CSC1-163 HW#3 JesseMayer

4.5-1:a. $gcd(m,n) = gcd(n, m \bmod n)$

$m \bmod n = 0$ to $n-1$

Therefore one iteration reduces the value of n between 1 and n.

b. $l = m \bmod n$

$gcd(m,n) = gcd(n,l) = gcd(l, n \bmod l)$

$l \leq n/2 \Rightarrow n \bmod l < l \leq n/2$

$n/2 < l < n \Rightarrow n \bmod l = n - l < n/2$

Therefore after two iterations of Euclid's algorithm the instance size will always be decreased by at least a factor of two.

4.5-2: $K = \lceil 7/2 \rceil = 4$

	0	1	2	3	4	5	6
	q^s	12^i	5	17	20	30	8
	q^s	12	5^i	17	20	30	8
	9	5^s	12^i	17	20	30	8
	9	5^s	12	17	20	30	8^i
	9	5	8^s	17	20	30	12^i
	8	5	9	17	20	30	12

$s = 2$ is smaller than $k-1 = 3$ so we proceed with right part of the array:

	0	1	2	3	4	5	6
				17^s	20^i	30	12
				17^s	20	30	12^i
				17	12^s	30	20^i
				12	17	30	20

$s = 4$ is greater than $k-1 = 3$ so we proceed with left part of array, but there is only one element $A[3]$ and $s = 3$ is equal to $k-1 = 3$ so this our median.

4.5-3: Algorithm NonrecursiveQuickselect(A[l..r], k)

```
while(l < r)
    p← A[l]
    s← l
    for i← l+1 to r do
        if A[i]<p
            s ←s+1
            swap(A[s], A[i])
    swap(A[l], A[s])
    if S = k-1
        return A[s]
    else if s >l+k-1
        r← s-1
    else
        l← s+1
        k← K-1-s
```

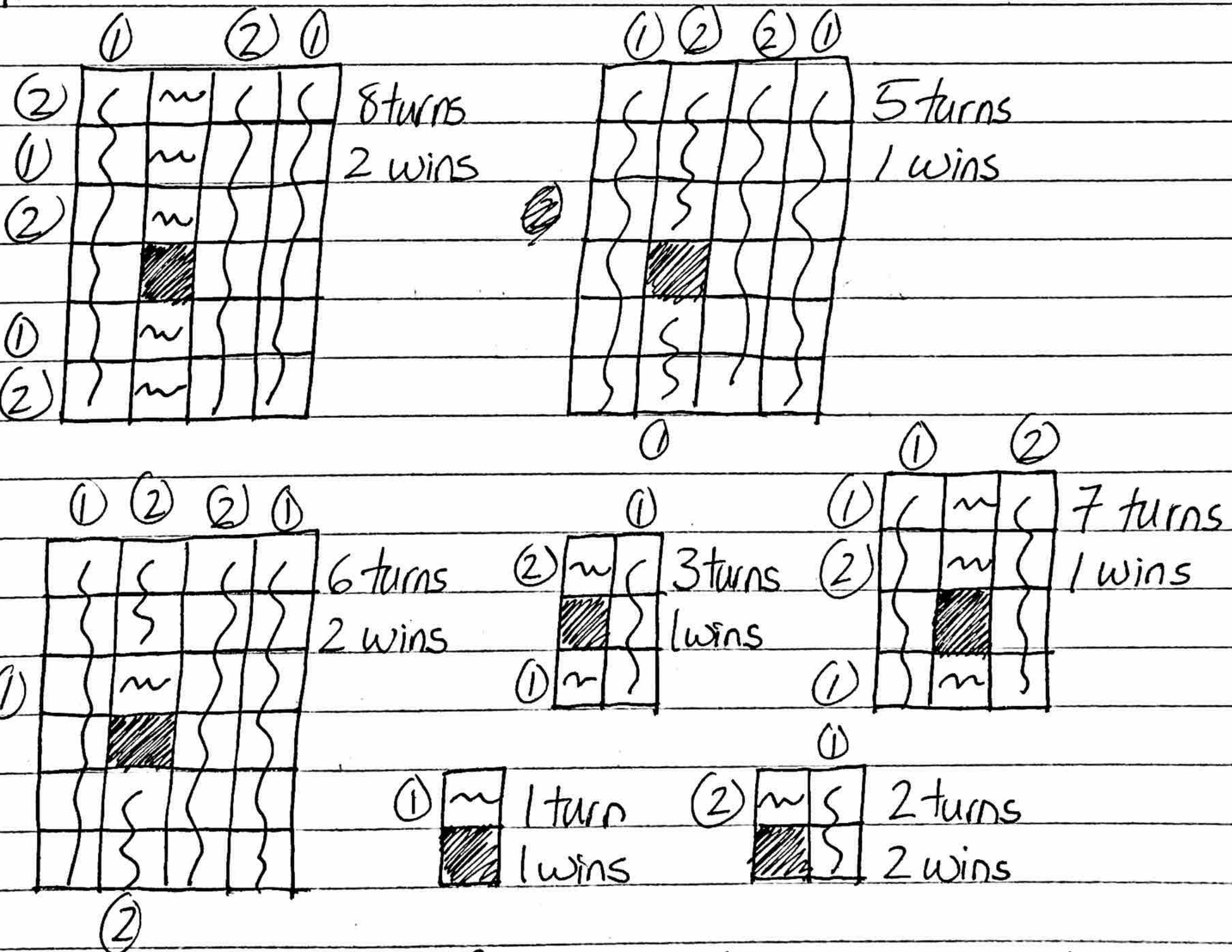
4.5-8: a. ~~Algorithm~~ Case 1: if the key is a leaf node mark its pointer as Null in its parent. Case 2: if the key is a node with one child then change the pointer in The key's parent node to point to the key's child node. case3: if the key is a node with two children then find the minimum value in The right child node's tree. Then switch these two nodes and delete the key using case 1 or 2.

