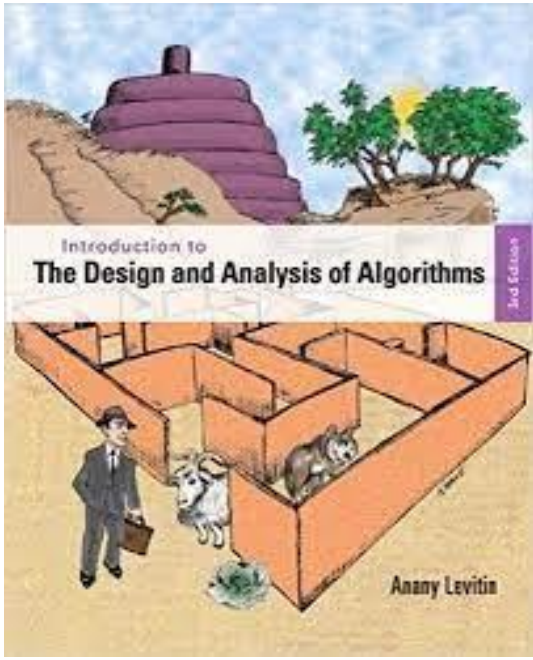
4-Decrease and Conquer



Decrease and Conquer

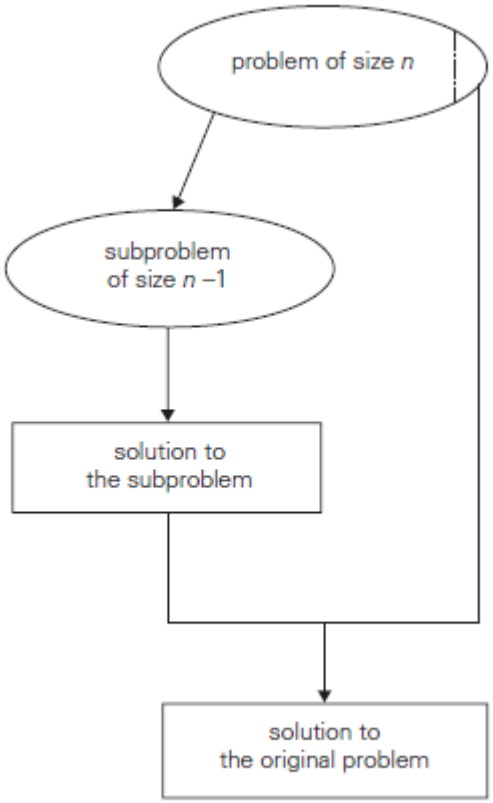
1. Reduce problem instance to smaller instance of the same problem
2. Solve smaller instance
3. Extend solution of smaller instance to obtain solution to original instance

- Can be implemented either top-down or bottom-up
- Also referred to as *inductive* or *incremental* approach

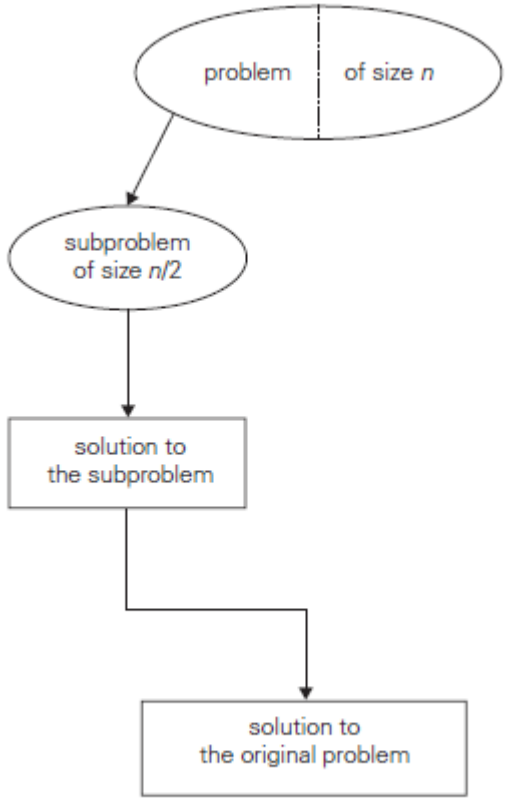
Decrease and Conquer

Three Types

- *Decrease by a constant* (usually by 1):
  - insertion sort
  - topological sorting
  - algorithms for generating permutations, subsets
- *Decrease by a constant factor* (usually by half)
  - binary search and bisection method
  - exponentiation by squaring
  - multiplication à la russe
- *Variable-size decrease*
  - Euclid's algorithm
  - selection by partition
  - Nim-like games



Decrease-(by one)-and-conquer technique.



