CSCI-163 HW#2 Jesse Mayer

3.1-4: a. Algorithm Polynomial (P[0..n], x)
Poly ← 0
for i ← n to 0 do
    pow ← 1
    for j ← 1 to i do
        pow ← pow * X
    poly ← poly + P[i] * pow
return poly

$M(n) = \sum_{i=0}^{n} \sum_{j=1}^{i} 1 = \sum_{i=0}^{n} = \frac{n(n+1)}{2} \in \theta(n^2)$

b. Algorithm Polynomial V2 (P[0..n], x)
Poly ← P[0]
pow ← 1
for i ← 1 to n do
    pow ← pow * x
    Poly ← Poly + P[i] * pow
return Poly

$M(n) = \sum_{i=1}^{n} 2 = 2n$

c. It is not possible to design an algorithm with a better-than-linear efficiency for this problem because to evaluate a polynomial of degree n at a point X_0 we must process all n+1 coefficients and make n additions to do so.

3.1-7: Algorithm FindFake (Stack[0..n-1], Coin[0..n-1])
for i 0 to n-1 do
    if Stack[i] -> Coin[i].weight == 11
        fake ← i
        return fake

3.1-8: EXAMPLE -> AXEMPLE -> AEXMPLE
-> AEEMPLX -> AEELPMX -> AEELMPX

3.1-9: Selection sort is stable since it only updates the position of min if there is a value less than the value at the current min position. Therefore it preserves the original order of equal values and is thus by definition stable.

3.1-11: $E \overset{?}{\leftrightarrow} X \overset{?}{\leftrightarrow} AMPLE \rightarrow EA \overset{?}{X} \leftrightarrow MPLE \rightarrow EAM \overset{?}{X} \leftrightarrow PLE$
$\rightarrow EAMP \overset{?}{X} \leftrightarrow LE \rightarrow EAMPL \overset{?}{X} \leftrightarrow E \rightarrow EAMPLE \overset{?}{X} \rightarrow E \overset{?}{\leftrightarrow} AMPLEX$
$\rightarrow AE \overset{?}{\leftrightarrow} M \overset{?}{\leftrightarrow} P \overset{?}{\leftrightarrow} LEX \rightarrow AEMLP \overset{?}{\leftrightarrow} EX \rightarrow AEMLEPX$
$\rightarrow A \overset{?}{\leftrightarrow} E \overset{?}{\leftrightarrow} M \overset{?}{\leftrightarrow} LEPX \rightarrow AELM \overset{?}{E} PX \rightarrow AELEM \overset{?}{\leftrightarrow} P \overset{?}{\leftrightarrow} X$
$\rightarrow A \overset{?}{\leftrightarrow} E \overset{?}{\leftrightarrow} L \overset{?}{\leftrightarrow} EMPX \rightarrow AEEL \overset{?}{\leftrightarrow} M \overset{?}{\leftrightarrow} P \overset{?}{\leftrightarrow} X \rightarrow AEELMPX$

3.1-13: Bubble sort is a stable algorithm because it does not swap equal elements and therefore does not alter the relative order of equal elements, thus making it stable by definition.

3.2-1: a. $C_{worst}(n) = n+1$

b. $C_{avg}(n) = \frac{P}{n} + \frac{2P}{n} + \cdots \frac{iP}{n} + \cdots + \frac{nP}{n} + (n+1)(1-P)$

$= \frac{P}{n}[1+2+\cdots+i+\cdots+n] + (n+1)(1-P) = \frac{P}{n}\frac{n(n+1)}{2} + (n+1)(1-P)$

$= \frac{(2-P)(n+1)}{2}$

3.2-4: Number of trials = $n - m + 1 = 47 - 6 + 1 = 42$
Number of character comparisons = 42 - 1 (failed comparison) + 2 (successful comparisons) = 43

