4	R:	6
i:	1 2 3 4 5	i:		i:	
j:	6 5 4	j:		j:	
pivot:	10	pivot:	4	pivot:	11

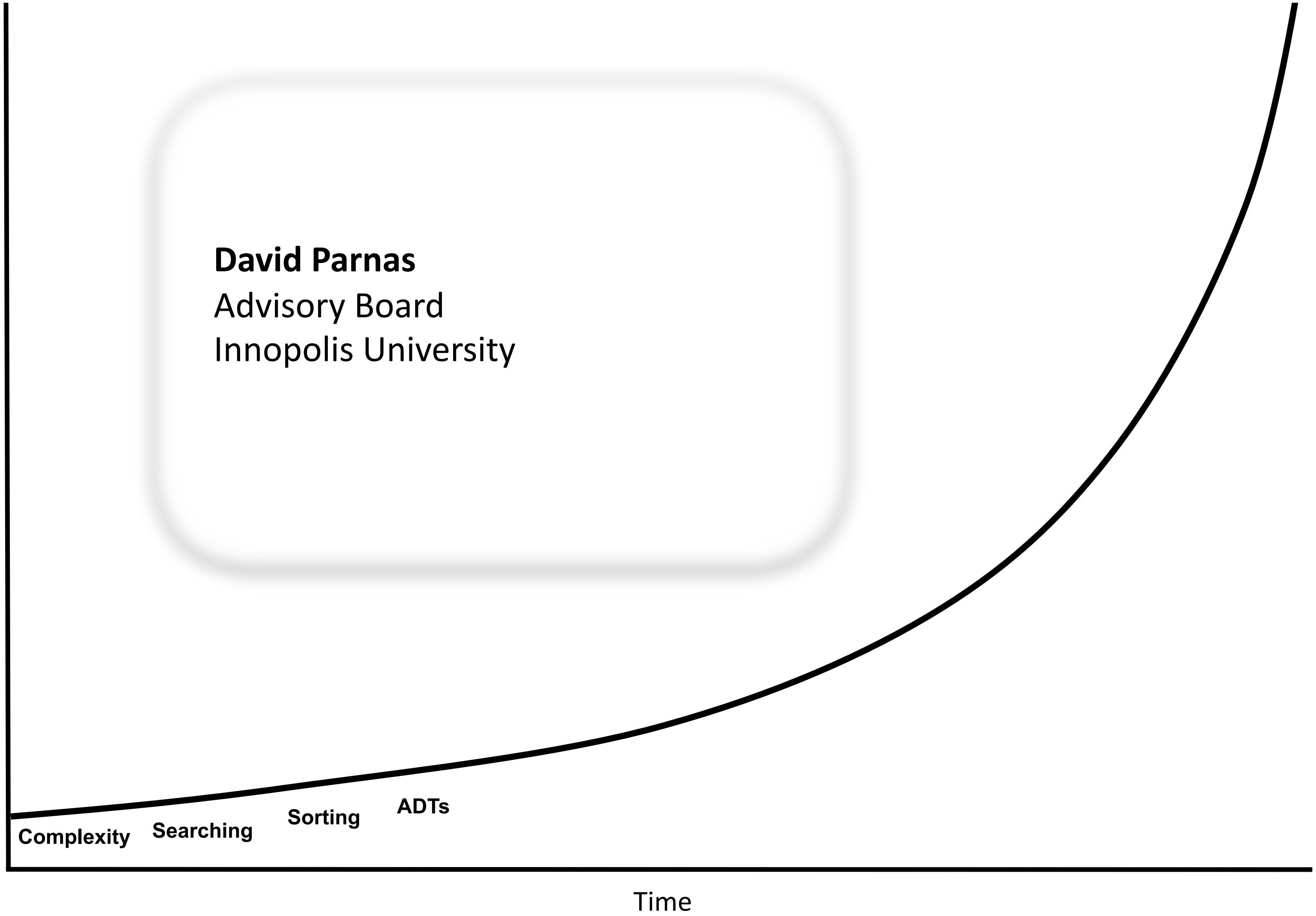
Algorithms and Data-Structures Topics

Abstract Data Types (ADTs)

- Information hiding
- Encapsulation
- Data-hiding
- Basis for object-orientation



Algorithms and Data-Structures Topics

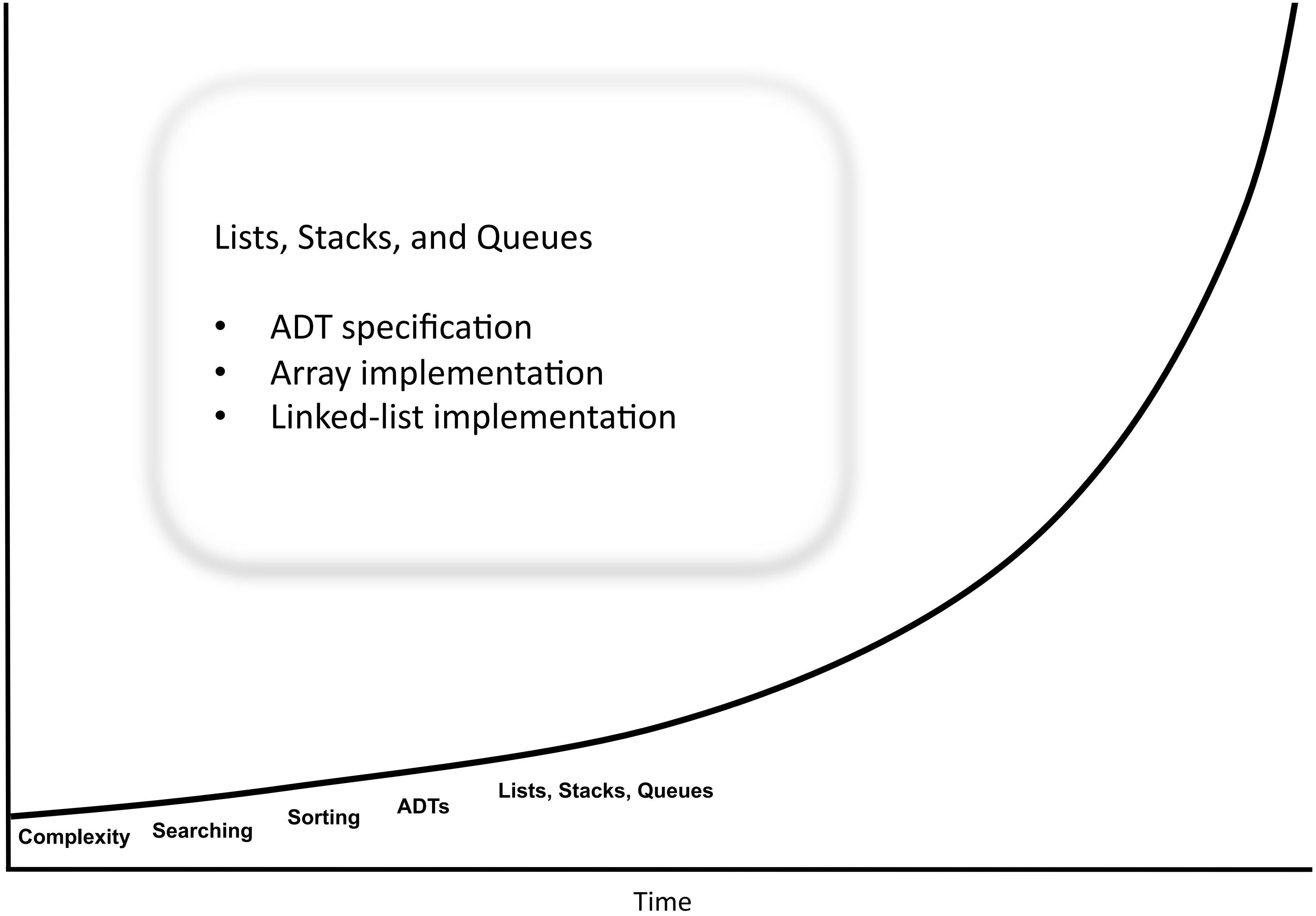


David Parnas
Advisory Board
Innopolis University

Algorithms and Data-Structures Topics

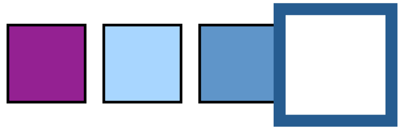
Lists, Stacks, and Queues

- ADT specification
- Array implementation
- Linked-list implementation

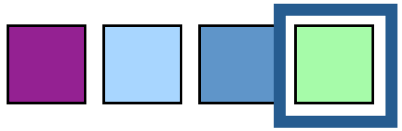


- *Example of List manipulation*

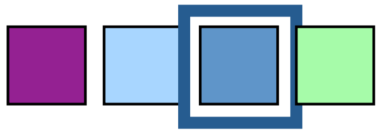
w = Next(Last(L),L)



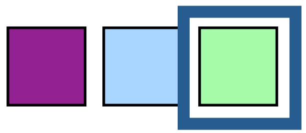
Insert(e,w,L)