Decrease-(by half)-and-conquer technique.

What's the difference?

Consider the problem of exponentiation: Compute a^n

- Brute Force:
- Divide and conquer:
- Decrease by one:
- Decrease by constant factor:

Insertion Sort

To sort array A[0..n-1], sort A[0..n-2] recursively and then insert A[n-1] in its proper place among the sorted A[0..n-2]

Usually implemented bottom up (nonrecursively)

Insertion Sort

Example: Sort 6, 4, 1, 8, 5

6 | 4 1 8 5
4 6 | 1 8 5
1 4 6 | 8 5
1 4 6 8 | 5
1 4 5 6 8

Pseudocode of Insertion Sort

ALGORITHM *InsertionSort*$(A[0..n-1])$

```
//Sorts a given array by insertion sort
//Input: An array A[0..n − 1] of n orderable elements
//Output: Array A[0..n − 1] sorted in nondecreasing order
for i ← 1 to n − 1 do
    v ← A[i]
    j ← i − 1
    while j ≥ 0 and A[j] > v do
        A[j + 1] ← A[j]
        j ← j − 1
    A[j + 1] ← v
```

Analysis of Insertion Sort

- Time efficiency

$$C_{worst}(n) = \frac{n(n-1)}{2} \in \Theta(n^2)$$

$$C_{avg}(n) \approx \frac{n^2}{4} \in \Theta(n^2)$$

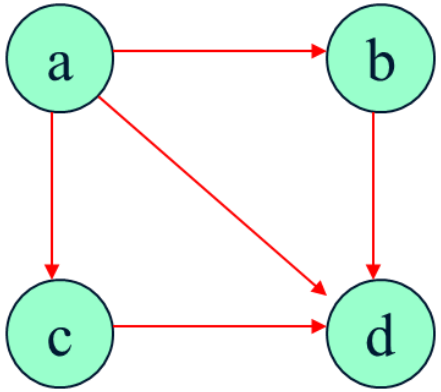
$$C_{best}(n) = n - 1 \in \Theta(n)$$ (also fast on almost sorted arrays)

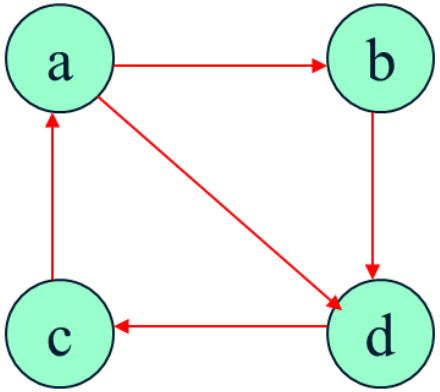
- Space efficiency: in-place
- Stability: yes
- Best elementary sorting algorithm overall
- Binary insertion sort

Dags and Topological Sorting

A *dag*: a directed acyclic graph, i.e. a directed graph with no (directed) cycles