3.2-8: a. Algorith Substrings (T[0..n-1])
count ← 0
for i ← 0 to n-2 do
  if (T[i] = A) do
    for j ← i to n-1 do
      if (T[j] = B) do
        count ← count + 1
return count
$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i}^{n-1} 1 = \sum_{i=0}^{n-2} n-1-i+1 = \sum_{i=0}^{n-2} n-i = \frac{n(n+1)}{2} - 1 \in \Theta(n^2)$

b. Algorith SubstringsV2(T[0..n-1])
total_count ← 0
count_A ← 0
for i ← 0 to n-1 do
  if (T[i] = A) do
    count_A ← count_A + 1
  if (T[i] = B) do
    total_count ← total_count + count_A
return total_count

3.3-2: Algorithm ClosestPair(A[0..n-1])
//sort n points in ascending order
for i ← 0 to n-2 do
  min ← i
  for j ← i+1 to n-1 do
    if A[j] < A[min]
      min ← j
  swap A[i] and A[min]
for k ← 0 to n-2 do //compute distance between adjacent points
  D[k] = A[k+1] - A[k]
min ← 0
for m ← 0 to n-2 do //Find smallest distance
  min ← min{min, D[m]}

3.3-4: a. (i) $d_M(P_1, P_2) \geq 0$ since $d_M(P_1,P_2) = |x_1-x_2| + |y_1-y_2|$, the absolute functions mean the result will be therefore nonnegative. Also $d_M(P_1, P_2) = 0$ if and only if $P_1 = P_2$ because that is the only case in which the distance between two points is 0.

(ii) $d_M(P_1,P_2) = |x_1 - x_2| + |y_1 - y_2|$

$d_M(P_2, P_1) = |x_2 - x_1| + |y_2 - y_1|$

due to the nature of the absolute function:

$|x_1 - x_2| = |x_2 - x_1|$ and $|y_1 - y_2| = |y_2 - y_1|$

so $|x_1-x_2| + |y_1-y_2| = |x_2 - x_1| + |y_2 - y_1|$

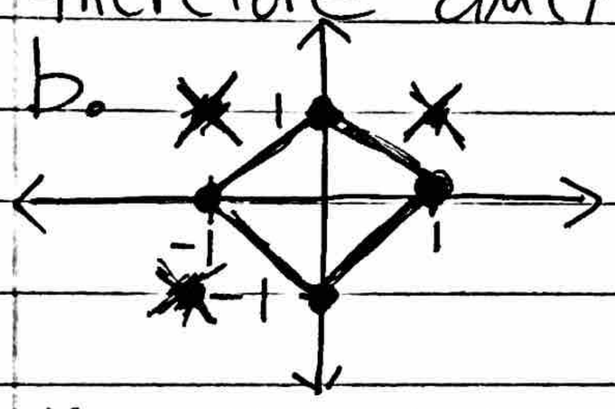
therefore $d_M(P_1, P_2) = d_M(P_2, P_1)$

(iii) $d_M(P_1,P_2) = |x_1 - x_2| + |y_1 - y_2| = |(x_1 - x_3) + (x_3 - x_2)| + |(y_1 - y_3) + (y_3 - y_2)|$

$\leq |(x_1 - x_3)| + |(x_3 - x_2)| + |(y_1 - y_3)| + |(y_3 - y_2)| = d_M(P_1, P_3) + d_M(P_3, P_2)$

therefore $d_M(P_1, P_2) \leq d_M(P_1, P_3) + d_M(P_3, P_2)$

b.