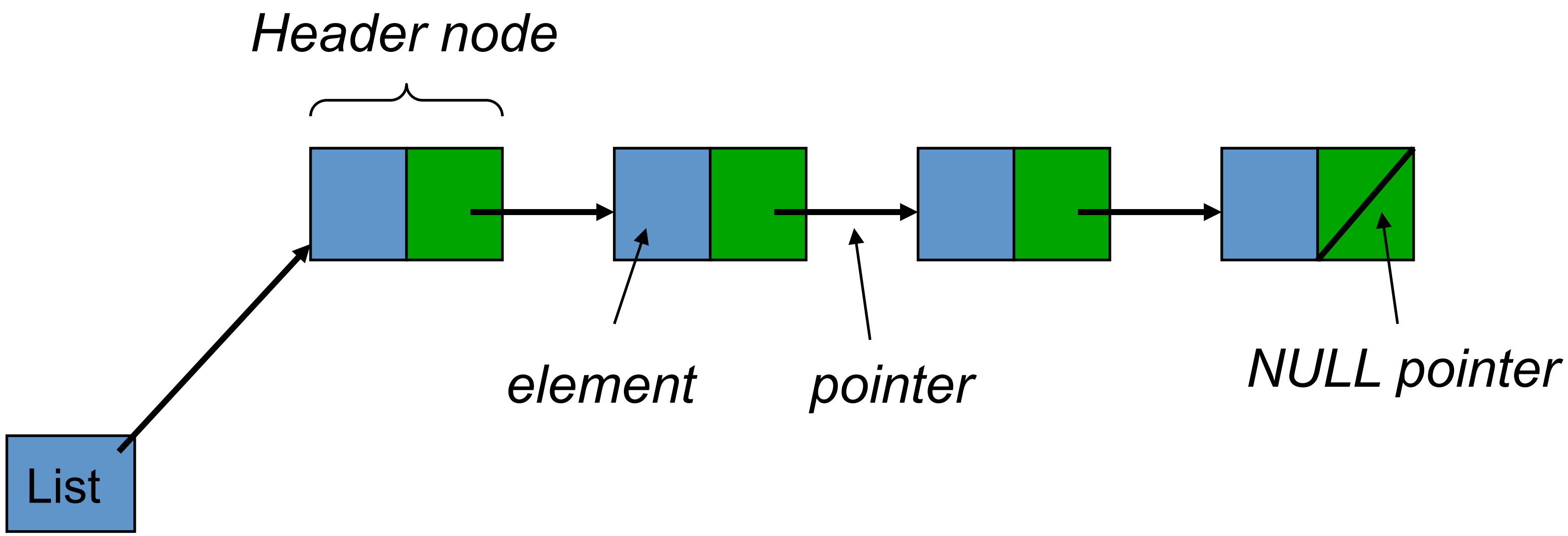


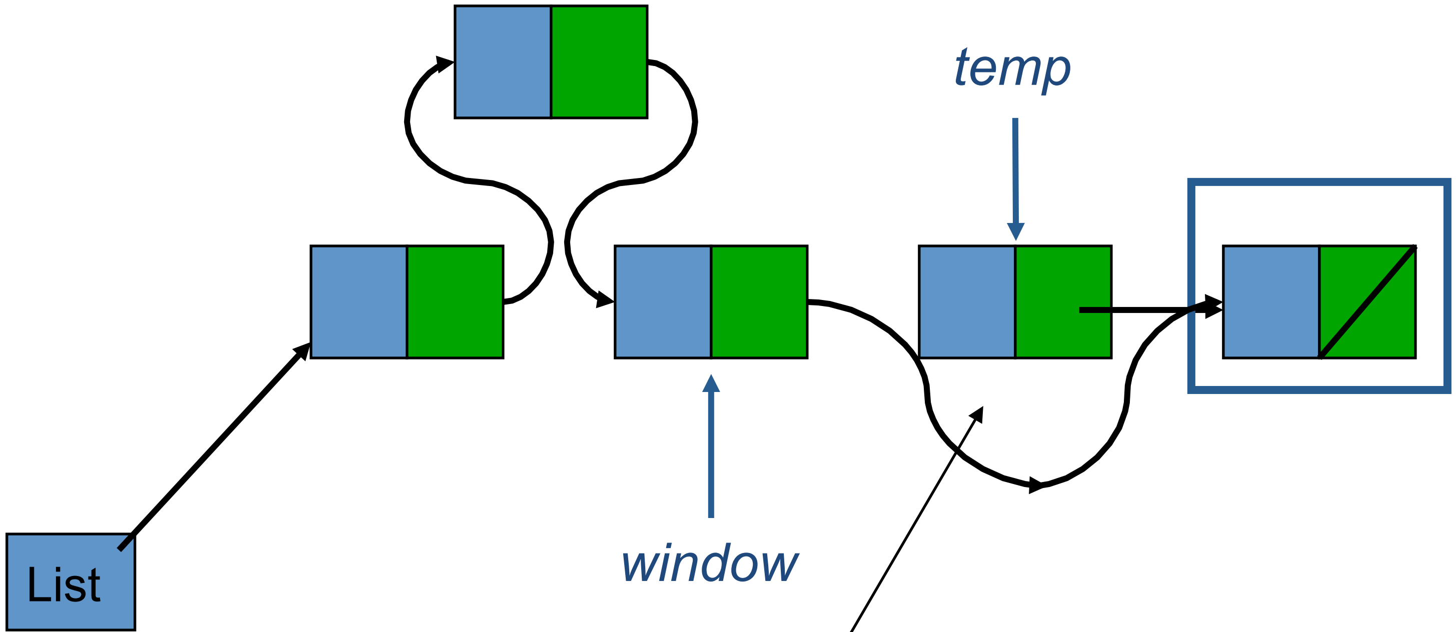
w = Previous(w,L)



Delete(w,L)




Header node
element
pointer
NULL pointer
List

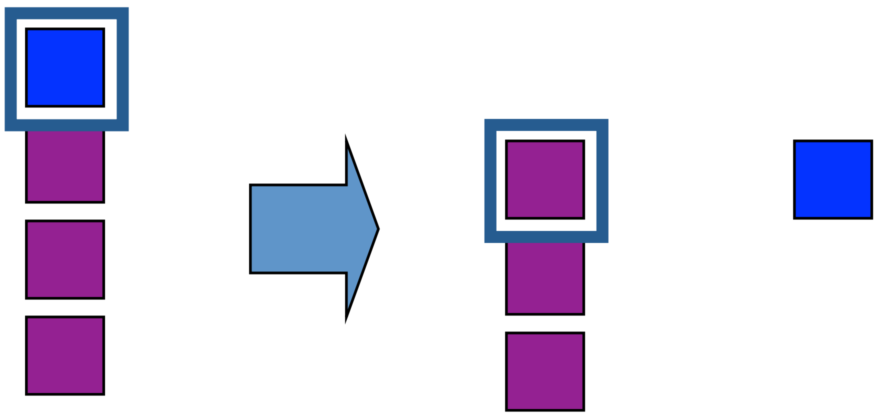


To delete a node at this window position we re-arrange the links and free the node

Pop: $S \rightarrow E$:

Pop(S)

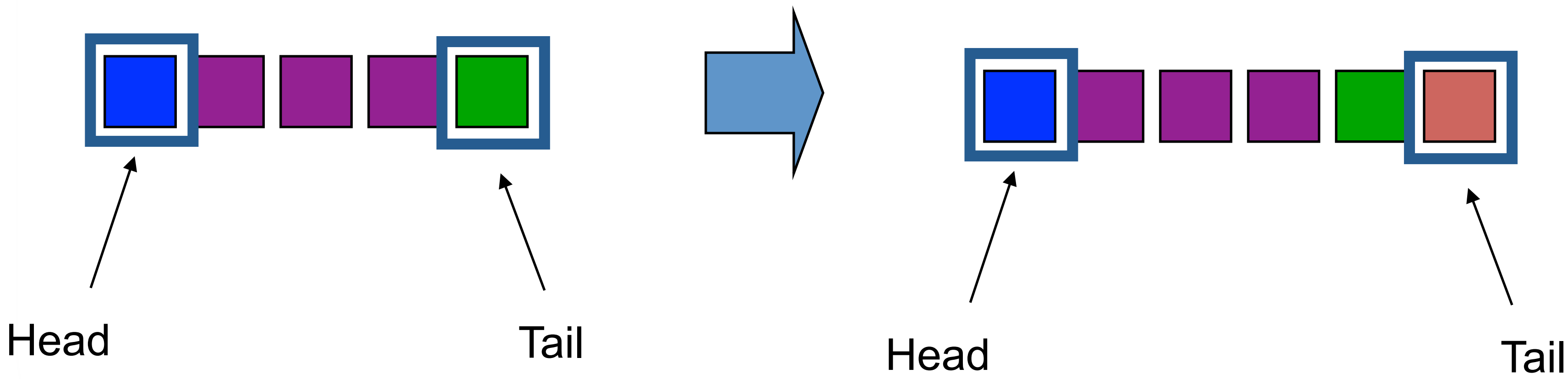
Remove the top element from the stack: i.e. return the top element and delete it from the stack



Enqueue: $E \times Q \rightarrow Q$:

Enqueue(e, Q)

