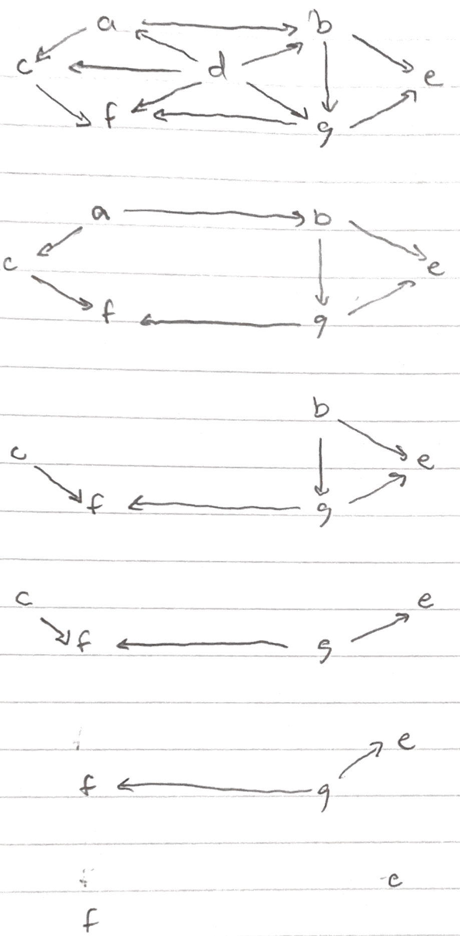
Homework 4

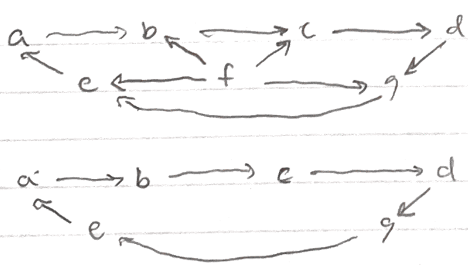
4.2 5a.



Solution

d; a, b, c, g, e, f

5b.



f

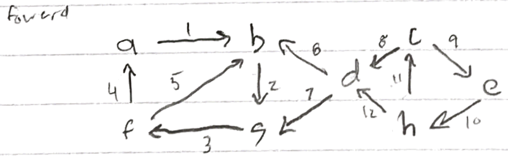
no solution since no more vertices that can be deleted because none are a vertex with no incoming edge.

6a. A non empty dag must have at least one source, because a dag is a digraph that contains no cycles and is directed, thus there exist a source called a dag's root. Therefore in every non-empty dag, there exist at least one source, which one being the dag's root which is the unique source of the dag.

b. To find a source in a digraph represented by its adjacency matrix, you would have to look at a vertex of a dag and check its column to see only 0's in the column. If a vertex of a dag only has 0's in its column then it is a source, since a vertex is a source if and only if its column in adjacency matrix contains only 0's. The time efficiency of this operation is $O(|V^2|)$

c. To find a source in a digraph represented by its adjacency lists, you would have see if a vertex doesn't appear in any of the dag's adjacency lists. If a vertex doesn't appear in any of the dag's adjacency lists, then it is a source because a vertex is a source if and only if the vertex appears in none of the dag's adjacency list. The time efficiency of this operation is $O(|V|+|E|)$.

9a. forward



f
g
b
a

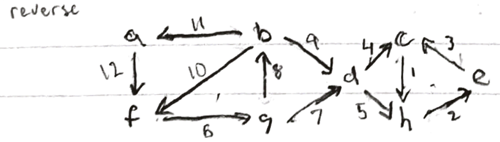
stac

order: a b g f d c e h

dead end
f_1 g_2 b_3 a_4 d_5 h_6 e_7 c_8

stack f_1 g_2 b_3 a_4 d_5 h_6 c_7 c_8

reverse



e_7 h_6 c_8 d_5

b_3 g_2 f_1 a_4

$c_8 \downarrow h_7 \downarrow e_6$

d_5

$a_4 \downarrow b_3 \downarrow g_2 \downarrow f_1$

Thus strongly connected components of the DFS tree are {c,h,e}, {d}, {a, f, g, b}

b. For adjaccny matrix

- DFS traversal in original digraph is $\Theta(|V|^2)$
- DFS traversal in the edge reversal graph $\Theta(|V|^2)$
- DFS traversal in the newly formed graph $\Theta(|V|^2)$

thus overall efficiency of algorithm $= \Theta(|V|^2) + \Theta(|V|^2) + \Theta(|V|^2)$

$= \boxed{\Theta(|V|^2)}$

For adjacency lists

- If the graph is represented by adjacency lists then efficiency is $\Theta(|V| + |E|)$

thus overall efficiency of algorithm $= \boxed{\Theta(|V| + |E|)}$.

c. In general, a dag has any where from 1 to N number of strongly connected components. In dag a, it has 3 strongly connected components.