This is not a variable-size-decrease algorithm because you cannot just reduce the problem by deleting the key from a smaller binary tree.

b. Finding the smallest value in the right subtree means transversing n-2 pointers so it is in $\theta(n)$.

4.5-10: $n=1 \Rightarrow$ first player loses
$2 \leq n \leq m+1 \Rightarrow$ first player wins by taking n-1 chips
$n = m+2 \Rightarrow$ first player loses
$m+3 \leq n \leq 2m+2 \Rightarrow$ first player wins by taking $(n-1) \bmod (m+1)$ chip
Therefore the first player will only loose if $n \bmod (m+1) = 1$
otherwise the first player wins by taking $(n-1) \bmod (m+1)$ chips.

4.5-11:



If the max number of possible turns are taken it is better to go second since the first player would only win in instances where $m=n+1$ or $n=m+1$. However, as the game proceeds moves can be

made that ensure less than the max number of moves occur in this game thus giving no side a clear advantage. In these cases the first player wins if an odd number of moves are made in the game, while the second player wins if an even number of moves are made in the game. The second player has some advantage because in their first move they can significantly impact how many subsequent moves can be made.

5.1-1: a. Algorithm PoLE(A[l..r])
if l=r
  return l
else
  m ← ⌊(l+r)/2⌋
  p1 ← PoLE(A[l..m])
  p2 ← PoLE(A[m+1..r])
  if A[p1] ≥ A[p2]
    return p1
  else
    return p2

b. If there are several elements of the largest value then the leftmost position will be returned.

c. $C(n) = C(\lceil n/2 \rceil) + C(\lfloor n/2 \rfloor) + 1 \quad n > 1, \quad C(1) = 0$

$C(2^k) = C(\lceil 2^k/2 \rceil) + C(\lfloor 2^k/2 \rfloor) + 1 = C(\lceil 2^{k-1} \rceil) + C(\lfloor 2^{k-1} \rfloor) + 1$

$= 2C(2^{k-1}) + 1 = 2(2C(2^{k-2}) + 1) + 1 = 2^2(2C(2^{k-3}) + 1) + 2 + 1$

$= 2^3 C(2^{k-3}) + 2^2 + 2 + 1$

$\Rightarrow 2^i((2^{k-i}) + 2^{i-1} + 2^{i-2} + \cdots + 1$

$C(2^k) = 2^k C(2^{k-k}) + 2^{k-1} + 2^{k-2} + \cdots + 1 = 2^k - 1 = n - 1$

$C(n) = n - 1$

d. The bruteforce algorithm also makes n-1 comparisons, but does not have the overhead from recursive calls