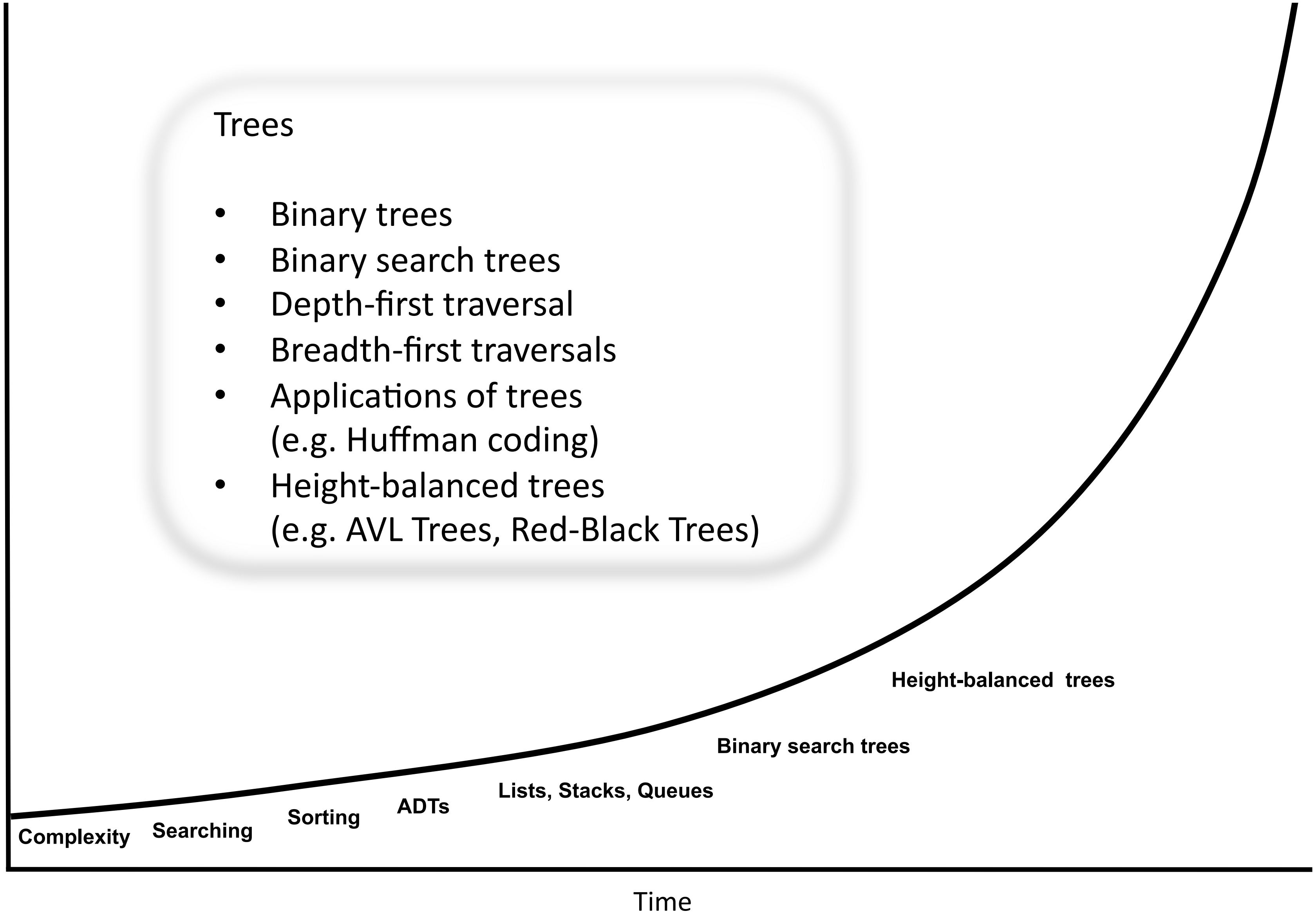
Add an element *e* the the tail of the queue

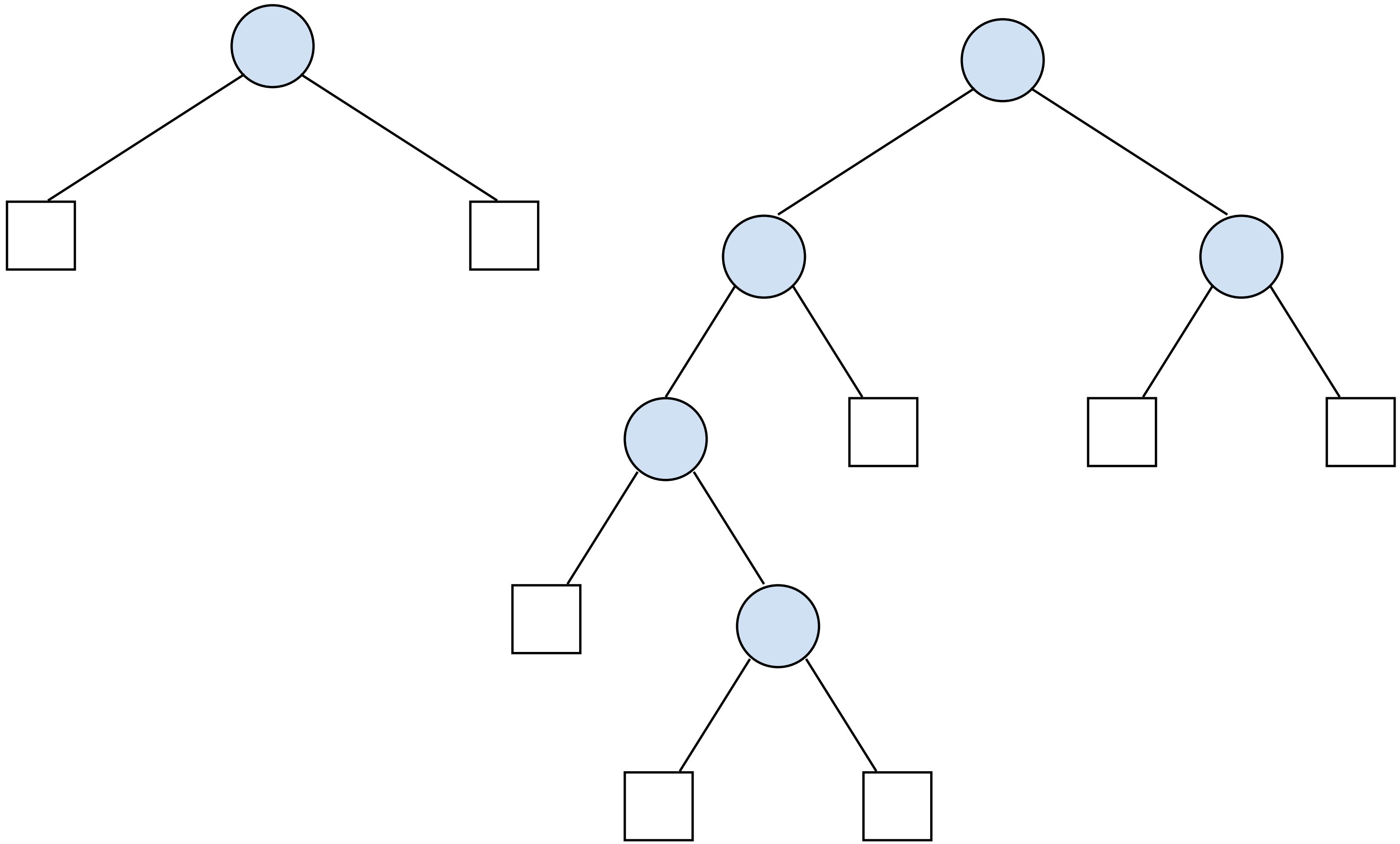


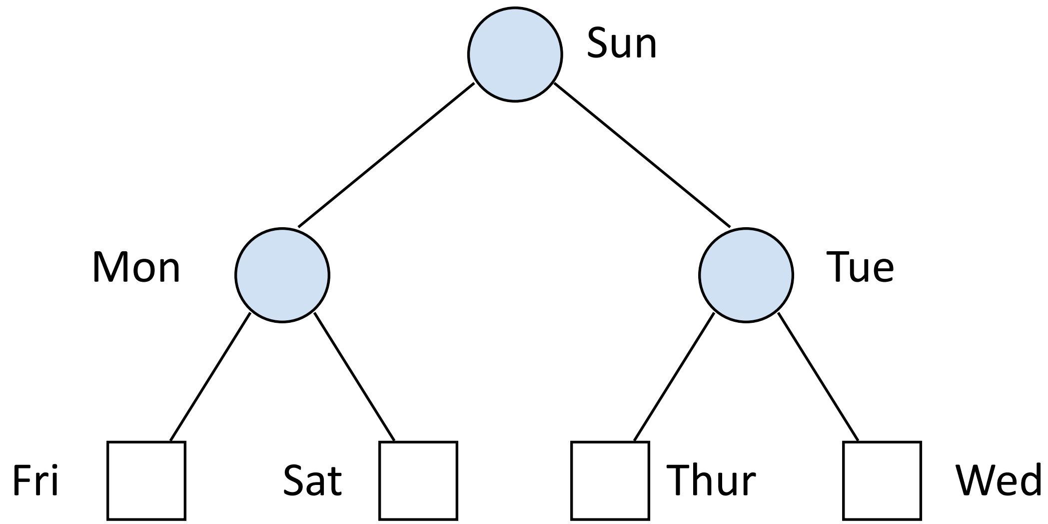
Algorithms and Data-Structures Topics

Trees

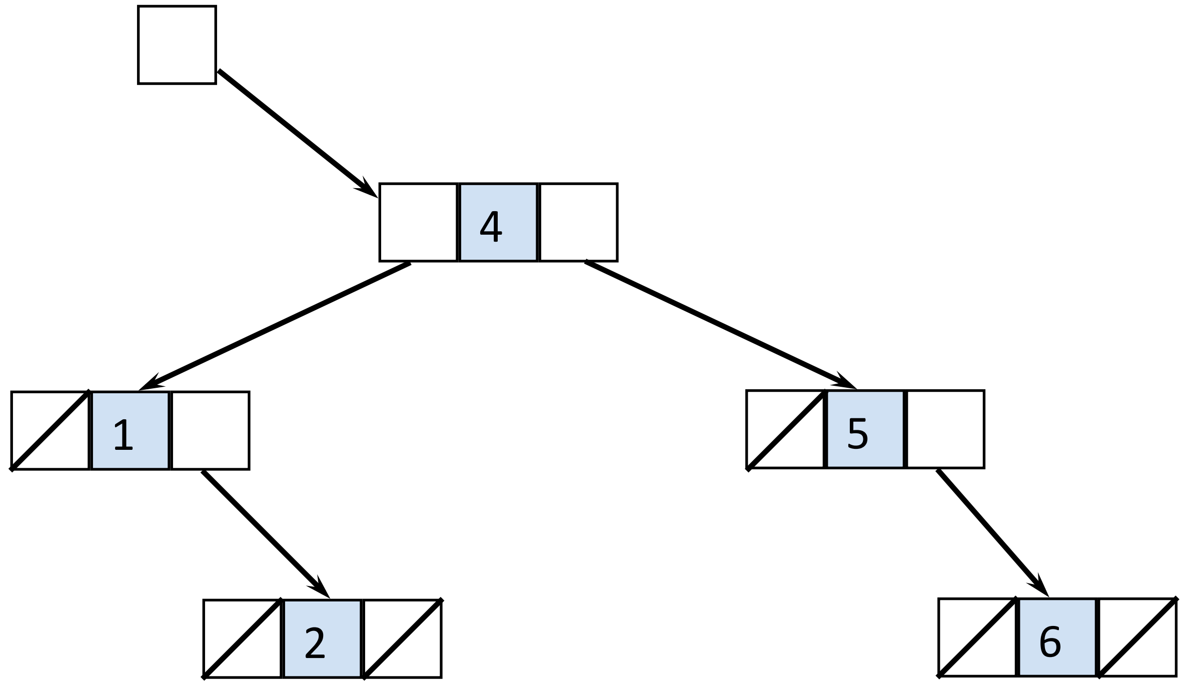
- Binary trees
- Binary search trees
- Depth-first traversal
- Breadth-first traversals
- Applications of trees
  (e.g. Huffman coding)
- Height-balanced trees
  (e.g. AVL Trees, Red-Black Trees)