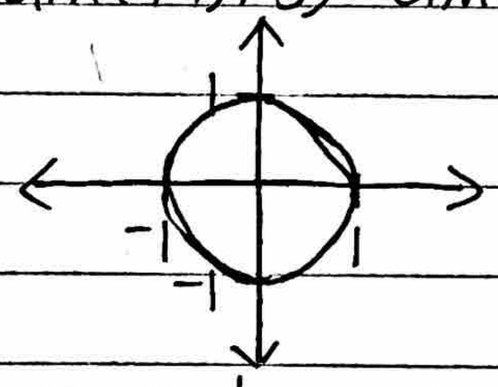


Manhattan distance

$|(x-0)| + |(y-0)| = 1$

$|x| + |y| = 1$

$\pm x \pm y = 1$



Euclidean distance

$\sqrt{(x-0)^2 + (y-0)^2} = 1$

$x^2 + y^2 = 1$

c. False, for example for points $P_1(0,0)$, $P_2(3/4, 1/4)$:

$d_E(P_1, P_2) = \sqrt{(0-3/4)^2 + (0-1/4)^2} < 1$

$d_M(P_1, P_2) = |0 - 3/4| + |0 - 1/4| = 1$

$< 1 \neq 1$

3.4-1: a. $A(n) = n \times \frac{1}{2}(n-1)! = \frac{1}{2} n! \in \Theta(n!)$

b. (i) 1 hour = 60 min x 60 sec = 3600 s

Additions = $(10^{10}/s)(3600 s) = 3.6 \times 10^{13}$

$18! = 6.40 \times 10^{15}$, $17! = 3.56 \times 10^{14}$, $16! = 2.09 \times 10^{13}$

$2.09 \times 10^{13} < 3.6 \times 10^{13} < 3.56 \times 10^{14}$

The max number of cities for which the problem can be solved in this time is 16.

(ii) 24 hours $= 24 \times 60\,\text{min} \times 60\,\text{s} = 86400\,\text{s}$

Additions $= (10^{10}/\text{s})(86400\,\text{s}) = 8.64 \times 10^{14}$

$3.56 \times 10^{14} < 8.64 \times 10^{14} < 6.40 \times 10^{15}$

The max number of cities for which the problem can be solved in this time is 17.

3.4-6: Algorithm Partition (A[0..n-1])

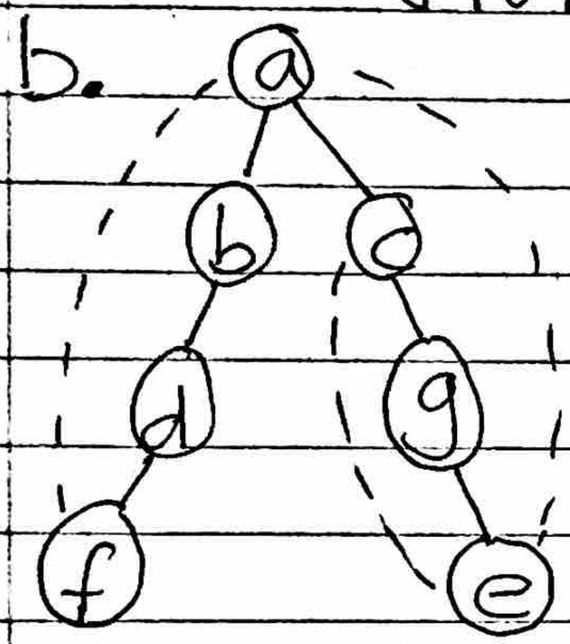
```
Total_Sum ← 0
for i ← 0 to n-1 do
    Total_Sum ← Total_Sum + A[i]
if Total_Sum mod 2 = 1
    Exit // Set cannot be evenly divided into two parts
k ← 0, m ← 0, S1[k] ← A[0], S1_Sum ← A[0], S2_Sum ← 0,
for j ← 1 to n-1 do                          k ← k+1
    if (S1_Sum ≥ S2_Sum)
        S2[m] ← A[j]
        S2_Sum ← S2_Sum + A[j]
        m ← m+1
    else
        S1[k] ← A[j]
        S1_Sum ← S1_Sum + A[j]
        k ← k+1
return S1, S2
```

3.5-1: a.

	a	b	c	d	e	f	g
a	0	1	1	1	1	0	0
b	1	0	0	1	0	1	0
c	1	0	0	0	0	0	1
d	1	1	0	0	0	1	0
e	1	0	0	0	0	0	1
f	0	1	0	1	0	0	0
g	0	0	1	0	1	0	0