$T(n) = aT(n/b) + f(n) \qquad f(n) = n^d$

5.1-5: a. $T(n) = 4T(n/2) + n, \; T(1) = 1$

$a = 4, b = 2, d = 1 \Rightarrow 4 > 2^1 \Rightarrow T(n) \in (n^{\log_b a})$

$n^{\log_2 4} = n^{2 \log_2 2} = n^{2.1} \approx n^2 \Rightarrow T(n) \in (n^2)$

b. $T(n) = 4T(n/2) + n^2, \; T(1) = 1$

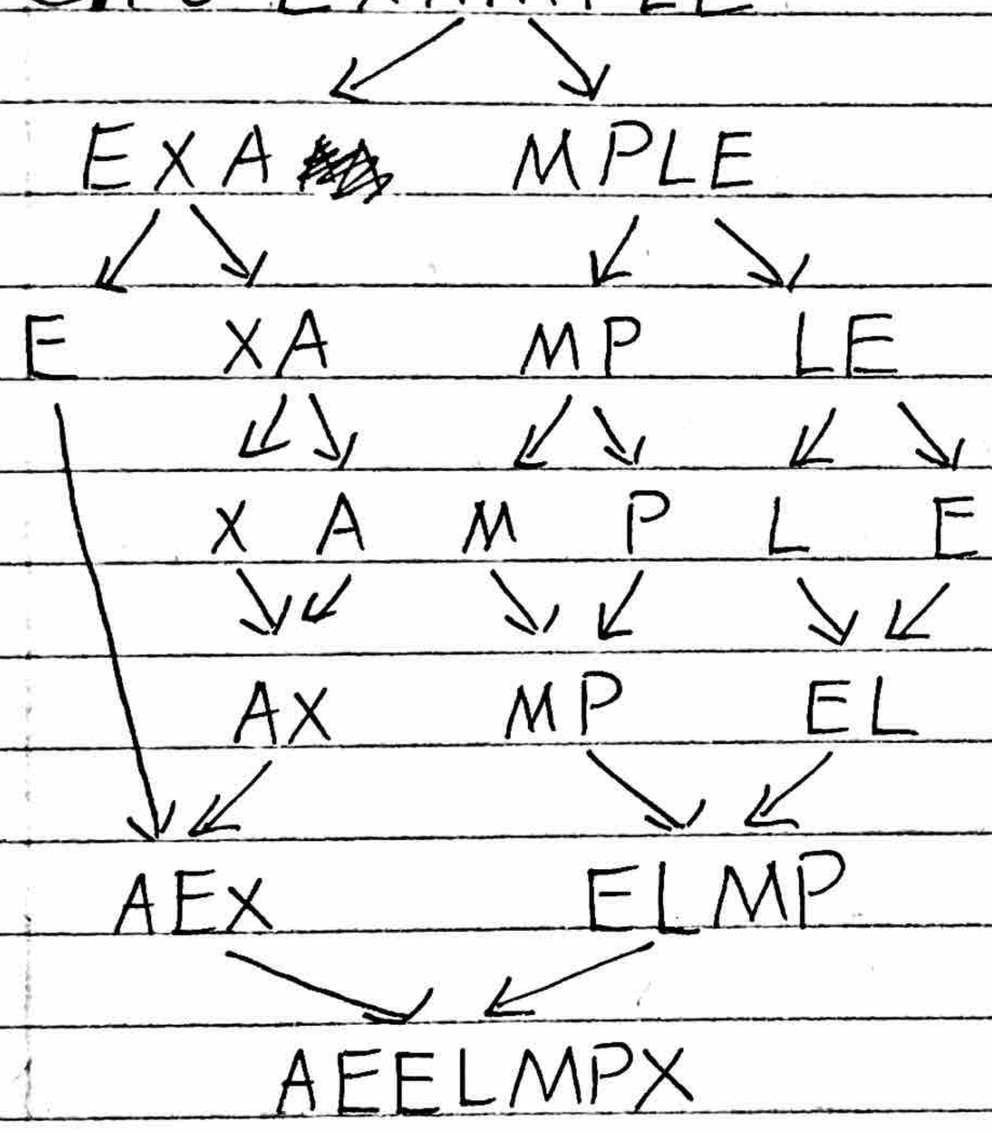
$a = 4, b = 2, d = 2 \Rightarrow 4 = 2^2 \Rightarrow T(n) \in (n^d \log n)$