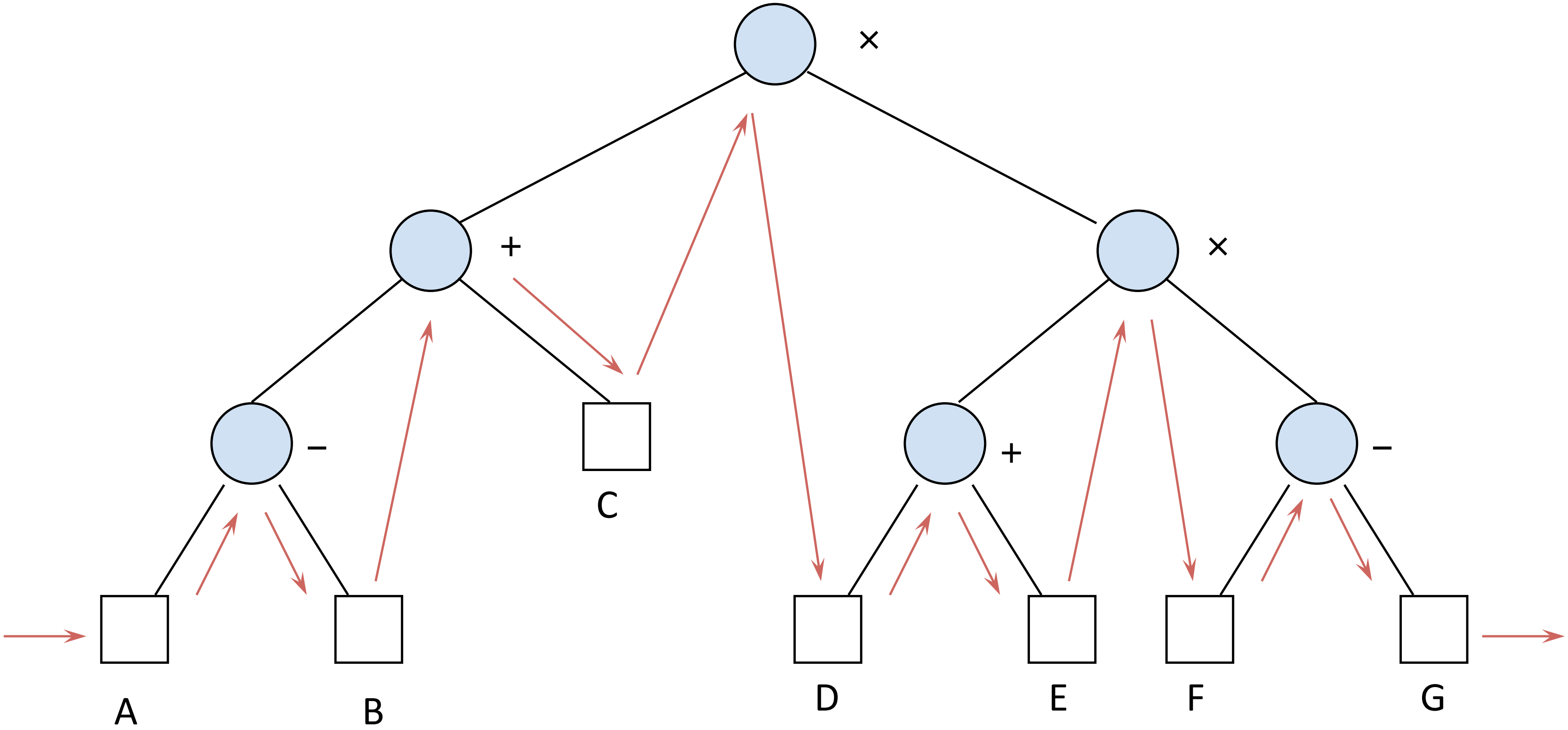




Binary Search Tree

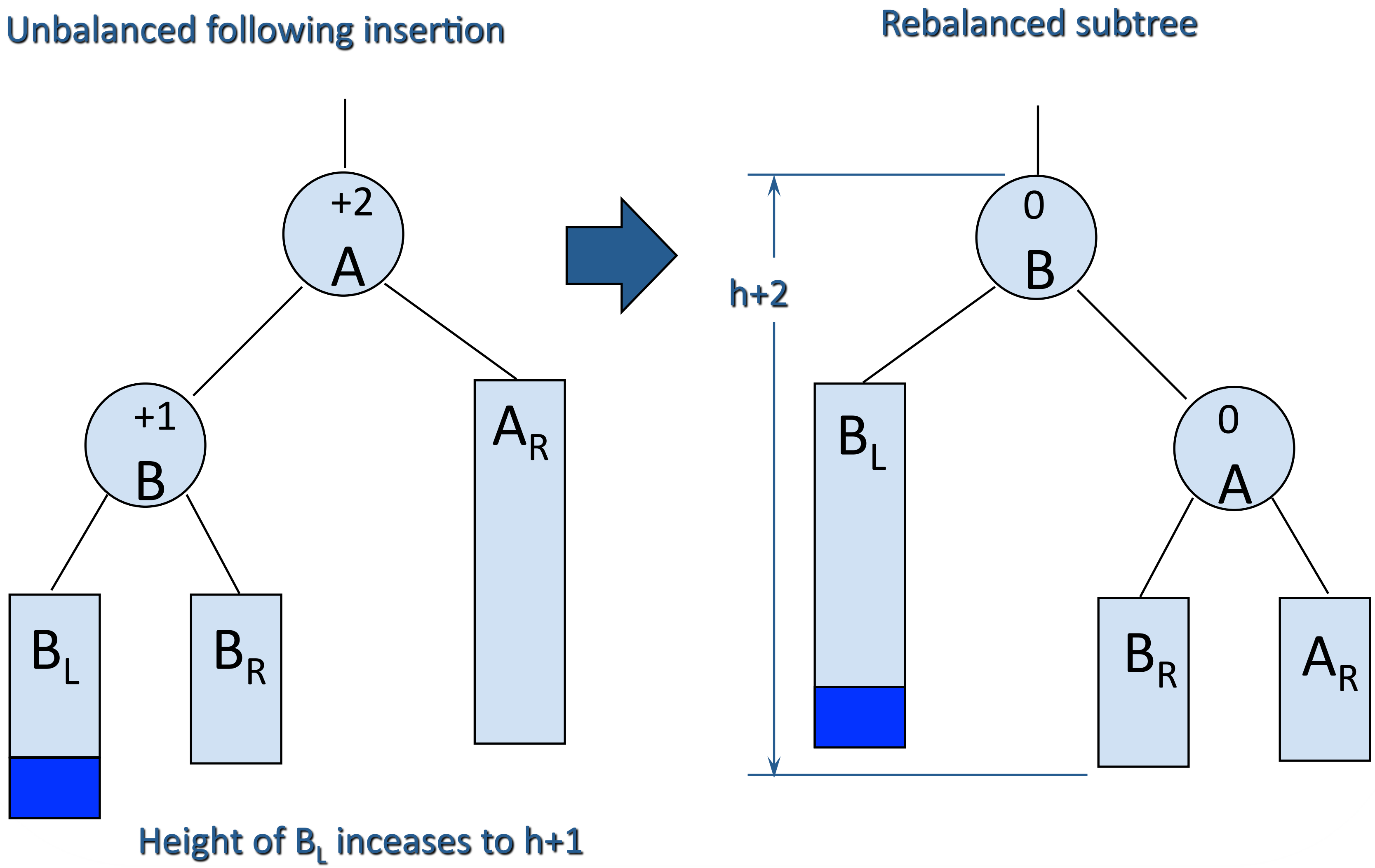


Pointer Implementation

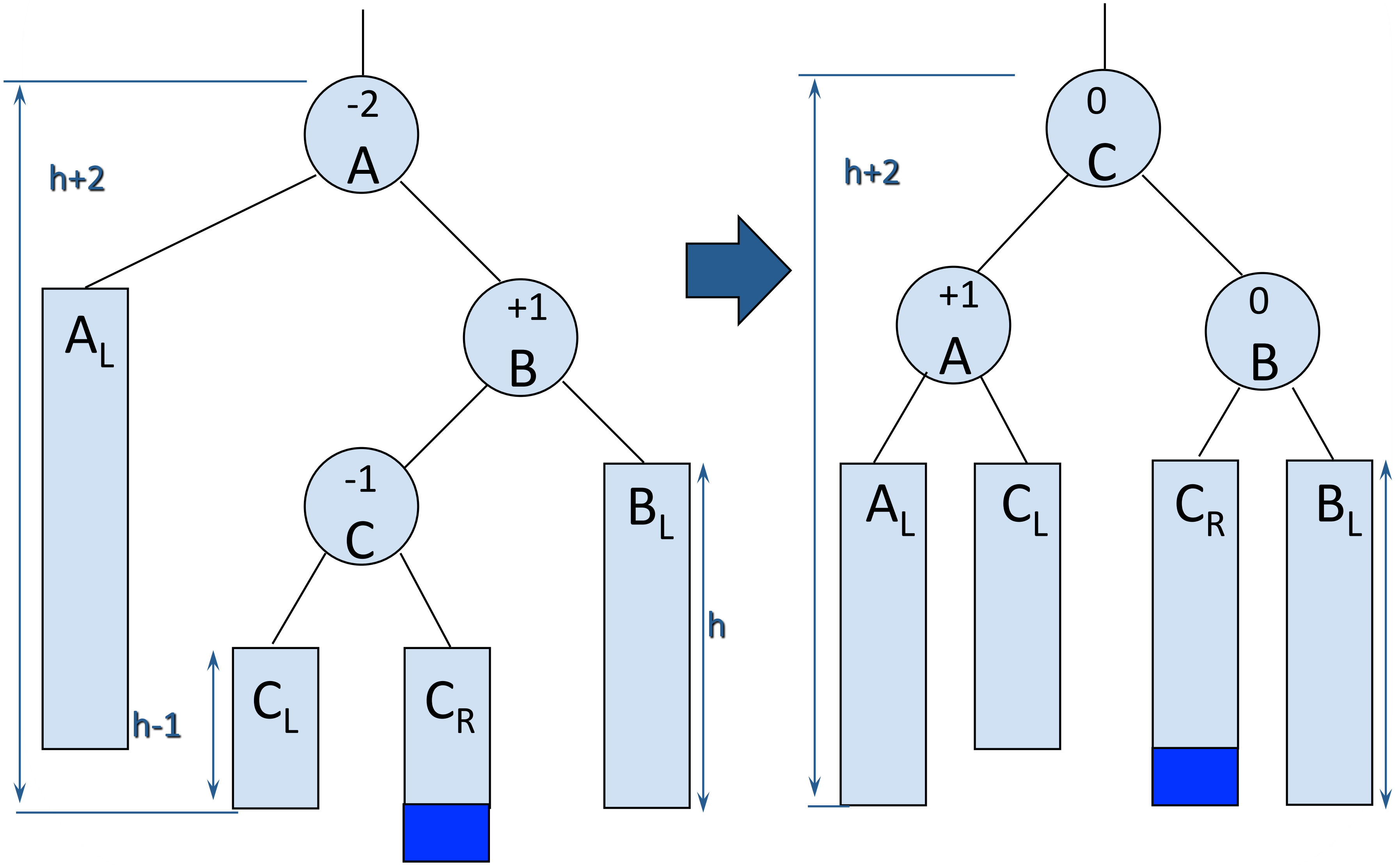


Inorder Traversal

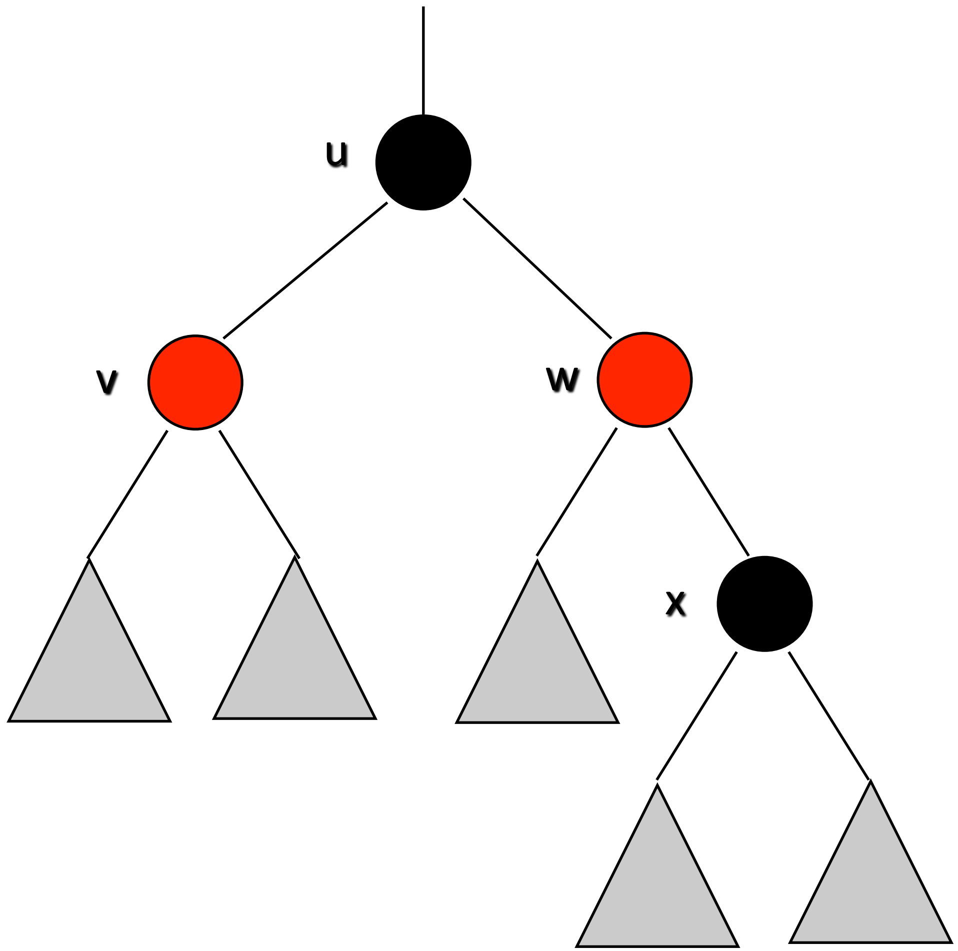
