**SUBJECT**: DESIGN AND ANALYSIS OF ALGORITHMS

**CODE**: 503040

Duration: 150 minutes

Allowed to use materials.

**LAB 05: Decrease-and-Conquer**

# Objectives

Understand the properties of Decrease-and-Conquer algorithm design technique

Be able to design, implement, and analyze Decrease-and-Conquer algorithms solving common problems.

# Idea

Reduce problem instance to smaller instance of the same problem

Solve smaller instance

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

There are 3 types of Decrease and Conquer

1. Decrease by a constant (usually by 1):
2. insertion sort
3. graph traversal algorithms (DFS and BFS)
4. topological sorting
5. Decrease by a constant factor (usually by half)
6. binary search
7. exponentiation by squaring
8. Variable-size decrease
9. Euclid’s algorithm
10. Selection by partition

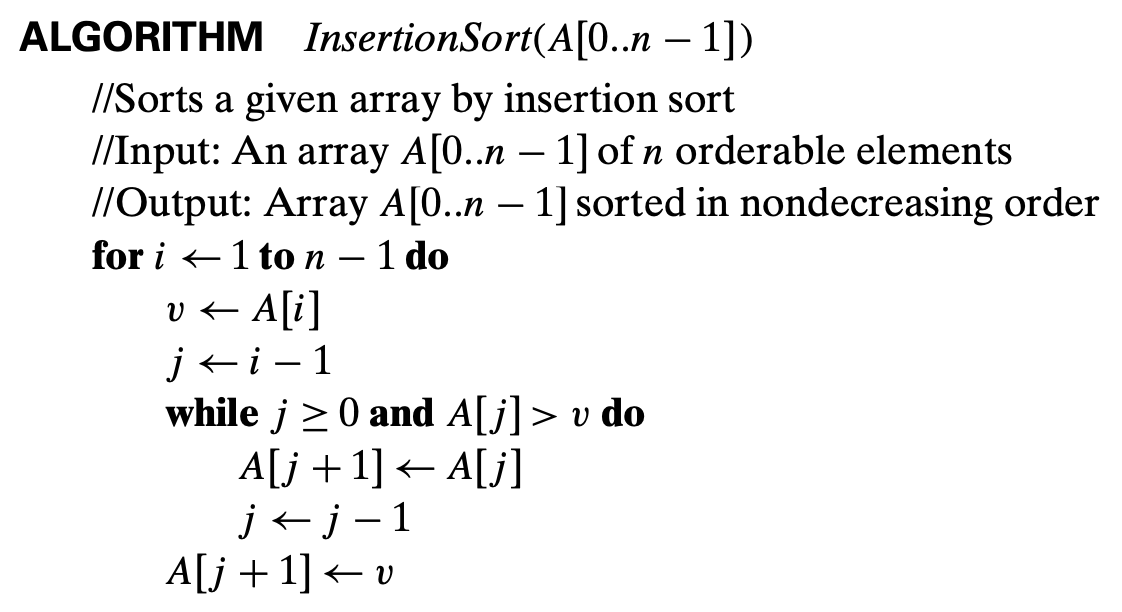
This usually results in a recursive algorithm.

## An example of a Decrease-and-Conquer algorithm

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)

Pseudocode of Insertion Sort algorithm is given as follows:



Implementation of InsertionSort in Python is presented as follows

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | **def** **insertion\_sort**(A):  """  function sorts a given array by insertion sort  Input:  A (array): sequence of n orderable elements  Output:  Array A sorted in nondescreasing order  """  n = len(A)  **for** i **in** range(**1**, n):  v = A[i]  j = i - **1**  **while** j >= **0** **and** A[j] > v:  A[j+**1**] = A[j]  j = j - **1**  A[j+**1**] = v  **return** A  #testcase 1  **import** **random**  X = list(range(**10**))  random.shuffle(X)  **print**(X)  Y = insertion\_sort(X)  **print**(Y) | |

Analysis:

1/ Basic operation: comparison A[j] > v on line 13

2/ Worst case: Input A is sorted in decreasing order

3/Counting the number of basic operations in the worst case:

…

**Time efficiency**

***Cworst*(*n*) = *n*(*n*-1)/2 ∈ Θ(*n*2)**

# Exercises

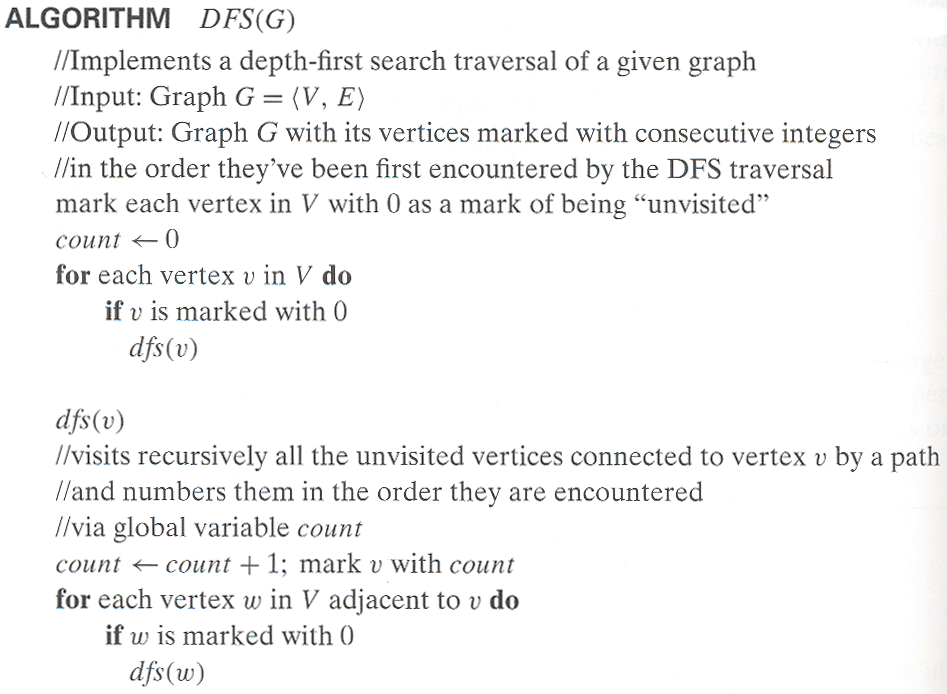
For each of the problems in this section, implement (in Python) and analyze a decrease-and-conquer algorithm to solve the problem.

**Warm up**

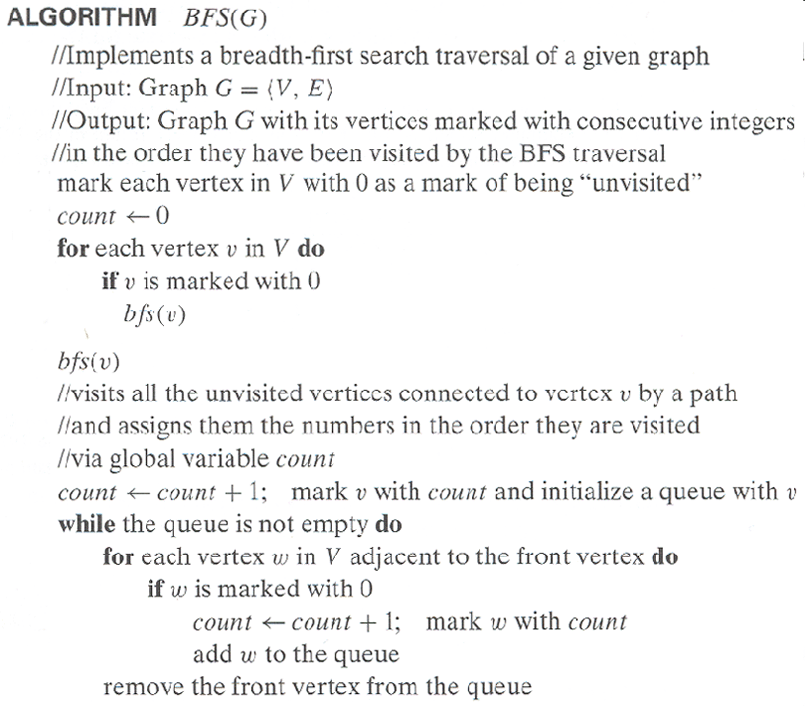
1. Recursive insertion sort.
2. Recursive Exponentiation by squaring
3. Recursive Euclid’s algorithm for greatest common divisor

