

- Decrease and Conquer:

Three variations

→ Decrease by a constant

$$f(n) = \begin{cases} f(n-1) \cdot a & \text{if } n > 0, \\ 1 & \text{if } n = 0, \end{cases}$$

→ Decrease by a constant Factor:

ex. decreases by a factor of two.

$$a^n = \begin{cases} (a^{n/2})^2 & \text{if } n \text{ is even and positive,} \\ (a^{(n-1)/2})^2 \cdot a & \text{if } n \text{ is odd,} \\ 1 & \text{if } n = 0. \end{cases}$$

→ Variable Size decrease:

ex: Euclids algorithm

• Straight insertion sort:

Recursive →

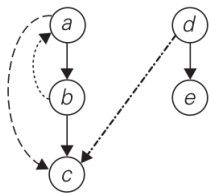
Bottom-up
Iterative approach

```
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
```

$A[0] \leq \dots \leq A[j] < A[j+1] \leq \dots \leq A[i-1] \mid A[i] \dots A[n-1]$
smaller than or equal to A[i] greater than A[i]

→ more
efficient!

• Direct graph:



Tree edges:

Back edges:

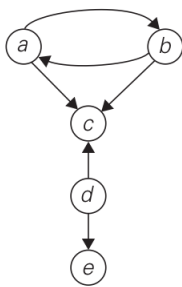
Forward edges:

Cross edges

Day:

(Directed acyclic graph)

(b)



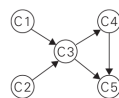
Directed cycle:

(a b a)

(a)

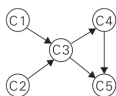
* IF a digraph has no directed cycles, the topological Sorting problem for it has a solution.

* IF the DFS finds a back edge, then the topological Sort for the vertices is impossible.



→ DFS for topological sort.

FIGURE 4.6 Digraph representing the prerequisite structure of five courses.



(a)

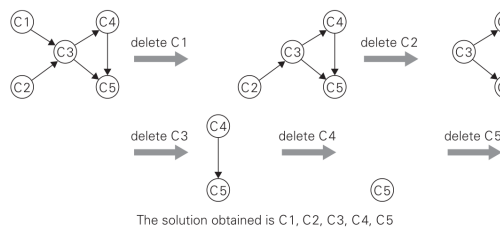
C5₁
C4₂
C3₃
C1₄ C2₅

(b)

The popping-off order:
C5, C4, C3, C1, C2
The topologically sorted list:
C2 → C1 → C3 → C4 → C5

(c)

FIGURE 4.7 (a) Digraph for which the topological sorting problem needs to be solved. (b) DFS traversal stack with the subscript numbers indicating the popping-off order. (c) Solution to the problem.



-Source:

-The order in which the vertices are removed is the solution to the topological sorting algorithm.

Binary Search:

ALGORITHM *BinarySearch*($A[0..n-1], K$)
 //Implements nonrecursive binary search
 //Input: An array $A[0..n-1]$ sorted in ascending order and
 // a search key K
 //Output: An index of the array's element that is equal to K
 // or -1 if there is no such element
 $l \leftarrow 0; \quad r \leftarrow n-1$
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

-efficiency: count the number of times the search key is compared with an element of the array.

Fake Coin problem

Russian peasant multiplication

$n \in m$

instance size by the value of n . Now, if n is even, an instance of half the size has to deal with $n/2$, and we have an obvious formula relating the solution to the problem's larger instance to the solution to the smaller one:

$$n \cdot m = \frac{n}{2} \cdot 2m.$$

If n is odd, we need only a slight adjustment of this formula:

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m.$$

- Josephus problem