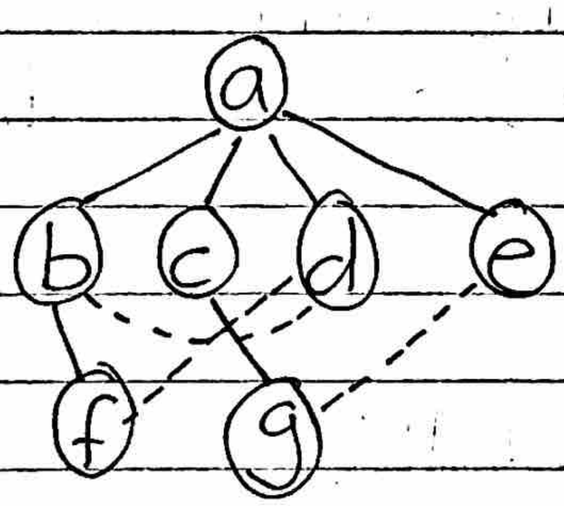
a → b → c → d → e
b → a → d → f
c → a → g
d → a → b → f
e → a → g
f → b → d
g → c → e

b.



$f_{4,1}$
$d_{3,5}$ $e_{7,2}$
$b_{2,6}$ $g_{6,3}$
$a_{1,7}$ $c_{5,4}$

3.5-4:



3.5-8: a. Algorithm DFS_Bipartite
for all v in V do visited(v)=0
for all v in V do if (visited(v)==0) DFS(v,0)
DFS(v,p)
  visited(v)=1
  part(v)=p
  for all w in adj(v) do {
    if (visited(w)==0){
      DFS(w, 1-p)}
    else{

```
            if(part(v) == part(w))
                return 0
    }}}return 1}
b. Algorithm BFS_Bipartite(G,s)
 For each vertex v in V[G]-{s} do
        color[v] ← Blue
          d[v] ← ∞
          partition[v] ← 0
    color[s] ← Red
    partition[s] ← 1
    d[s] ← 0
    Q ← [s]
    while Queue 'Q' is empty do
        v ← head[Q]
          for each u in Adj[v] do
            if partition[v] ← partion[u]
                return 0
            else
                if color[u] ← Blue
                    color[u] ← Red
                    d[u] ← d[v] +1
                    partition[u] ← 3 - partition[v]
                    Enqueue(Q,V)
                    Dequeue(Q)
    Color[v] ← Green
    Return 1
```

4.1-4: Algorithm Powerset($\{a_1,\dots,a_n\}$, n)
if n=0
  return $\{\}$
T(n-1) = PowerSet($\{a_1,\dots,a_{n-1}\}$, n-1)
return T(n) = $\{a_n\} \cup \{a_n, T(n-1)\}$

4.1-6: Algorithm Tournament(n) // 1=win, 0=tie, -1=loss
for i ← 1 to n-1
  S[i][i+1] ← 1 or 0
  ~~for j ← i+2 to n~~
  if S[i][i+1] = 1
    S[i+1][i] = -1
  if S[i][i+1] = 0
    S[i+1][i] = 0
  for j ← i+2 to n
    S[i][j] = 1, 0, or -1
    if S[i][j] = 1
      S[j][i] = -1
    if S[i][j] = 0
      S[j][i] = 0
    if S[i][j] = -1
      S[j][i] = 1
return S[][]