a dag





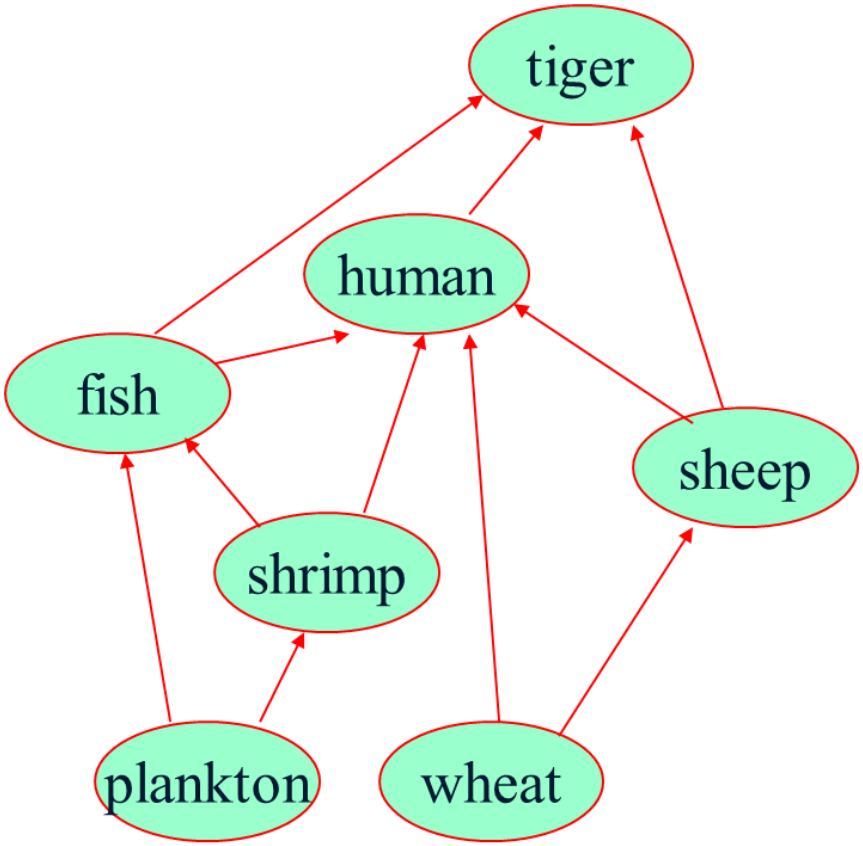
not a dag

Arise in modeling many problems that involve prerequisite constraints (construction projects, document version control)

Vertices of a dag can be linearly ordered so that for every edge its starting vertex is listed before its ending vertex (*topological sorting*). Being a dag is also a necessary condition for topological sorting be possible.

Topological Sorting Example

Order the following items in a food chain



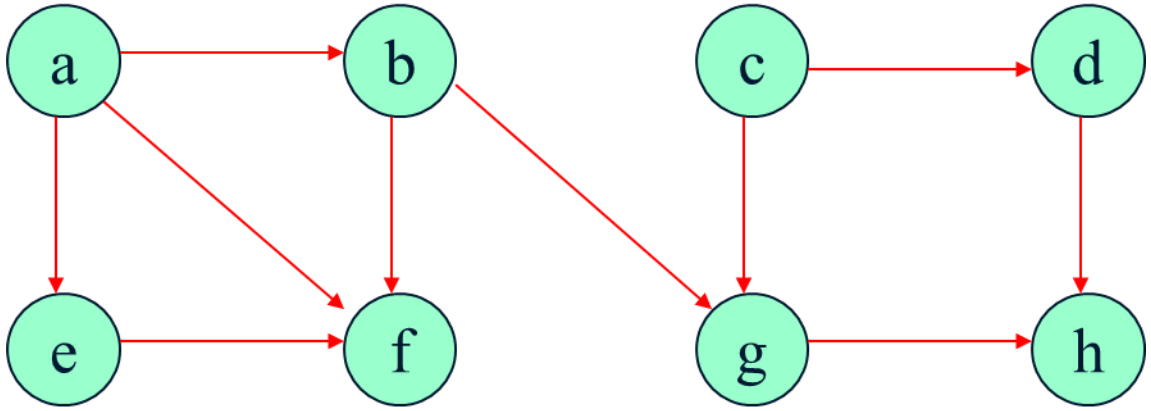
DFS-based Algorithm

DFS-based algorithm for topological sorting

- Perform DFS traversal, noting the order vertices are popped off the traversal stack
- Reverse order solves topological sorting problem
- Back edges encountered?→ NOT a dag!