$\Rightarrow T(n) \in (n^2 \log n)$

c. $T(n) = 4T(n/2) + n^3, \; T(1) = 1$

$a = 4, b = 2, d = 3 \Rightarrow 4 < 2^3 \Rightarrow T(n) \in (n^d) \Rightarrow T(n) \in (n^3)$

5.1-6: EXAMPLE



5.1-8: a. $C_{worst}(n) = 2C_{worst}(n/2) + n - 1, \quad n > 1, \; C_{worst}(1) = 0$

$C_{worst}(2^k) = 2C_{worst}(2^{k-1}) + 2^k - 1 = 2(2C_{worst}(2^{k-2}) + 2^{k-1} - 1)$

$= 2^2 C_{worst}(2^{k-2}) + 2 \cdot 2^k - 2 - 1 = 2^2(2C_{worst}(2^{k-3}) + 2^{k-2} - 1) + 2 \cdot 2^k - 2 - 1$

$= 2C_{worst}(2^{k-3}) + 3 \cdot 2^k - 2^2 - 2 - 1$

$\Rightarrow 2^i C_{worst}(2^{k-i}) + i 2^k - 2^{i-1} - 2^{i-2} - \dots - 1$

$C_{worst}(2^k) = 2^k \cancel{C_{worst}(2^{k-k})}^{\to 0} + k2^k - 2^{k-1} - 2^{k-2} - \dots - 1$

$= k2^k - (2^k - 1)$

$C_{worst} = n \log n - (n - 1)$

b. $C_{best}(n) = 2C_{best}(n/2) + n/2$, $n > 1$, $C_{best}(1) = 0$

$C_{best}(2^k) = 2C_{best}(2^{k-1}) + 2^{k-1} = 2(2C_{best}(2^{k-2}) + 2^{k-2}) + 2^{k-1}$

$= 2^2 C_{best}(2^{k-2}) + 2^{k-1} + 2^{k-1} = 2^2(2C_{best}(2^{k-3}) + 2^{k-3}) + 2^{k-1} + 2^{k-1}$

$= 2^3 C_{best}(2^{k-3}) + 2^{k-1} + 2^{k-1}$

$\Rightarrow 2^i C_{best}(2^{k-i}) + i2^{k-1}$

$C_{best}(2^k) = 2^k \cancel{C_{best}(2^{k-k})}^{\to 0} + k2^{k-1}$

$C_{best}(n) = \frac{n \log n}{2}$

c. $C_{worst}(n) = 2C_{worst}(n/2) + 2n$, $n > 1$, $C_{worst}(1) = 0$

$a = 2$, $b = 2$, $d = 0$ $\Rightarrow$ ~~$2 = 2^0$~~ $2 > 2^0 \Rightarrow T(n) \in \Theta(n^{\log_b a})$