4.3 2a. $\{1, 2, 3, 4\}$

start	1			
Insert 2 into 1 left to right	12	21		
Insert 3 into 12 left to right	123	132	312	
Insert 3 into 21 right to left	321	231	213	
Insert 4 into 123 left to right	1234	1243	1423	4123
Insert 4 into 132 right to left	4132	1432	1342	1324
Insert 4 into 312 left to right	3124	3142	3412	4312
Insert 4 into 321 right to left	4321	3421	3241	3214
Insert 4 into 231 left to right	2314	2341	2431	4231
Insert 4 into 213 right to left	4213	2413	2143	2134

b.

$\overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{3}\ \overleftarrow{4}$
$\overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{4}\ \overleftarrow{3}$
$\overleftarrow{1}\ \overleftarrow{4}\ \overleftarrow{2}\ \overleftarrow{3}$
$\overleftarrow{4}\ \overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{3}$
$\overrightarrow{4}\ \overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{3}$ —
$\overrightarrow{4}\ \overleftarrow{1}\ \overleftarrow{3}\ \overleftarrow{2}$
$\overleftarrow{1}\ \overrightarrow{4}\ \overleftarrow{3}\ \overleftarrow{2}$
$\overleftarrow{1}\ \overleftarrow{3}\ \overrightarrow{4}\ \overleftarrow{2}$
$\overleftarrow{1}\ \overleftarrow{3}\ \overleftarrow{2}\ \overrightarrow{4}$
$\overleftarrow{1}\ \overleftarrow{3}\ \overleftarrow{2}\ \overleftarrow{4}$ —
$\overrightarrow{3}\ \overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{4}$
$\overrightarrow{3}\ \overleftarrow{1}\ \overleftarrow{4}\ \overleftarrow{2}$
$\overrightarrow{3}\ \overleftarrow{4}\ \overleftarrow{1}\ \overleftarrow{2}$
$\overleftarrow{4}\ \overleftarrow{3}\ \overleftarrow{1}\ \overleftarrow{2}$

$\overrightarrow{4}\ \overrightarrow{3}\ \overleftarrow{1}\ \overleftarrow{2}$ —
$\overrightarrow{4}\ \overrightarrow{3}\ \overleftarrow{2}\ \overleftarrow{1}$
$\overrightarrow{3}\ \overrightarrow{4}\ \overleftarrow{2}\ \overleftarrow{1}$
$\overrightarrow{3}\ \overleftarrow{2}\ \overrightarrow{4}\ \overleftarrow{1}$
$\overrightarrow{3}\ \overleftarrow{2}\ \overleftarrow{1}\ \overrightarrow{4}$
$\overrightarrow{3}\ \overleftarrow{2}\ \overleftarrow{1}\ \overleftarrow{4}$ —
$\overleftarrow{2}\ \overrightarrow{3}\ \overleftarrow{1}\ \overleftarrow{4}$
$\overleftarrow{2}\ \overrightarrow{3}\ \overleftarrow{4}\ \overleftarrow{1}$
$\overleftarrow{2}\ \overleftarrow{4}\ \overrightarrow{3}\ \overleftarrow{1}$
$\overleftarrow{4}\ \overleftarrow{2}\ \overrightarrow{3}\ \overleftarrow{1}$
$\overrightarrow{4}\ \overleftarrow{2}\ \overrightarrow{3}\ \overleftarrow{1}$ —
$\overrightarrow{4}\ \overleftarrow{2}\ \overleftarrow{1}\ \overrightarrow{3}$
$\overleftarrow{2}\ \overrightarrow{4}\ \overleftarrow{1}\ \overrightarrow{3}$
$\overleftarrow{2}\ \overleftarrow{1}\ \overrightarrow{4}\ \overrightarrow{3}$
$\overleftarrow{2}\ \overleftarrow{1}\ \overrightarrow{3}\ \overrightarrow{4}$

* lines with dashes show change of arrows and are not part of permutation list

c. (1 2 3 4) (2 1 3 4) (1 3 2 4) (3 1 2 4) (2 3 1 4) (3 2 1 4)
(1 2 3 4) (2 1 4 3) (1 4 2 3) (4 1 2 3) (2 4 1 3) (4 2 1 3)
(1 3 4 2) (3 1 4 2) (1 4 3 2) (4 1 3 2) (3 4 1 2) (4 3 1 2)
(2 3 4 1) (3 2 4 1) (2 4 3 1) (4 2 3 1) (3 4 2 1) (4 3 2 1)