DFS-based Algorithm

Example



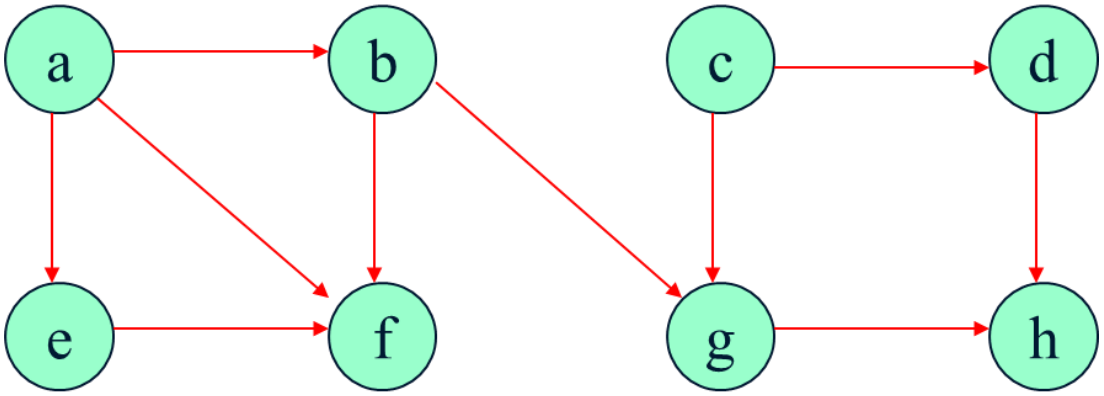
Efficiency:

Source Removal Algorithm

Source removal algorithm

Repeatedly identify and remove a *source* (a vertex with no incoming edges) and all the edges incident to it until either no vertex is left (problem is solved) or there is no source among remaining vertices (not a dag)

Example:



Efficiency: same as efficiency of the DFS-based algorithm

Generating Permutations

Minimal-change decrease-by-one algorithm

If n = 1 return 1; otherwise, generate recursively the list of all permutations of 12…n-1 and then insert n into each of those permutations by starting with inserting n into 12...n-1 by moving right to left and then switching direction for each new permutation

Generating Permutations

Example: n=3

start	1		
insert 2 into 1 right to left	12	21	
insert 3 into 12 right to left	123	132	312
insert 3 into 21 left to right	321	231	213

Other permutation generating algorithms

- Johnson-Trotter (p. 145)
- Lexicographic-order algorithm (p. 146)
- Heap's algorithm (Problem 4 in Exercises 4.3)

Generating Subsets

Binary reflected Gray code: minimal-change algorithm for generating 2^n bit strings corresponding to all the subsets of an n-element set where $n > 0$

If n=1 make list L of two bit strings 0 and 1
else
    generate recursively list $L1$ of bit strings of length n-1
    copy list $L1$ in reverse order to get list $L2$
    add 0 in front of each bit string in list $L1$
    add 1 in front of each bit string in list $L2$
    append $L2$ to $L1$ to get L
return L

Decrease-by-Constant-Factor Algorithms

In this variation of decrease-and-conquer, instance size is reduced by the same factor (typically, 2)

Examples:

- binary search and the method of bisection
- exponentiation by squaring
- multiplication à la russe (Russian peasant method)
- fake-coin puzzle
- Josephus problem

Binary Search

Very efficient algorithm for searching in sorted array:

K

vs

A[0] . . . A[m] . . . A[n-1]

If K = A[m], stop (successful search); otherwise, continue searching by the same method in A[0..m-1] if K < A[m] and in A[m+1..n-1] if K > A[m]

$l \leftarrow 0$; $r \leftarrow n-1$
while $l \leq r$ do
    $m \leftarrow \lfloor (l+r)/2 \rfloor$
    if $K = A[m]$ return m
    else if $K < A[m]$ $r \leftarrow m-1$
    else $l \leftarrow m+1$
return -1

Analysis of Binary Search

- Time efficiency
- worst-case recurrence: $C_w(n) = 1 + C_w(\lfloor \frac{n}{2} \rfloor),\ C_w(1) = 1$
  solution: $C_w(n) = \lceil \log_2(n+1) \rceil$