$n^{\log_2 2} = n$

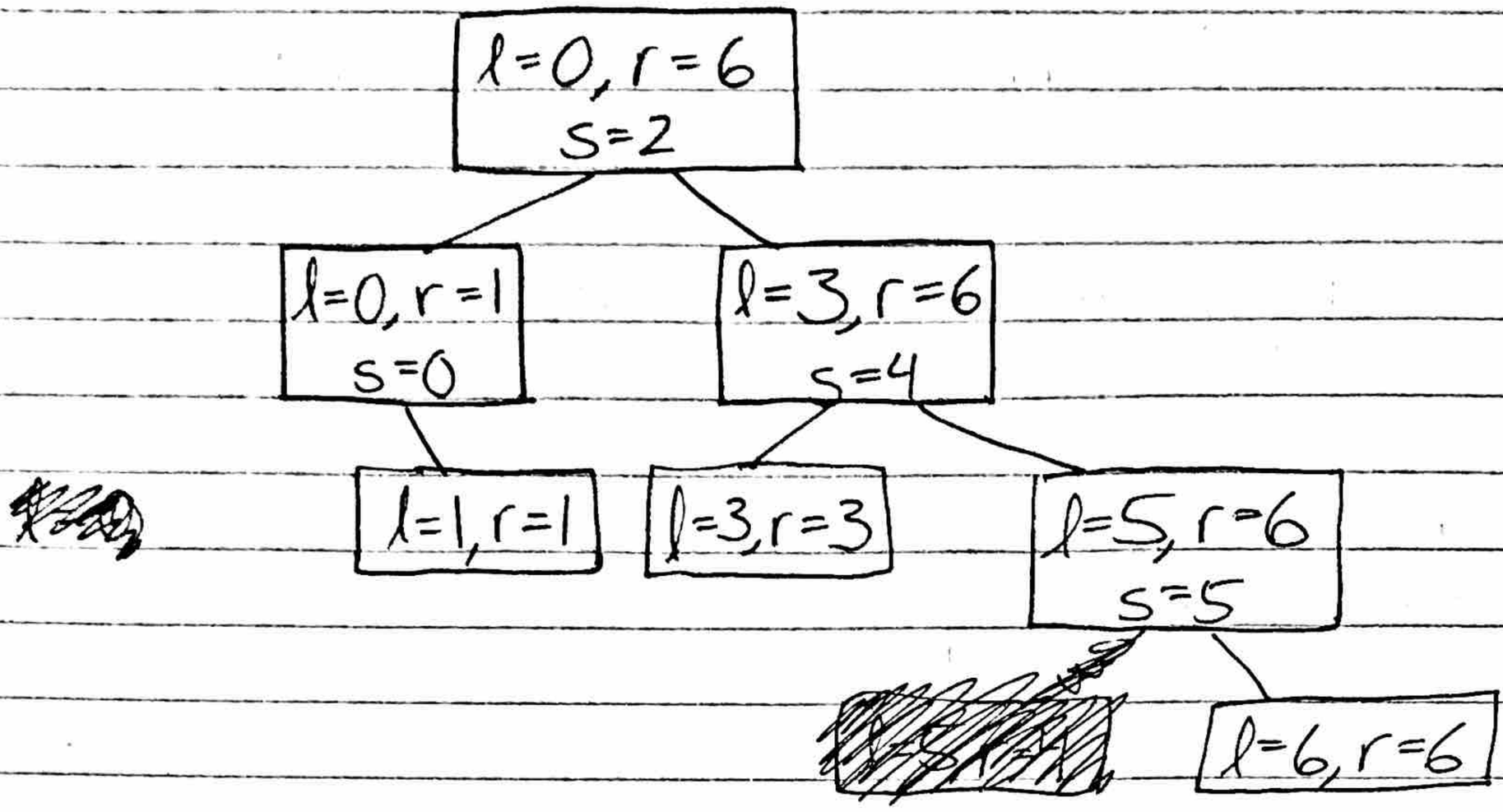
If $f(n) \in \Theta(n^d)$ with $d \geq 0$ then $T(n) \in \Theta(n^d \log n)$ if $a = b^d$ and would therefore have a worst case recurrence relation of $\Theta(n \log n)$. No, taking the number of key moves into account does not change the algorithm's efficiency class.

5.2-1:

0	1	2	3	4	5	6
E	X^i	A	M	P	L	E^j
F	E^i	A	M	P	L	X^j
E	E	A^j	M^i	P	L	X
A	E	E	M	P	L	X
A	E^{ij}					
A^j	E^i					
A	E					
	E					

5	6
P	X^{ij}
P^j	X^i
P	X
	X

M	P^i	L	X^j
M	P^i	L^j	X
M	L^i	P^j	X
M	L^j	P^i	X
L	M	P	X
L			



5.2-4: If an array of n elements with the leftmost position being the pivot then the left-to-right scan will go out of bounds if the pivot position holds the largest value in the array. Therefore a sentinel at the end of the array equal to the first element A[0] will stop the scan from going beyond position n. A single sentinel suffices for any input because when Partition(A[l..r]) is called for r<n-1 all the elements r are greater or equal to the elements in A[l..r] so A[r+1] will naturally be a sentinel to stop the scan going beyond r+1.

5.2-5: a. This is one of the worst case inputs since it will partition the array into a subarray of size 1 and another of size n-1. The recursive depth will be n and lead to an overall time $T(n) \in \Theta(n^2)$.
b. This is another of the worst case inputs and will partition sub arrays of size 1 and n-1. Leading to a recursive depth of n and overall time $T(n) \in \Theta(n^2)$.