AVL Trees – Height Balancing



AVL Trees - RL rotation



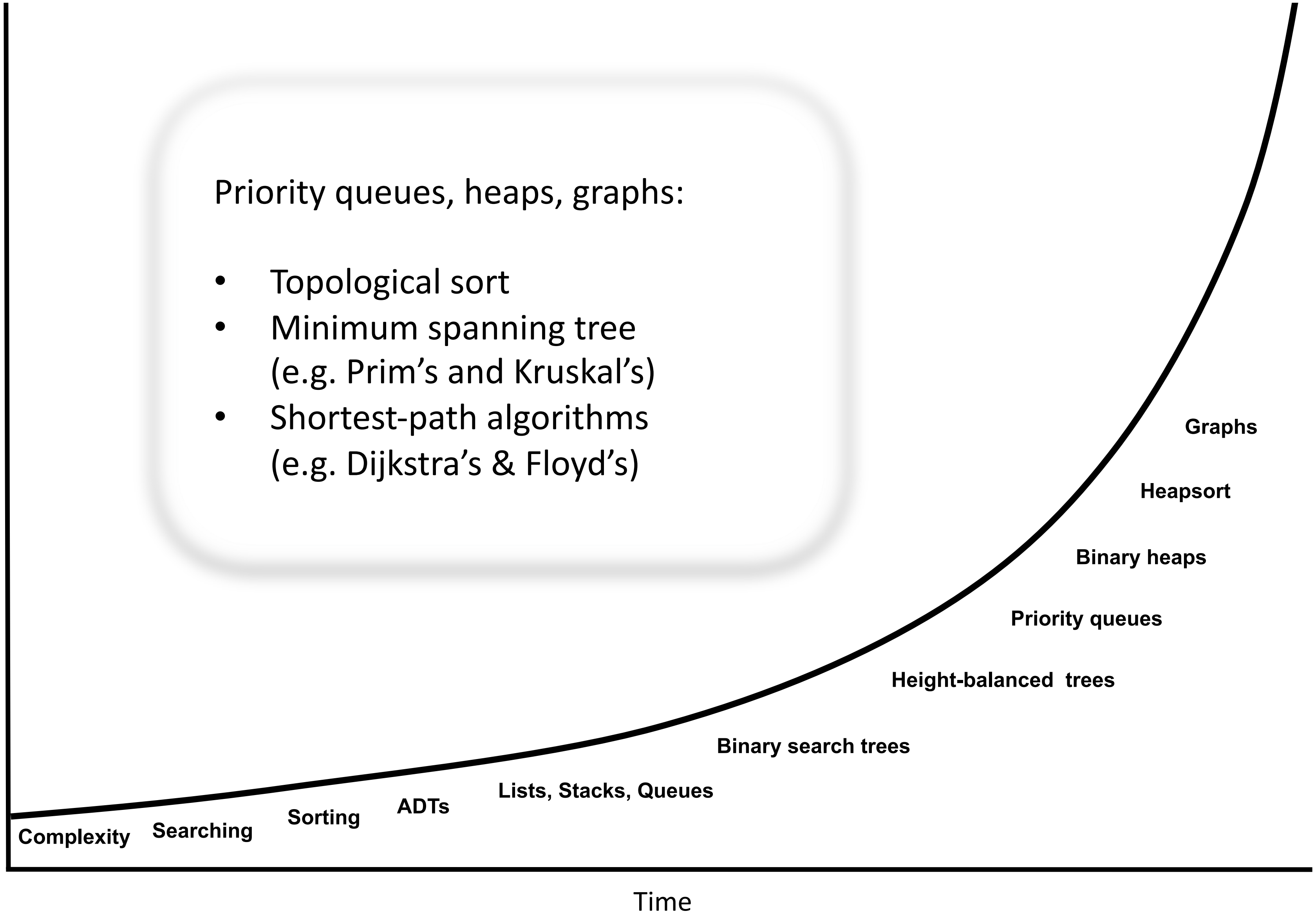
Red-Black Height-balanced Trees



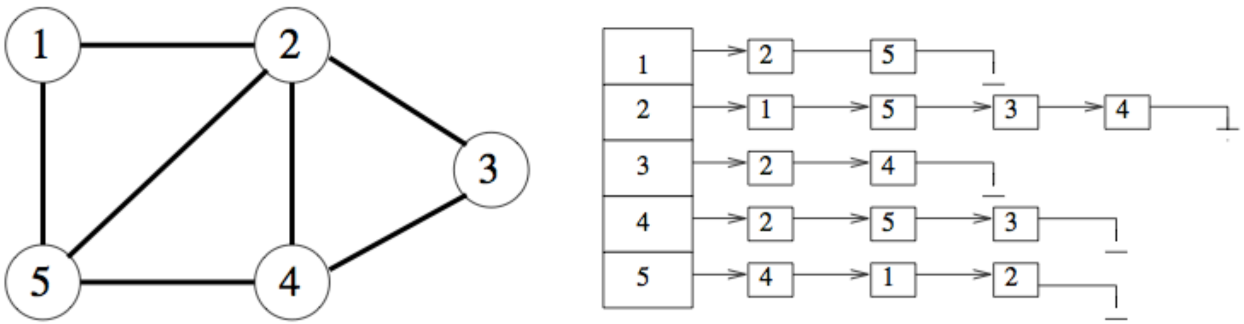
Algorithms and Data-Structures Topics

Priority queues, heaps, graphs:

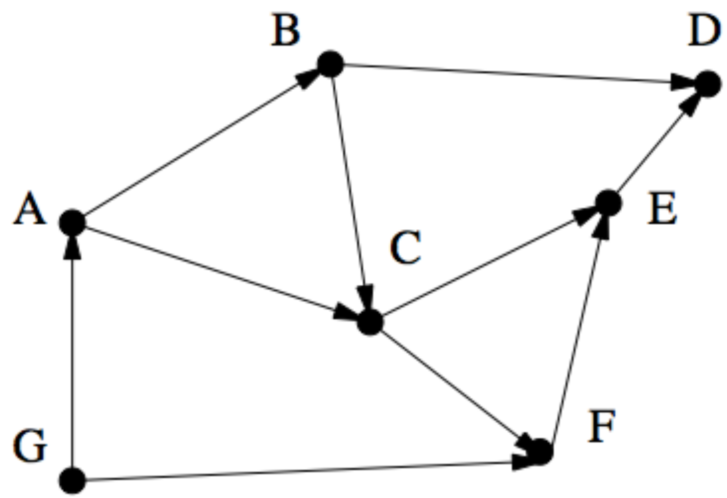
- Topological sort
- Minimum spanning tree
  (e.g. Prim's and Kruskal's)
- Shortest-path algorithms
  (e.g. Dijkstra's & Floyd's)



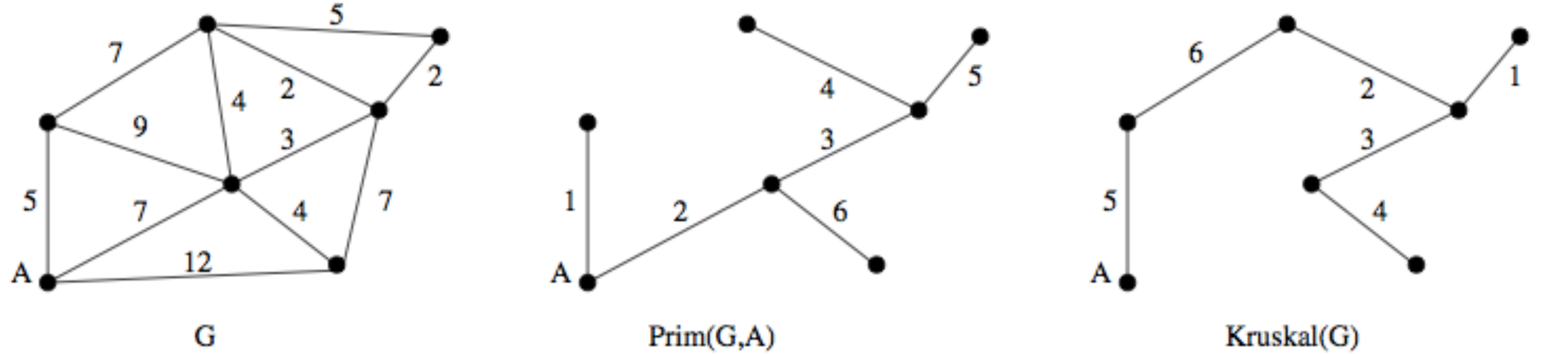
	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0



Adjacency matrix and list implementations of a graph

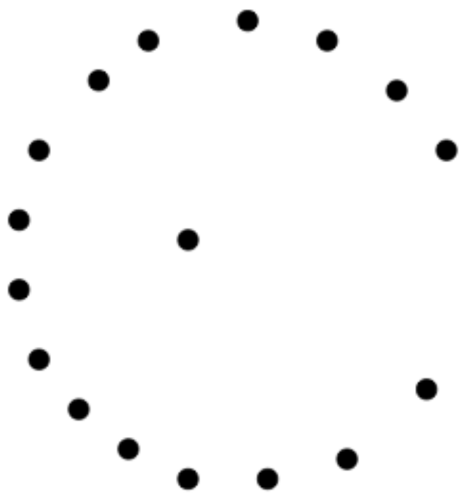


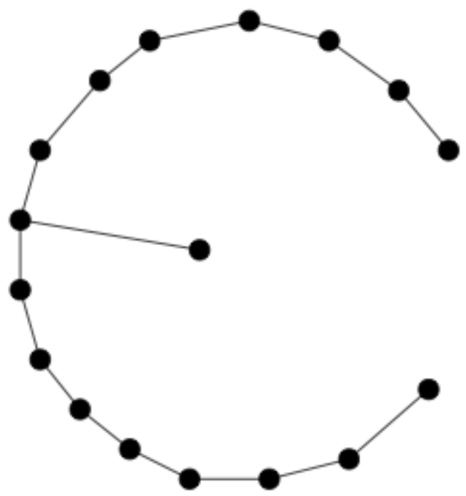
A DAG with only one topological sort (G, A, B, C, F, E, D)



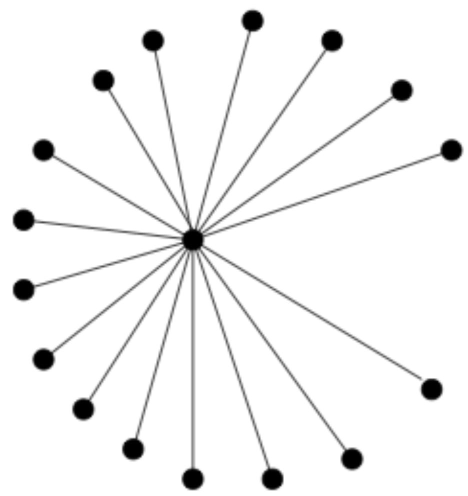
Minimum Spanning Trees

Tree formed from subset of graph edges that connects all vertices
and minimizes the total path length