5. Squashed order

Bit String	Subsets	Bit String	Subsets
0000	$\emptyset$	1000	$\{a_1\}$
0001	$\{a_4\}$	1001	$\{a_1, a_4\}$
0010	$\{a_3\}$	1010	$\{a_1, a_3\}$
0011	$\{a_3, a_4\}$	1011	$\{a_1, a_3, a_4\}$
0100	$\{a_2\}$	1100	$\{a_1, a_2\}$
0101	$\{a_2, a_4\}$	1101	$\{a_1, a_2, a_4\}$
0110	$\{a_2, a_3\}$	1110	$\{a_1, a_2, a_3\}$
0111	$\{a_2, a_3, a_4\}$	1111	$\{a_1, a_2, a_3, a_4\}$

Binary Gray Code

Binary code	Gray code	subsets
0	0000	$\{\emptyset\}$
1	0001	$\{a_1\}$
2	0011	$\{a_1, a_2\}$
3	0010	$\{a_2\}$
4	0110	$\{a_2, a_3\}$
5	0111	$\{a_1, a_2, a_3\}$
6	0101	$\{a_1, a_3\}$
7	0100	$\{a_3\}$
8	1100	$\{a_3, a_4\}$
9	1101	$\{a_1, a_3, a_4\}$
10	1111	$\{a_1, a_2, a_3, a_4\}$
11	1110	$\{a_2, a_3, a_4\}$
12	1010	$\{a_2, a_4\}$
13	1011	$\{a_1, a_2, a_4\}$
14	1001	$\{a_1, a_4\}$
15	1000	$\{a_4\}$

L

0   1

L_1 L_2

0   1   1   0

00   01   11   10

L_1 L_2

00 01 ~~10~~ 10   10 11 01 00

000 001 011 010   110 111 101 100

L_1

000 001 011 010 110 111 101 100

0000 0001 0011 0010 0110 0111 0101 0100

L_2

100 101 111 110 010 011 001 000

1100 1101 1111 1110 1010 1011 1001 1000

7. Algorithm BitString (n)

```
//Input: positive integer n
//Output: All bit strings of length n in B[0...n-1]
if n = 0
    print(B)
else
    B[n-1] ← 0, BitString(n-1)
    B[n-1] ← 1, BitString(n-1)
```

9a. Check cpp file that was submitted with this pdf.

9b.

Binary number	Gray code
0 = 0000	G0 = 0000
1 = 0001	G1 = 0001
2 = 0010	G2 = 0011
3 = 0011	G3 = 0010
4 = 0100	G4 = 0110
5 = 0101	G5 = 0111
6 = 0110	G6 = 0101
7 = 0111	G7 = 0100
8 = 1000	G8 = 1100
9 = 1001	G9 = 1101
10 = 1010	G10 = 1111
11 = 1011	G11 = 1110
12 = 1100	G12 = 1010
13 = 1101	G13 = 1011
14 = 1110	G14 = 1001
15 = 1111	G15 = 1000

4.4 2. Algorithm LogBase2 (n)

```
    // Input: integer of which log2 value to be computed
    // Output: return floor of log2 n
    if n = 1
        return 0
    else
        return 1 + LogBase2 (⌊n/2⌋)
```

$$T(n) \begin{cases} 0 & \text{for } n = 1 \\ 1 + T(\lfloor \frac{n}{2} \rfloor) & \text{for } n > 1 \end{cases}$$

Master Theorem

$a = 1$ $b = 2$ $d = 0$ since constant count

$1 = 2^0 \rightarrow 1 = 1$ thus $\Theta(n^d \log n) \rightarrow \Theta(n^0 \log n)$

$\therefore$ therefore the time efficiency is $\Theta(\log_2 n)$

W 2 1098 T

3 a. 3 | 14 | 27 | 31 | 39 | 42 | 55 | 70 | 74 | 81 | 85 | 93 | 98

$n = 13$

largest number of key comparisons $= \lceil \log_2 (n+1) \rceil$

$= \lceil \log_2 (13+1) \rceil$

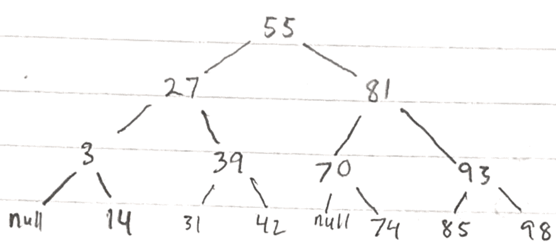
$= \lceil \log_2 (14) \rceil$

$= \lceil 3.8077 \rceil$

$= 4$

$\therefore$ largest number of key comparisons made by binary search is 4.

b.