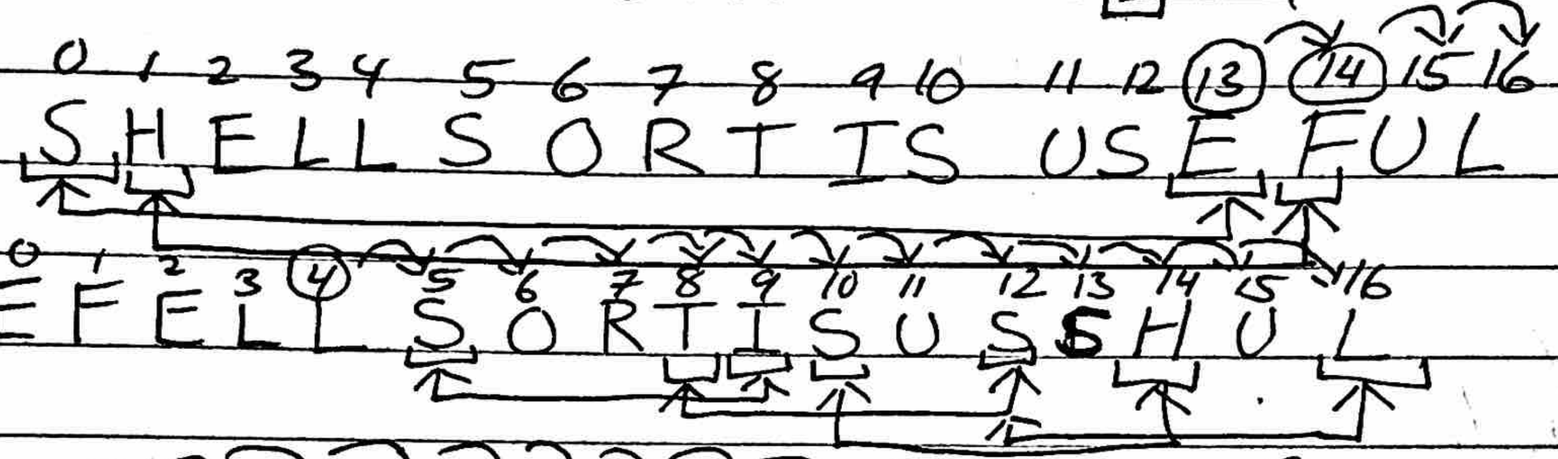
$$C(n) = \sum_{i=1}^{n-1} \sum_{j=i+2}^{n} 1 \approx \frac{n(n-1)}{2} \in \theta(n^2)$$

4.1-7: E|XAMPLE → EX|AMPLE → AEX|MPLE
→ AEMX|PLE → AEMPX|LE → AELMPX|E
→ AEELMPX

4.1-8: a. Any value less than or equal to every element in the array, such as -1 for an array of nonnegative integers.

b. Yes, the worst case for both the orignal and sentinel versions is: $\sum_{i=1}^{n-1}\sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} \in \Theta(n^2)$