  This is VERY fast: e.g., $C_w(10^6) = 20$

- Optimal for searching a sorted array
- Limitations: must be a sorted array (not linked list)
- Bad (degenerate) example of divide-and-conquer
- Has a continuous counterpart called bisection method for solving equations in one unknown $f(x) = 0$

Exponentiation by Squaring

The problem: compute a^n where n is nonnegative integer

The problem can be solved by applying recursively the formulas:

For even values of n

$$a^n = \left(a^{\frac{n}{2}}\right)^2 \quad if\ n > 0\ and\ a^0 = 1$$

For odd values of n

$$a^n = \left(a^{\frac{n-1}{2}}\right)^2 a$$

Recurrence: $M(n) = M\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + f(n)\ \ where\ \ f(n) = 1\ or\ 2,$

$$M(0) = 0$$

Master theorem: $M(n) \in \Theta(logn) = \Theta(b)\ \ where\ \ b = \lceil \log_2(n+1) \rceil$

Russian Peasant Multiplication

The problem: Compute the product of two positive integers

Can be solved by a decrease-by-half algorithm based on the following formulas.

For even values of n

$$n * m = \frac{n}{2} * 2m$$

For odd values of n

$$n * m = \frac{n-1}{2} * 2m + m \quad if\ n > 1\ and\ m\ if\ n = 1$$

Example of Russian Peasant Multiplication

Compute 20 * 26

n	m	
20	26	
10	52	
5	104	104
2	208	+
1	416	416
		520

Note: Method reduces to adding *m*'s values corresponding to odd *n*'s.

Fake-Coin Puzzle (simpler version)

There are *n* identically looking coins one of which is fake. There is a balance scale but there are no weights; the scale can tell whether two sets of coins weigh the same and, if not, which of the two sets is heavier (but not by how much). Design an efficient algorithm for detecting the fake coin. Assume that the fake coin is known to be lighter than the genuine ones.

Decrease by factor 2 algorithm

Decrease by factor 3 algorithm

Variable-Size-Decrease Algorithms

In the variable-size-decrease variation of decrease-and-conquer, instance size reduction varies from one iteration to another

Examples:

- Euclid's algorithm for greatest common divisor
- partition-based algorithm for selection problem
- interpolation search
- some algorithms on binary search trees
- Nim and Nim-like games

Euclid's Algorithm

Euclid's algorithm is based on repeated application of equality

$$\gcd(m, n) = \gcd(n, m \bmod n)$$

Ex.:

gcd(80,44) = gcd(44,36) = gcd(36, 12) = gcd(12,0) = 12

One can prove that the size, measured by the second number, decreases at least by half after two consecutive iterations.

Hence, $T(n) \in O(\log n)$

Selection Problem

Find the *k*-th smallest element in a list of *n* numbers

- $k = 1$ or $k = n$

- *median*: $k = \lceil n/2 \rceil$

  Example: 4, 1, 10, 9, 7, 12, 8, 2, 15 median = ?

The median is used in statistics as a measure of an average value of a sample. In fact, it is a better (more robust) indicator than the mean, which is used for the same purpose.

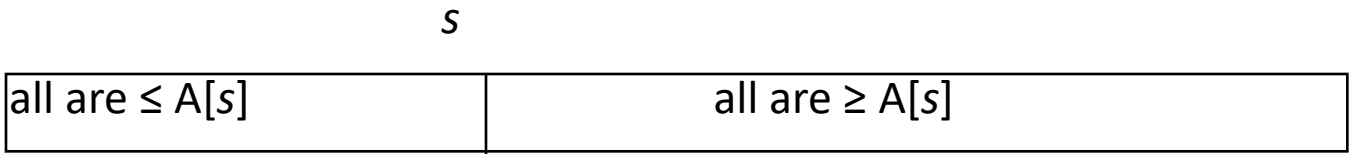
Digression: Post Office Location Problem

Given *n* village locations along a straight highway, where should a new post office be located to minimize the average distance from the villages to the post office?