5.2-9: a. Algorithm Dutch Flag (A[0..n-1])
i←0
j←0
r←n-1 mid← A[⌊n/2⌋-1]
~~while~~ j≤r
~~if A[j]~~
Encode(A[0..n-1]) //represents 'R' as 0, 'W' as 1, and 'B' as 2
while j≤r
if A[j] < mid
swap(A[i], A[j])
i←i+1
j←j+1
else if A[j] > mid
swap(A[j], A[r])
r←r-1
else
j←j+1
Decode(A[0..n-1]) //returns 0,1,2's to their equivalent characters
b. This ~~can~~ solution to the Dutch national flag problem can be used in quicksort for three way partitioning which is much faster.

5.3-1: Algorithm Levels(T)
if T=∅
return 0
else
return max{Levels(T_left), Levels(T_right)}+1

Levels(T) is invoked for all n nodes in the tree and max{} has a constant run time so the time efficiency is Θ(n)+C = Θ(n).

5.3-3: Yes, we can find the height using a queue that results in the same time efficiency $\Theta(n)$.

```
Algorithm Height(T)
if (T == ∅)
    return 0
q.push(T)
height ← 0
while(1)
    count ← q.size
    if count = 0
        return height
    height ← height + 1
    while (count > 0)
        n ← q.front
        q.pop()
        if(n->left != ∅)
            q.push(n->left)
        if(n->right != ∅)
            q.push(n->right)
        count--
```

5.3-5: a. a, b, d, e, c, f
b. d, b, e, a, c, f
c. d, e, b, f, c, a

5.3-9: $x = n+1$, $E_l = I_l + 2n_l$, $E_r = I_r + 2n_r$

$$
\begin{aligned}
E &= (E_l + x_l) + (E_r + x_r) \\
&= (I_l + 2n_l + x_l) + (I_r + 2n_r + x_r) \\
&= ((I_l + n_l) + (I_r + n_r)) + (n_l + n_r) + (x_l + x_r) \\
&= I + (n-1) + x \\
&= I + 2n
\end{aligned}
$$

$c_2 = a_1 * b_1$

$c_0 = a_0 * b_0$

$c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$

$c = a * b = c_2 10^2 + c_1 10^1 + c_0$

5.4-2: $2101 * 1130 = c_2 10^2 + c_1 10^1 + c_0$

$c_2 = 21 * 11$ $\quad c_0 = 01 * 30$ $\quad c_1 = \underbrace{(21+01) * (11+30)}_{22*41} - (c_2 + c_0)$

$C = 21 * 11 \Rightarrow c_2 = 2*1 = 2,\ c_0 = 1*1 = 1,\ c_1 = (2+1)*(1+1) - (2+1) = 3$

$\Rightarrow 21 * 11 = 231$

$01 * 30 \Rightarrow c_2 = 0*3 = 0,\ c_0 = 1*0 = 0,\ c_1 = (0+1)*(3+0) - (0+0) = 3$

$\Rightarrow 01 * 30 = 30$

$22 * 41 \Rightarrow c_2 = 2*4 = 8,\ c_0 = 2*1 = 2,\ c_1 = (2+2)*(4+1) - (8+2) = 10$

$22 * 41 = 902$

$2101 * 1130 = 231 \times 10^4 + (902 - 231 - 30) \times 10^2 + 30$

$= 2310000 + 64100 + 30$

$= 2374130$

5.4-6: $\begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} * \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix} = \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$

$m_1 + m_4 - m_5 + m_7 = (a_{00} + a_{11})(b_{00} + b_{11}) + a_{11}(b_{10} - b_{00}) - (a_{00} + a_{01}) b_{11} +$
$(a_{01} - a_{11})(b_{10} + b_{11}) = a_{00}b_{00} + a_{11}b_{00} + a_{00}b_{11} + a_{11}b_{11} + a_{11}b_{10} - a_{11}b_{00}$
$- a_{01}b_{11} + a_{01}b_{10} - a_{11}b_{10} + a_{01}b_{11} - a_{11}b_{11} = a_{00}b_{00} + a_{01}b_{10}$

