4.1

7. Apply insertion sort to sort the list *E, X, A, M, P, L, E* in alphabetical order.

E, X, *A, M, P, L, E*

E, ***X, A,*** *M, P, L, E →E,* ***A, X,*** *M, P, L, E →****A,*** *E,* ***X,*** *M, P, L, E*

A, E, ***X, M,*** *P, L, E →A, E,* ***M, X,*** *P, L, E*

A, E, M, ***X, P,*** *L, E →A, E, M,* ***P, X,*** *L, E*

A, E, M, P, ***X, L,*** *E →A, E, M, P,* ***L, X,*** *E →A, E, M,* ***L,*** *P,* ***X,*** *E →A, E,* ***L,*** *M, P,* ***X,*** *E*

A, E, L, M, P, ***X, E →****A, E, L, M, P,* ***E, X →****A, E, L, M,* ***E,*** *P,* ***X →****A, E, L,* ***E,*** *M, P,* ***X →****A, E,* ***E,*** *L, M, P,* ***X***

A, E, E, L, M, P, X

12. Shellsort (more accurately Shell's sort) is an important sorting algorithm that works by applying insertion sort to each of several interleaving sublists of a given list. On each pass through the list with an increment h_i taken from some predefined decreasing sequence of step sizes, $h_1 > \ldots > h_i > \ldots > 1$, which must end with 1. (The algorithm works for any such sequence, though some sequences are known to yield a better efficiency than others. For example, the sequence 1, 4, 13, 40, 121, . . ., used, of course, in reverse, is known to be the best for this purpose.)

a. Apply shellsort to the list *S, H, E, L, L, S, O, R, T, I, S, U, S, E, F, U, L*

S, H, E, L, L, S, O, R, T, I, S, U, S, E, F, U, L

S T

H I

E S

L U

L S

S E SWAP

O F SWAP

R U

L

S, H, E, L, L, E, F, R, T, I, S, U, S, S, O, U, L

(keep making the gap smaller and swap…)

E, E, F, H, I, L, L, L, O, R, S, S, S, S, T, U, U

b. Is shellsort a stable sorting algorithm?

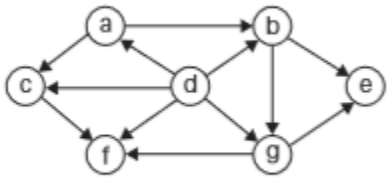
No.

c. Implement shellsort, straight insertion sort, selection sort, and bubble sort in the language of your choice and compare their performance on random arrays of sizes 10^n for n = 2, 3, 4, 5, and 6 as well as on increasing and decreasing arrays of these sizes.

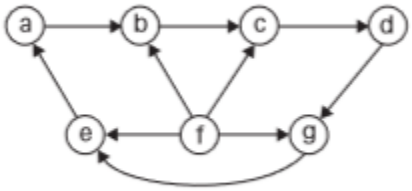
Shell sort is the fastest because of its O(nlogn) complexity, Insertion and Bubble sort were about the same and selection sort did the worst

4.2

1. Apply the DFS-based algorithm to solve the topological sorting problem for the following digraphs:



(a)



(b)

a. d b g e a c f

b. Not a dag, can't sort with DFS

5. Apply the source-removal algorithm to the digraphs of Problem 1 above.

a. d a c f b g e

b. Not a dag, can't sort with Source-Removal

9. A digraph is called ***strongly connected*** if for any pair of two distinct vertices *u* and *v* there exists a directed path from *u* and *v* and a directed path from *v* and *u*. In general, a digraph's vertices can be partitioned into disjoint maximal subsets of vertices that are mutually accessible via directed paths; these subsets are called ***strongly connected components*** of the digraph. There are two DFS-based algorithms for identifying strongly connected components. Here is the simpler (but somewhat less efficient) one of the two:

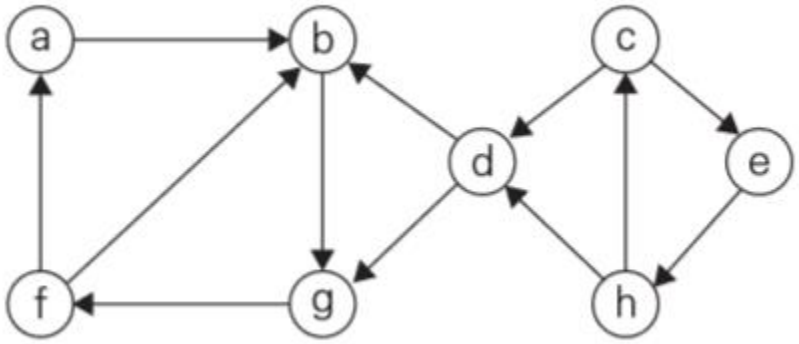
Step 1 Perform a DFS traversal of the digraph given and number its vertices in the order they become dead ends.

Step 2 Reverse the directions of all the edges of the digraph.

Step 3 Perform a DFS traversal of the new digraph by starting (and, if necessary, restarting) the traversal at the highest numbered vertex among still unvisited vertices.

The strongly connected components are exactly the vertices of the DFS trees obtained during the last traversal.

a. Apply this algorithm to the following digraph to determine its strongly connected components:



{c, e, h} →d → {a, b, g, f}

b. What is the time efficiency class of this algorithm? Give separate answers for the adjacency matrix representation and adjacency list representation of an input digraph.

Time efficiency for adjacency matrix representation is O(n^2)

Time efficiency Adjacency list representation is O(n)

c. How many strongly connected components does a dag have?

None.

4.3

1. Is it realistic to implement an algorithm that requires generating all permutations of a 25-element set on your computer? What about all the subset of such a set?

There are too many permutations of a 25-element set to be represented on a computer and as for the subset the are way less subset to represent so it is very much possible.

2. Generate all the subsets of a four-element set A = {a1, a2, a3, a4} by each of the two algorithms outlined in this section.

Answer:

First Method

N

0 ∅

1 ∅ {a1}

2 ∅ {a1} {a2} {a1, a2}

3 ∅ {a1} {a2} {a1, a2} {a3} {a1, a3} {a2, a3} {a1, a2, a3}

4 ∅ {a1} {a2} {a1, a2} {a3} {a1, a3} {a2, a3} {a1, a2, a3}
{a4} {a1, a4} {a2, a4} {a3, a4} {a1, a2, a4} {a1, a3, a4}
{a2, a3, a4} {a1, a2, a3, a4}

Second method

Bit String:	0000	0001	0010	0100	0111	1000	1001
Subset:	∅	{a4}	{a3}	{a2}	{a2, a3, a4}	{a1}	{a1, a4}

1010	1100	1110	1011
{a1, a3}	{a1, a2}	{a1, a2, a3}	{a1, a3, a4}

1101	1111
{a1, a2, a4}	{a1, a2, a3, a4}

4.4

3

a. What is the largest number of key comparisons made by binary search in searching for a key in the following array?

The maximum number of comparison that will take is 4

b. List all the keys of this array that will require the largest number of key comparisons when searched for by binary search.

All the keys are located in position 1, 3, 5, 8, 10, and 12.

c. Find the average number of key comparisons made by binary search in a successful search in this array. Assume that each key is searched for with

the same probability

The number of the average key comparison would be 41/13 which is 3.15.

d. Find the average number of key comparisons made by binary search in an unsuccessful search in this array. Assume that searches for keys in each of the 14 intervals formed by the array's elements are equally likely.

We need three key comparisons to make the key to do an unsuccessful search in this array which would be 3*(2/14) + 4*(12/14) = 54/14 = 3.86

4.5

2

Apply quickselect to find the median of the list of numbers 9, 12, 5, 17, 20, 30, 8.

The median is 17.

13

You need to search for a given number in an n × n matrix in which every row and every column is sorted in increasing order. Can you design a O(n) algorithm for this problem?

A) Find the middle element.

B) If middle element is same as key return.

C) If middle element is lesser than key then

…. Ca) search submatrix on lower side of middle element

…. Cb) Search submatrix on right hand side.of middle element

D) If middle element is greater than key then

…. Da) search vertical submatrix on left side of middle element

…. Db) search submatrix on right hand side.