Algorithms for the Selection Problem

The sorting-based algorithm: Sort and return the *k*-th element
Efficiency (if sorted by mergesort): Θ(*n*log *n*)

A faster algorithm is based on the array *partitioning*:



Assuming that the array is indexed from 0 to *n*-1 and *s* is a split position obtained by the array partitioning:

If *s* = *k*-1, the problem is solved;

if *s* > *k*-1, look for the *k*-th smallest element in the left part;
if *s* < *k*-1, look for the (*k*-*s*)-th smallest element in the right part.

Note: The algorithm can simply continue until *s* = *k*-1.

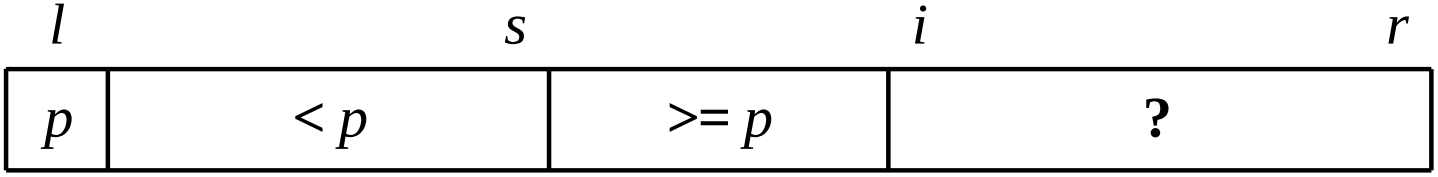
Two Partitioning Algorithms

There are two principal ways to partition an array:

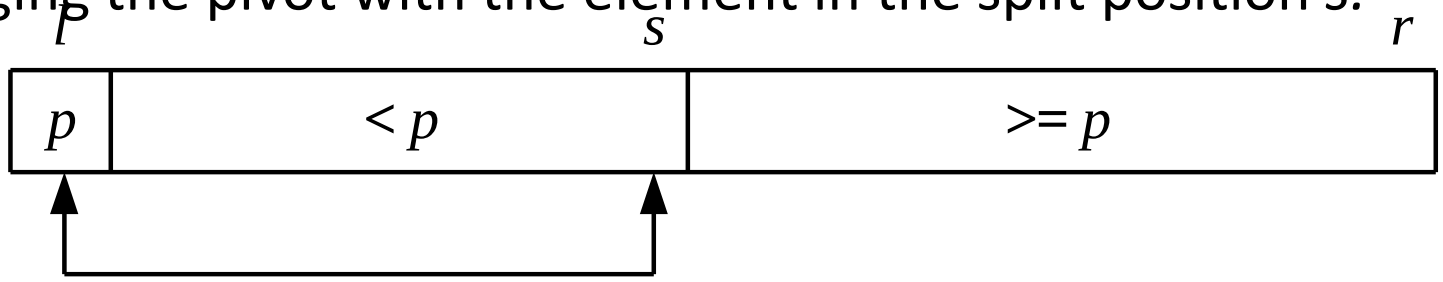
- One-directional scan (Lomuto's partitioning algorithm)
- Two-directional scan (Hoare's partitioning algorithm)

Lomuto's Partitioning Algorithm

Scans the array left to right maintaining the array's partition into three contiguous sections: $< p$, $\geq p$, and unknown, where p is the value of the first element (the partition's *pivot*).



On each iteration the unknown section is decreased by one element until it's empty and a partition is achieved by exchanging the pivot with the element in the split position *s*.



Tracing Lomuto's Partioning Algorithm

s	*i*							
4	1	10	8	7	12	9	2	15
	s	*i*						
4	1	10	8	7	12	9	2	15
	s						*i*	
4	1	10	8	7	12	9	2	15
		s						*i*
4	1	2	8	7	12	9	10	15
		s						
4	1	2	8	7	12	9	10	15
2	1	**4**	8	7	12	9	10	15

Tracing Quickselect (Partition-based Algorithm)