4.1-12: a. SHELLSORTISUSEFUL

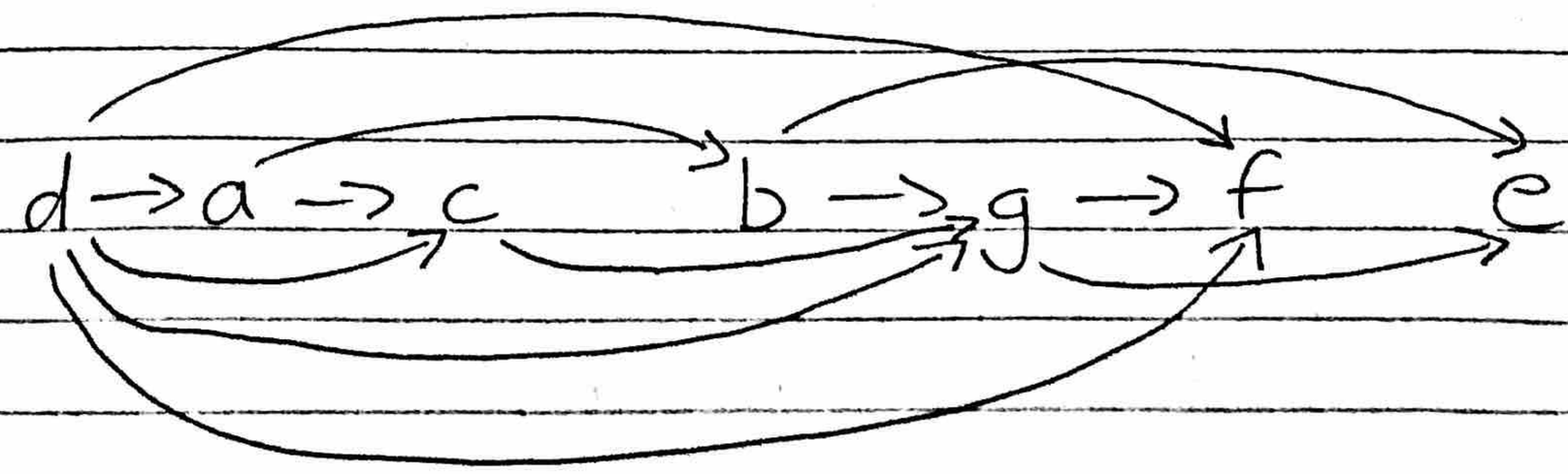


b. Shellsort is not a stable algorithm since it swaps values that are located far from each other, Thus altering the respective orientation of equal values.

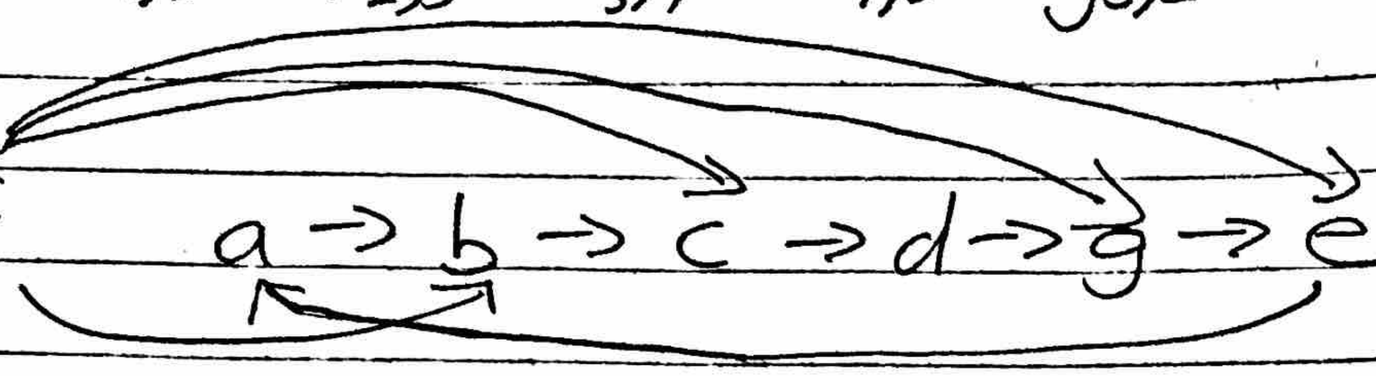
c.

array size	10^2	10^3	10^4	10^5	10^6
shellSort	2.17×10^{-3}ms	3.42×10^{-2}ms	5×10^{-1}ms	6.71ms	—
insertionSort	3.75×10^{-4}ms	3.44×10^{-3}ms	6.2×10^{-2}ms	3.2×10^{-1}ms	—
selectionSort	1.41×10^{-2}ms	1.29ms	128ms	1.31×10^{4}ms	—
bubbleSort	1.30×10^{-2}ms	1.21ms	123ms	1.24×10^{4}ms	—