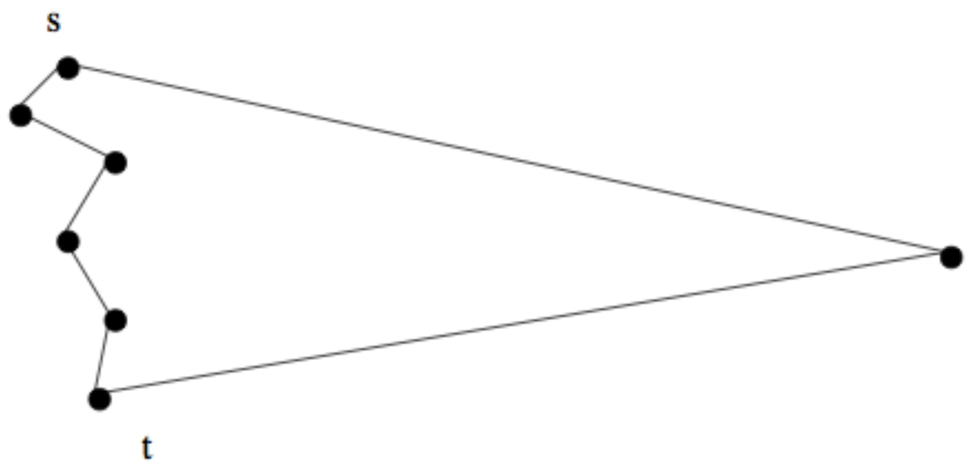




Minimum Spanning Tree

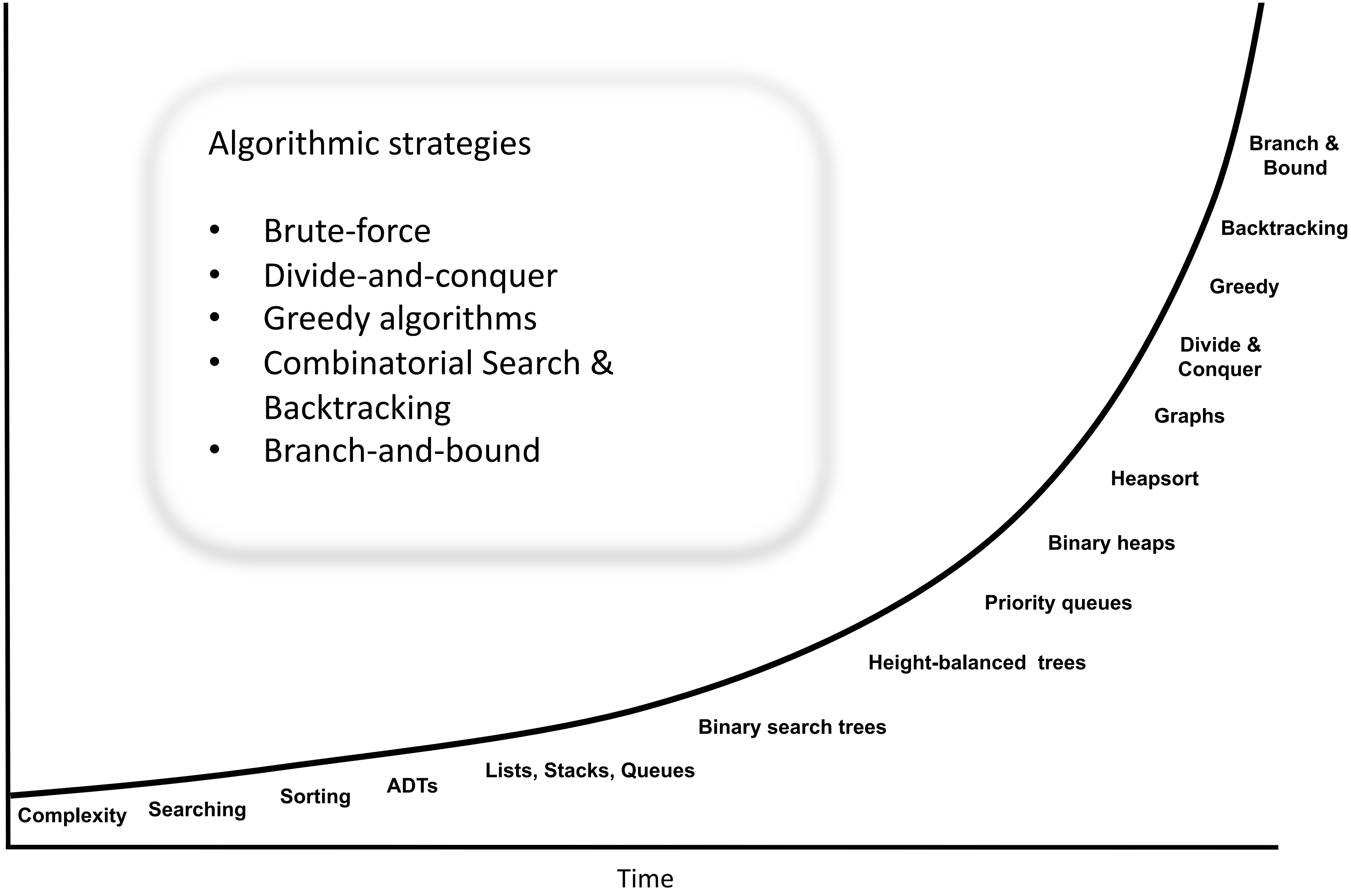


Shortest Path from Centre
Spanning Tree

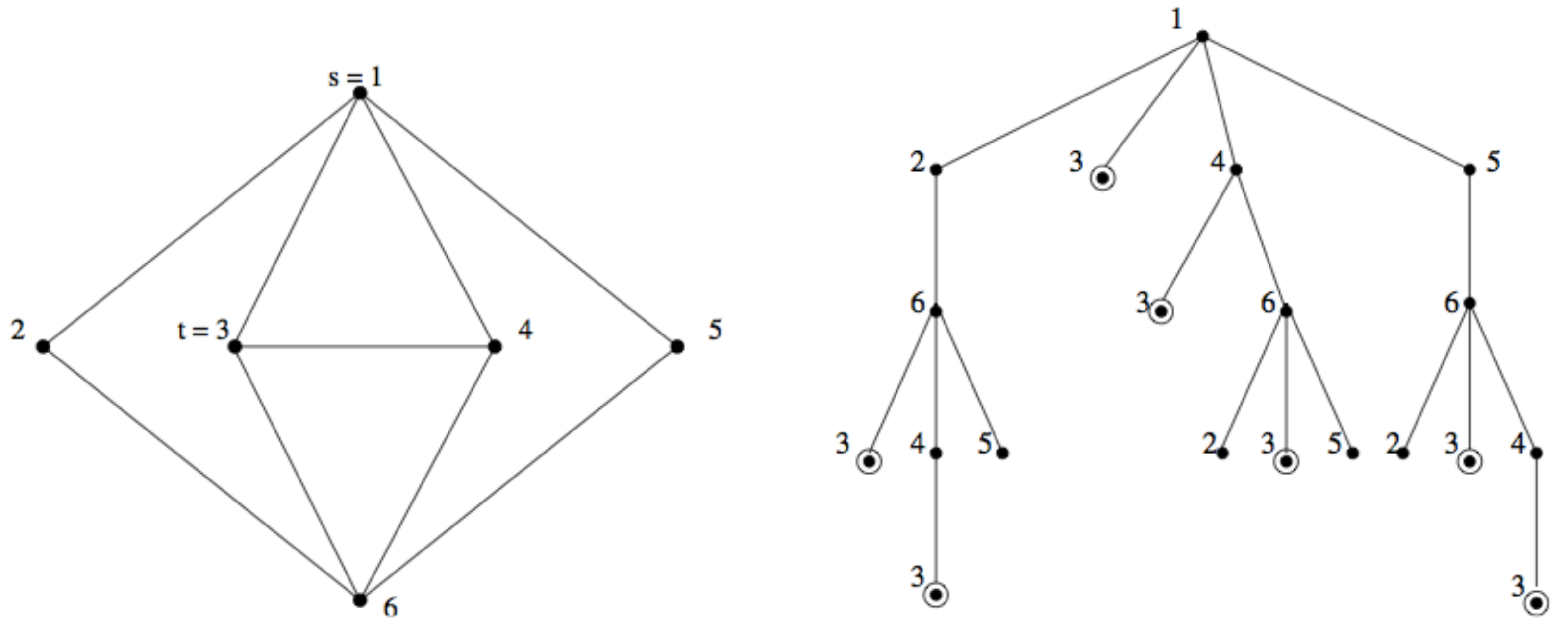


The shortest path from s to t may pass through many intermediate vertices

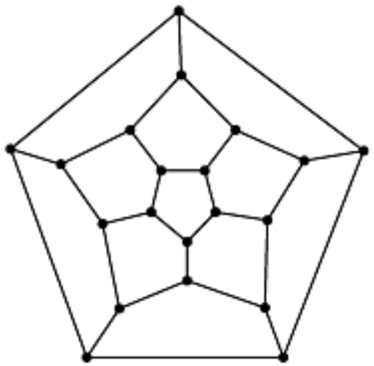
Algorithms and Data-Structures Topics



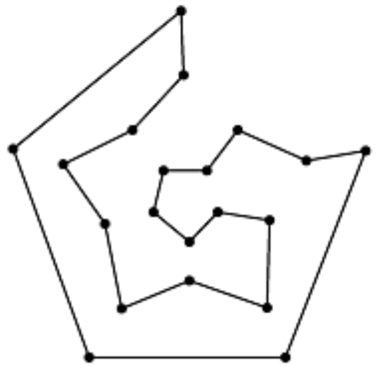
Combinatorial Search and Heuristic Methods



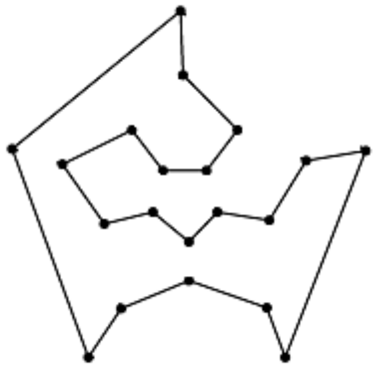
Search tree enumerating all simple s-t paths in the given graph (left).