Find the median of 4, 1, 10, 9, 7, 12, 8, 2, 15

Here: $n = 9$, $k = \lceil 9/2 \rceil = 5$, $k - 1 = 4$

after 1st partitioning: $s = 2 < k - 1 = 4$

after 2nd partitioning: $s = 4 = k - 1$

0	1	2	3	**4**	5	6	7	8
4	1	10	8	7	12	9	2	15
2	1	**4**	8	7	12	9	10	15
			8	7	12	9	10	15
			7	**8**	12	9	10	15

The median is A[4]= 8

Efficiency of Quickselect

Average case (average split in the middle):

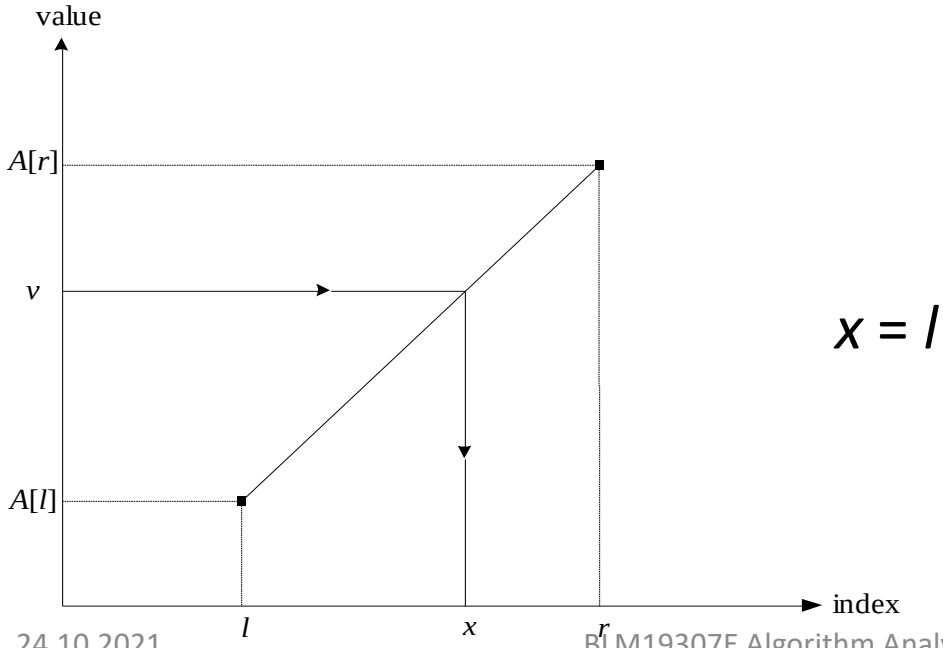
$$C(n) = C(n/2) + (n+1) \qquad C(n) \in \Theta(n)$$

Worst case (degenerate split): $C(n) \in \Theta(n^2)$

A more sophisticated choice of the pivot leads to a complicated algorithm with $\Theta(n)$ worst-case efficiency.

Interpolation Search

Searches a sorted array similar to binary search but estimates location of the search key in $A[l..r]$ by using its value v. Specifically, the values of the array's elements are assumed to grow linearly from $A[l]$ to $A[r]$ and the location of v is estimated as the x-coordinate of the point on the straight line through (l, A[l]) and (r, A[r]) whose y-coordinate is v:



$$x = l + \lfloor (v - A[l])(r - l)/(A[r] - A[l]) \rfloor$$

Analysis of Interpolation Search

- Efficiency

  average case: $C(n) < \log_2 \log_2 n + 1$

  worst case: $C(n) = n$

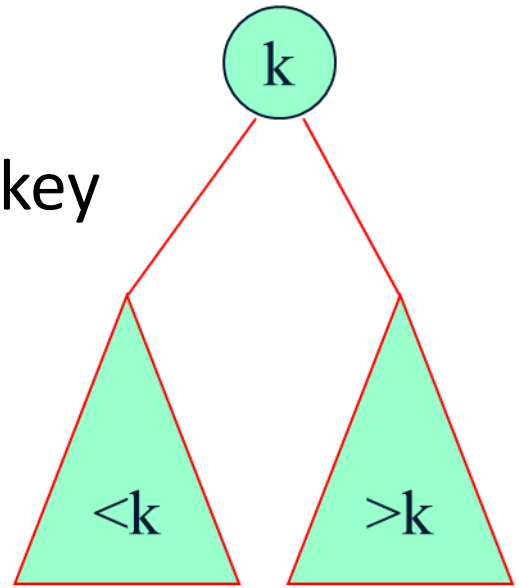
- Preferable to binary search only for VERY large arrays and/or expensive comparisons