4.2-1: a. $a_{1,6}$ $b_{2,4}$ $e_{3,1}$ $g_{4,3}$ $f_{5,2}$ $c_{6,5}$ $d_{7,7}$

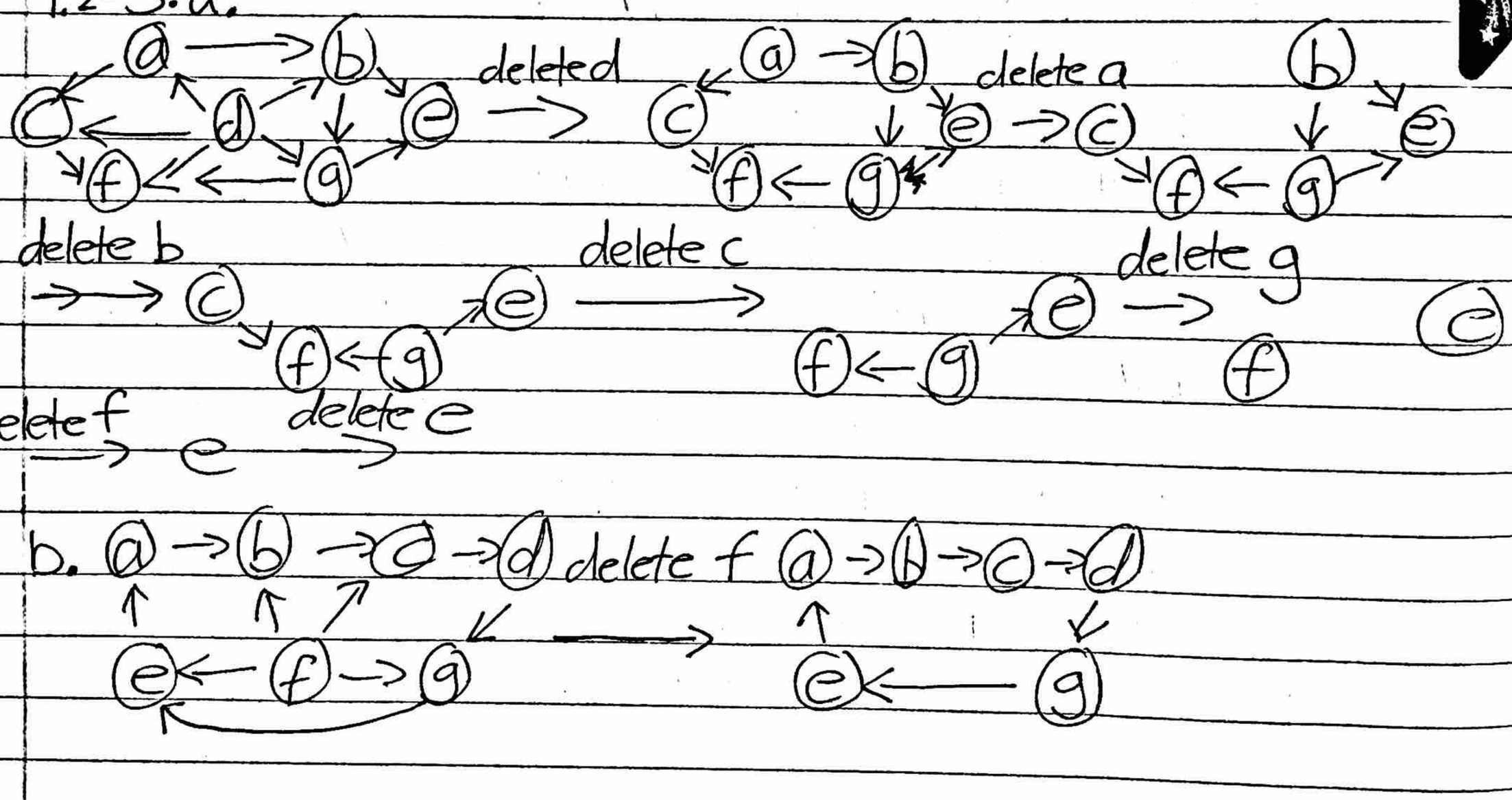


b. $a_{1,6}$ $b_{2,5}$ $c_{3,4}$ $d_{4,3}$ $g_{5,2}$ $e_{6,1}$ $f_{7,7}$



4.2-3: a. For topological sorting it is in $\Theta(|V|^2)$ for an adjacency matrix and $O(|V|+|E|)$ for an adjacency list.
b. To avoid reversing the vertex ordering fill an array from right to left as the vertices are popped.