**Intermediate exercises**

1. Depth-first search (DFS)



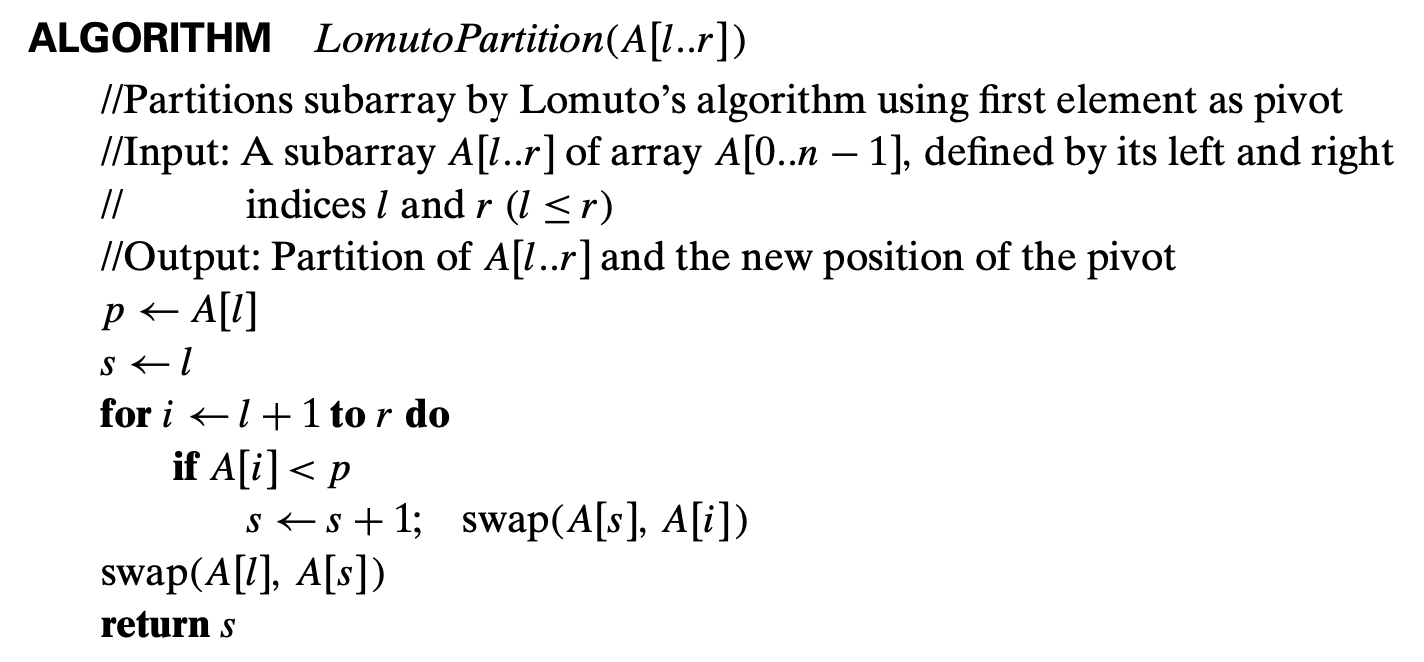
1. Breadth-first search (BFS)

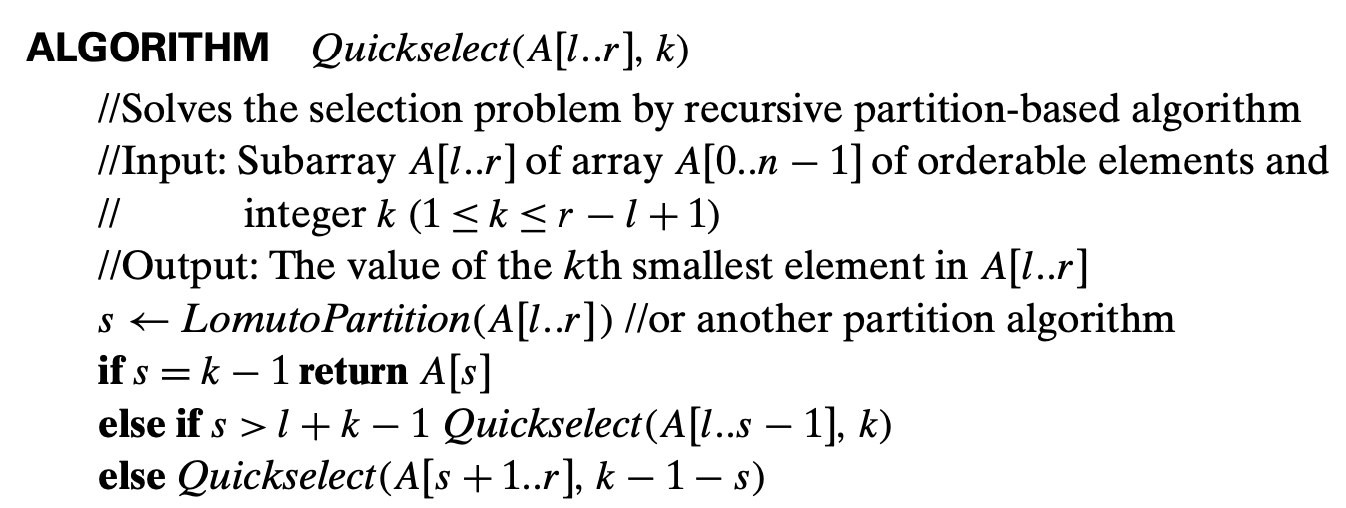


1. Topological Sorting
   1. Perform DFS traversal, noting the order vertices are popped off the traversal stack
   2. Reverse order solves topological sorting problem

**Upper-intermediate** **exercise**

1. Partition-based algorithm for selection problem





1. Binary Search Tree Algorithms
2. Searching
3. Insertion of a new key
4. Finding the smallest (or the largestdef) key