Typical Input for HCP



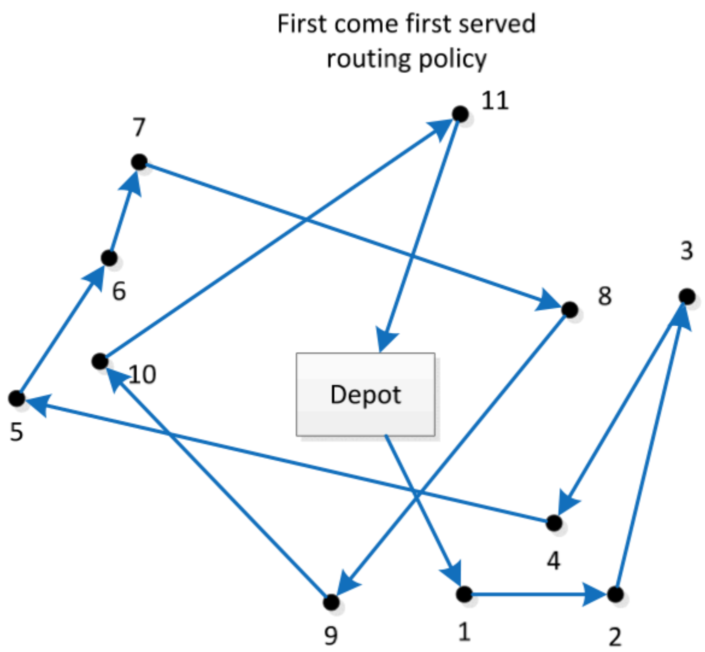
Hamiltonian cycle for the graph

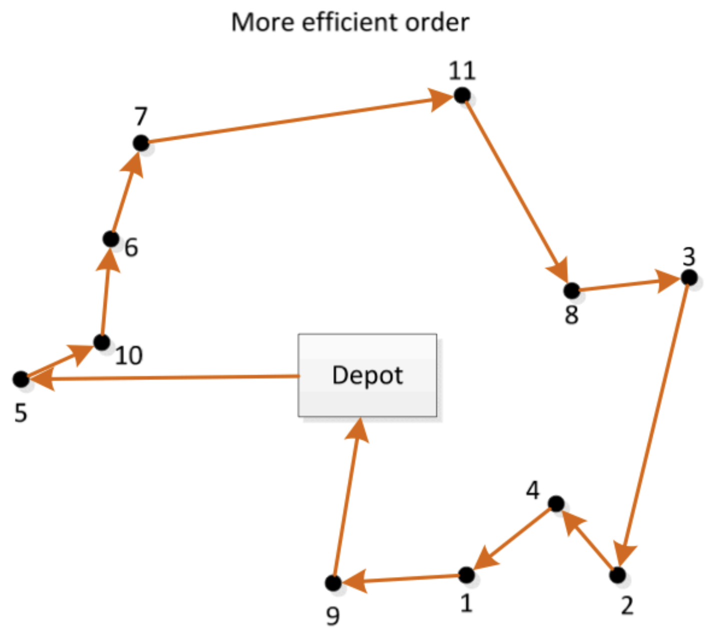


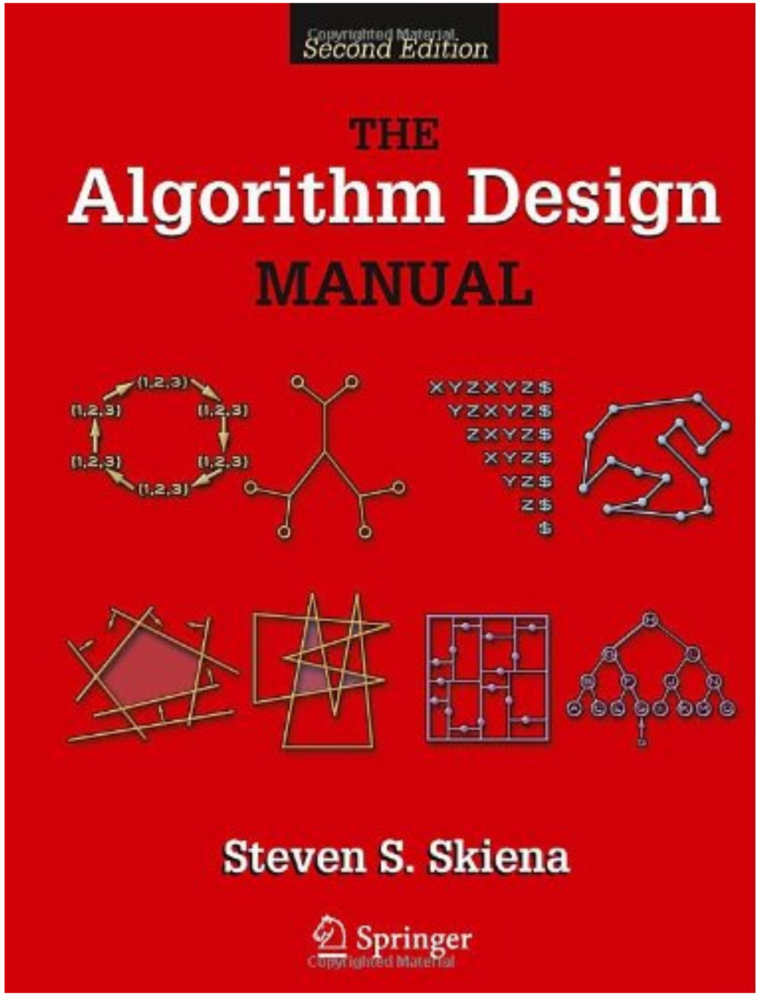
Another Hamiltonian cycle for the same graph in

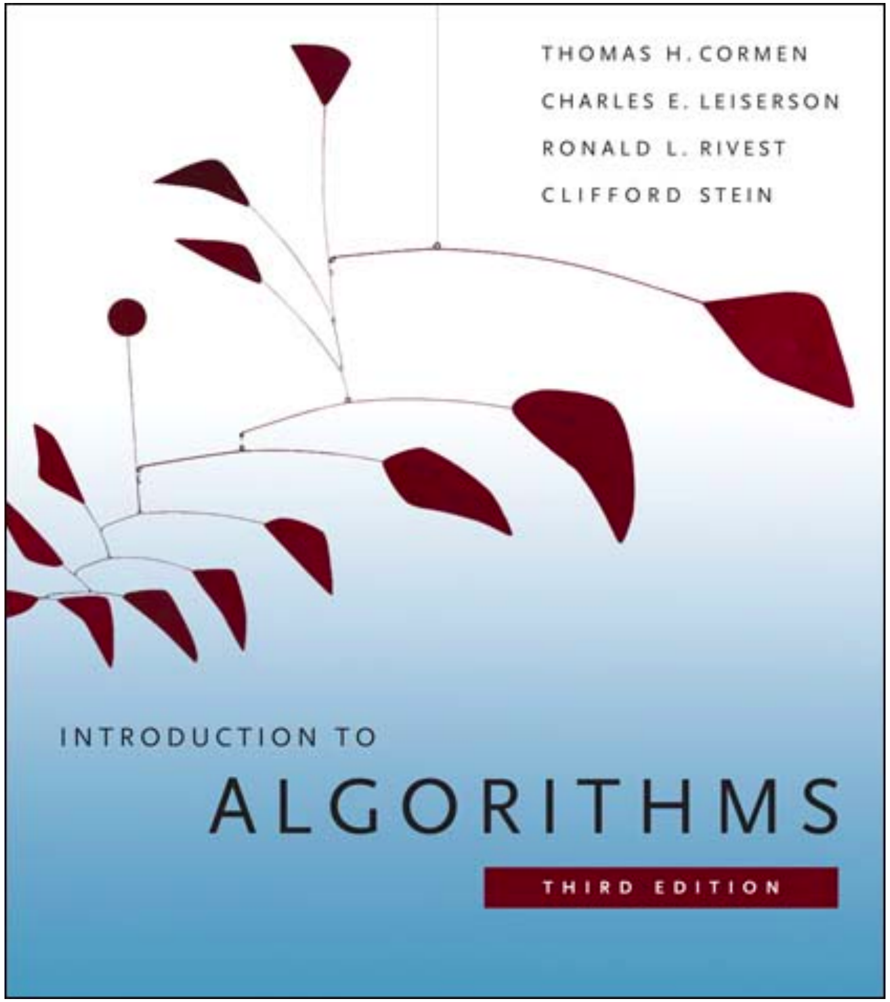
A Hamiltonian cycle for a given graph ***G=(V, E)*** consists on finding an ordering of the vertices of the graph ***G*** such that each vertex is visited exactly once

Travelling Salesman Problem … or, a variant, the vehicle routing problem

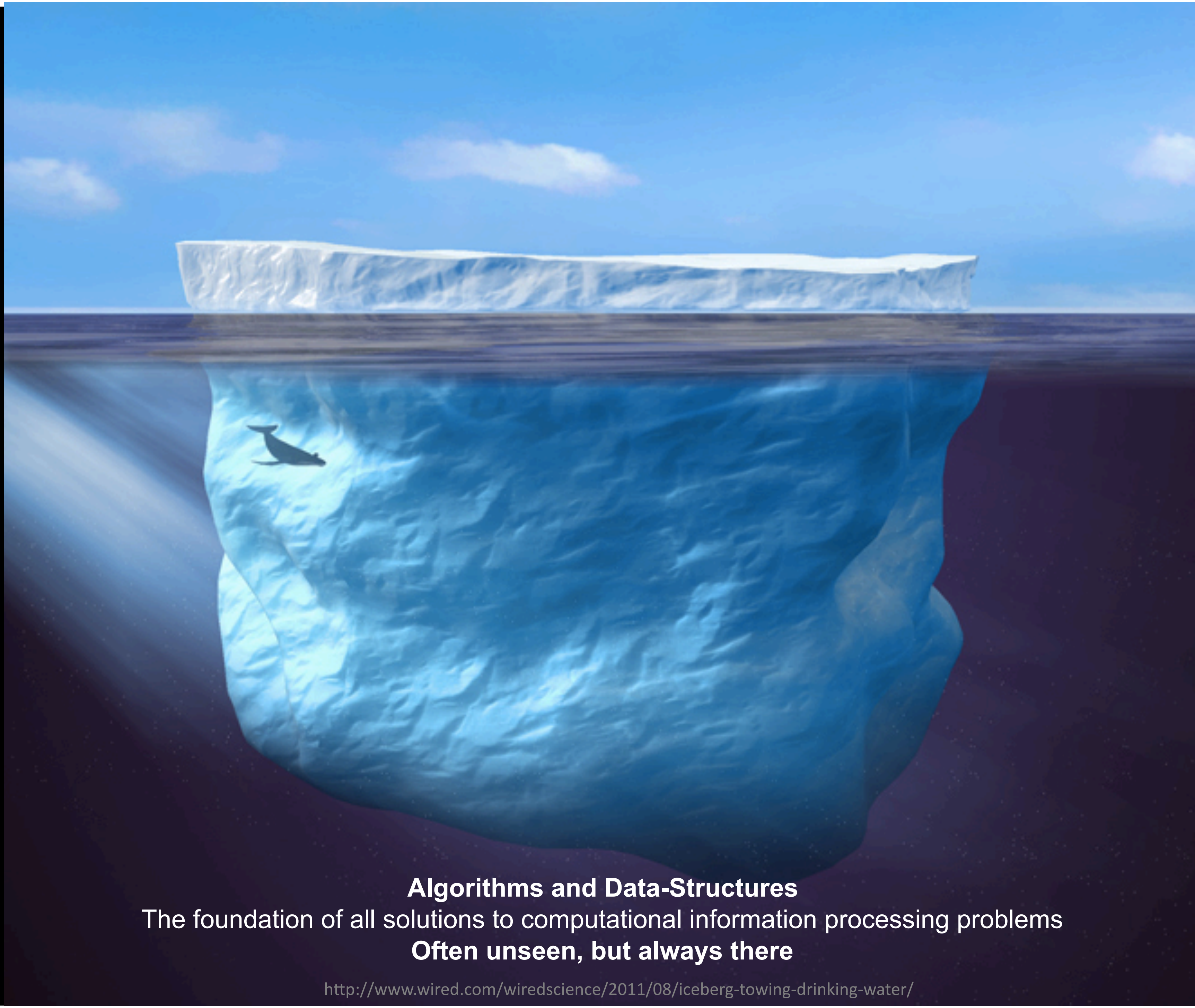




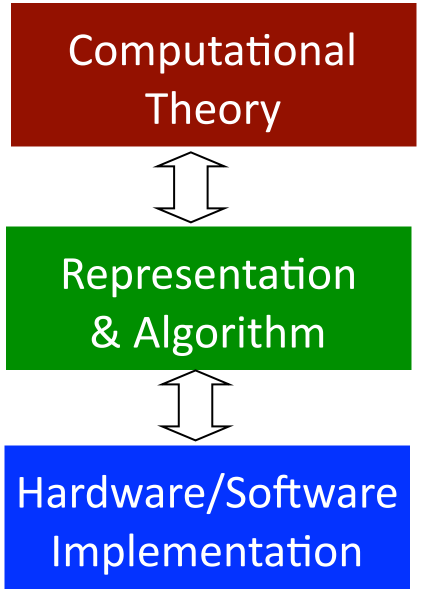




See `www.algorist.com`
for online resources


Algorithms and Data-Structures
The foundation of all solutions to computational information processing problems
Often unseen, but always there
http://www.wired.com/wiredscience/2011/08/iceberg-towing-drinking-water/

Marr's Hierarchy of Abstraction / Levels of Understanding Framework



Goal, logic, strategy, model

I/O representation, transformation algorithm

Physical realization