4.2-5: a.



b. a → b → c → d delete f a → b → c → d

4.2-6: a. A dag is a directed graph with no cycles and a source is a node with no incoming edges. Therefore a dag must have at least one source otherwise it would have a cycle and therefore wouldn't be a dag.

b. We know a vertex is a source if its column contains only 0's in the adjacency matrix and this operation has a time efficiency of $O(|V|^2)$.
c. We know a vertex is a source if it is not in any of the other vertices adjacency lists, and this operation has a time efficiency of $O(|V|+|E|)$.

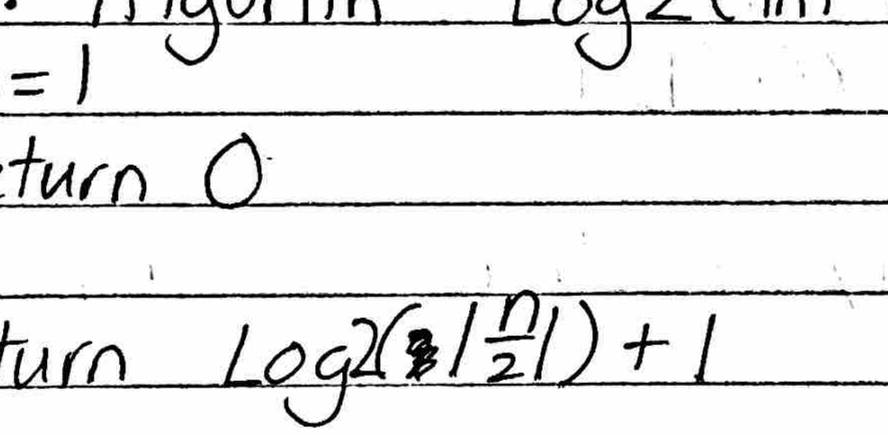
4.2-9: a. step1: $a_{1,4}$ $b_{2,3}$ $g_{3,2}$ $f_{4,1}$ $d_{5,5}$ $c_{6,8}$ $e_{7,7}$ $h_{8,6}$
step3: c_8, h_7, e_6; d_5; a_4, b_3, g_2, f_1
The strongly connected components are: {c, h, e}, {d}, {a, b, g, f}
b. The time efficiency of the adjacency matrix is $\Theta(|V|^2)$ and $\Theta(|V|+|E|)$ for the adjacency list.
c. A dag can have any number of strongly connected components, in this example it has 3.

4.4-2: Algorith Log2(int n)
if n=1
  return 0
else
  return Log2($\lfloor n/2 \rfloor$) + 1

$T(n) = \Theta(\log_2 n)$

4.4-3: a. $C_{worst}(13) = \lceil \log_2 (13+1) \rceil = \lceil 3.81 \rceil = 4$

b.



c. $C_{avg}^{yes}(13) = \log_2 13 - 1 = 2.700$

d. $C_{avg}^{NO}(13) = \log_2(13+1) = 3.807$

4.4-12: a. Algorithm RPM(int n, int m)

if n = 1

return m

else if n mod 2 = 0

return RPM(n/2, 2*m)

else

return RPM((n-1)/2, 2*m) + m

$C(n) = \Theta(\log_2 n)$

4.4-14: $J(2) = J(10_2) = 01_2 = 1$

$J(4) = J(100_2) = 001_2 = 1$

$J(8) = J(1000_2) = 0001_2 = 1$

$J(16) = J(10000_2) = 00001_2 = 1$

$J(32) = J(100000_2) = 000001_2 = 1$

...

Therefore for every n that is a power of 2 $J(n) = 1$.