The keys of this array that will require the largest number of key comparisons when searched for by binary search are 14, 31, 42, 74, 85, 98.

c. $C^{yes}_{avg}(n) = \log_2 n - 1$

$= \log_2 (13-1)$

$= \log_2 (12)$

$= 3.58$

Average number of key comparisons made by binary search in successful search in the array is 3.58

d. $C^{no}_{avg}(n) = \log_2 n + 1$

$= \log_2 (13+1)$

$= \log_2 (14)$

≈ 3.807

Average number of key comparisons made by binary search in an unsuccessful search in this array is 3.807

12a.
```
unsigned int russianPeasant (unsigned int x, unsigned int y) {
        int result = 0;
        while (x > 0) {
            if (x % 2 == 1)
                result = result + y
            x = x / 2;
            y = y * 2;
        }
        return result;
}
```

12b. $T(n) = \begin{cases} 0 & \text{for } x \leq 0 \\ 1 + T(\lfloor \frac{n}{2} \rfloor) & \text{for } x > 0 \end{cases}$

Master theorem, $a = 1$ $b = 2$ $d = 0$ since 1 and not n

$1 = 2^0 \Longrightarrow 1 = 1$ ✓

Thus $\Theta(n^d \log n) \rightarrow \Theta(n^0 \log n) \rightarrow \boxed{\Theta(\log n)}$

14. $J(2) = J(10) = 01 = 1$
$J(4) = J(100) = 001 = 1$
$J(8) = J(1000) = 0001 = 1$
$J(16) = J(10000) = 00001 = 1$
$J(32) = J(100000) = 000001 = 1$

For all powers of 2, the most significant bit is 1 and the rest of the bits will be 0's, So when the left shift of 1 bit is done, the resulting bit equals 1. Hence the solution to the Josephus problem is 1 for every n that is a power of 2

4.5 1a. If we measure an instance size of computing the greatest common divisor of m and n by the size of the second number n, after one iteration of Euclid's algorithm the size can decrease by any number between 1 and n. It decreases by any number between 1 and n because Euclid's algorithm uses formula gcd (m, n) which equals gcd (n, m mod n), thus the size of new pair will be m mod n.

When number of times operation is done $f(n)$ is a constant then $d = 0$

1b. $\gcd(m, n) = \gcd(n, m \bmod n) = \gcd(m \bmod n, \; n \bmod (m \bmod n))$

Consider $m \bmod n \le n/2$ and $n/2 < r < n$

If $r \le n/2$

$n \bmod (m \bmod n) < r \le n/2$

If $n/2 < r < n$

$n \bmod (m \bmod n) = n - (m \bmod n) < n/2$

Thus the size of an instance will always decrease at least by a factor of 2 after two successive iterations of Euclid's algorithm.

2.

9	12	5	17	20	30	8
p	~~s~~ i					

9	5	12	17	20	30	8
p	s		~~i~~ i			

9	5	8	17	20	30	12
p		s				

8	5	9	17	20	30	12
		p		i		

8	5	9	17	20	30	12
			p s	s ~~i~~ i		

8	5	9	17	12	30	20
			p s	s		i

8	5	9	12	17	30	20
				p s		i

Since $s = 4^{th}$ index and is greater than $k - 1 = 3$ and the left part of the list has a single number at index 3 which is ~~the~~ median of the list, ∴ the median of the given list is 12.

```
3. int partition(list, left, right, pivot) {
      int pivotvalue = list[pivot]
      swap(list[pivot], list[right]
      int storeIndex = left
      for i ← left to right - 1 do
           if list[i] < pivot value
                swap (list[store index], list[i])
                store index ++
      swap(list[right], list[store index])
      return storeindex

   int quickselect (list, left, right, n)
        while (1)
             if left = right
                  return list[left]
             pivot index = random (left, right)
             pivot index = partition (list, left, right, pivotindex)
             if n = pivot index
                  return list[n]
             elseif n < pivot index
                  right = pivot index - 1
             else
                  left = pivot index + 1.
```

8a. There are three cases of deleting a key from a binary tree.

case 1: If a key to be deleted is a leaf, make the pointer from its parent to the key's node null. If it doesn't have a parent since it is a root of a single tree, then make the tree empty

case 2: If a key to be deleted is a node with a single child, make the pointer from it's parent to key's node to point to the child. If the node that is being deleted is the root with the single child, make child the new root

case 3: If a key K to be deleted is a node with two children, its deletion takes these three stage procedure. First find the smallest key k in the right subtree of the K's node. k is the immediate successor of K in the inorder traversal of the given binary tree. It can be found by making one step to the right from the K's node and then all the way to the left until a node with no left subtree is reached. Second, exchange K and k. Third delete K in its new node by using case 1 or 2 depending on whether if the node is a leaf or has child.

This algorithm is not a variable-size decrease algorithm because it doesn't reduce the problem to that of deleting a key from a smaller binary tree

b. The time efficiency in the worst class is $\Theta(n)$, because the worst case is deleting the root from the binary tree and finding the smallest node in the right subtree requires following a chain of $n-2$ pointers which is $\Theta(n)$.