- Has a counterpart, the *method of false position* (*regula falsi*), for solving equations in one unknown

Binary Search Tree Algorithms

Several algorithms on BST requires recursive processing of just one of its subtrees, e.g.,

- Searching
- Insertion of a new key
- Finding the smallest (or the largest) key



Searching in Binary Search Tree

Algorithm BTS(x, v)

//Searches for node with key equal to v in BST rooted at node x

```
if x = NIL  return -1
else if  v = K(x)  return x
else if  v < K(x)  return BTS(left(x), v)
else return BTS(right(x), v)
```

Efficiency

worst case: $C(n) = n$

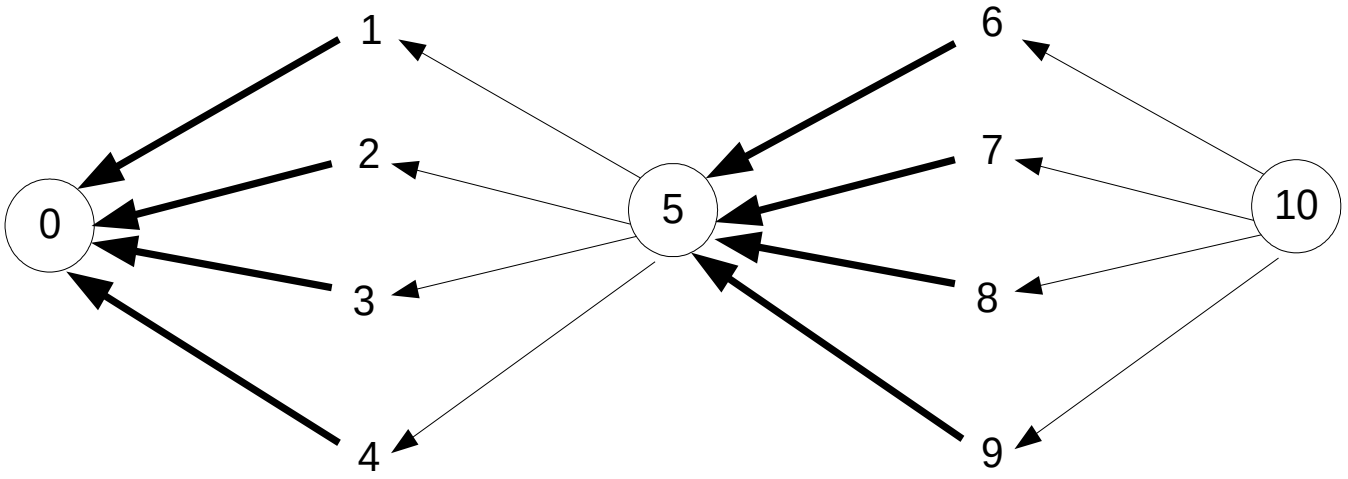
average case: $C(n) \approx 2\ln n \approx 1.39\log_2 n$

One-Pile Nim

There is a pile of *n* chips. Two players take turn by removing from the pile at least 1 and at most *m* chips. (The number of chips taken can vary from move to move.) The winner is the player that takes the last chip. Who wins the game – the player moving first or second, if both player make the best moves possible?

It's a good idea to analyze this and similar games "backwards", i.e., starting with $n = 0, 1, 2, \ldots$

Partial Graph of One-Pile Nim with *m* = 4



Vertex numbers indicate *n*, the number of chips in the pile. The losing position for the player to move are circled. Only winning moves from a winning position are shown (in bold).

Generalization: The player moving first wins iff *n* is not a multiple of 5 (more generally, *m*+1); the winning move is to take *n* mod 5 (*n* mod (*m*+1)) chips on every move.