$m_3 + m_5 = a_{00}(b_{01} - b_{11}) + (a_{00} + a_{01}) b_{11} = a_{00}b_{01} - a_{00}b_{11} + a_{00}b_{11} + a_{01}b_{11}$
$= a_{00}b_{01} + a_{01}b_{11}$

$m_2 + m_4 = (a_{10} + a_{11}) b_{00} + a_{11}(b_{10} - b_{00}) = a_{10}b_{00} + a_{11}b_{00} + a_{11}b_{10} - a_{11}b_{00}$
$= a_{10}b_{00} + a_{11}b_{10}$

$m_1 + m_3 - m_2 + m_6 = (a_{00} + a_{11})(b_{00} + b_{11}) + a_{00}(b_{01} - b_{11}) - (a_{10} + a_{11}) b_{00}$
$+ (a_{10} - a_{00})(b_{00} + b_{01}) = a_{00}b_{00} + a_{11}b_{00} + a_{00}b_{11} + a_{11}b_{11} + a_{00}b_{01} - a_{00}b_{11}$
$- a_{10}b_{00} - a_{11}b_{00} + a_{10}b_{00} - a_{00}b_{00} + a_{10}b_{01} - a_{00}b_{01} = a_{10}b_{01} + a_{11}b_{11}$

5.4-8: $A(n) = 7A(n/2) + 18(n/2)^2, \quad n > 1, \quad A(1) = 0$

$A(2^k) = 7A(2^{k-1}) + \frac{9}{2}4^k = 7[7A(2^{k-2}) + \frac{9}{2}4^{k-1}] + \frac{9}{2}4^k$

$= 7^2 A(2^{k-2}) + 7\frac{9}{2}4^{k-1} + \frac{9}{2}4^k = 7^3 A(2^{k-3}) + 7^2\frac{9}{2}4^{k-2} + 7\frac{9}{2}4^{k-1} + \frac{9}{2}4^k$

$\Rightarrow 7^k \cancel{A(2^{k-k})}^{0} + \frac{9}{2}\sum_{i=0}^{k-1} 7^i 4^{k-i} = \frac{9}{2}4^k \sum_{i=0}^{k-1} (7/4)^i$

$= \frac{9}{2}4^k \frac{(7/4)^k - 1}{(7/4) - 1} = 6(7^k - 4^k) = 6(7^{\log_2 n} - 4^{\log_2 n})$

$= 6(n^{\log_2 7} - n^2)$

6.1-3: a. Algorithm MaxMin(A[0..n-1])
//use insertion sort to sort elements of array
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
max ← A[n-1]
min ← A[0]
return min, max

$C_{worst}(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} \in \theta(n^2)$

b. The brute-force and divide-and-conquer algorithms for finding the largest and smallest elements scan the array only one time and thus have a linear efficiency $\theta(n)$, which is more efficient than the presorting-based algorithm.

6.1-4: mergesort $\in \theta(n \log n)$, binarysearch $\in \theta(\log_2 n)$, sequential search $\in \theta(n/2)$

$n \log_2 n + k \log_2 n \leq kn/2$

$\Rightarrow k \geq \frac{n \log_2 n}{n/2 - \log_2 n}$

For $n = 10^3 \Rightarrow k \geq \frac{10^3 \log_2 10^3}{10^3/2 - \log_2 10^3} \Rightarrow k \geq 20.337$, $k = 21$

For $n = 10^6 \Rightarrow k \geq \frac{10^6 \log_2 10^6}{10^6/2 - \log_2 10^6} \Rightarrow k \geq 39.865$, $k = 40$

6.1-7: Algorithm IsSum (A[0..n-1], int s)
mergesort(A[0..n-1])
i ← 0
j ← n-1
while i < j
  if A[i]+A[j]=s
    return 1
  else if A[i]+A[j]<s
    i ← i+1
  else
    j ← j-1

~~T(n) = Cmerge(n) + Csearch(n) = nlogn~~

$T(n) = C_{merge}(n) + C_{search}(n) = n\log n + n/2 \Rightarrow T(n) \in \theta(n)$

6.2-1:
$$\begin{bmatrix}1&1&1&2\\2&1&1&3\\1&-1&3&8\end{bmatrix} \to \begin{bmatrix}1&0&0&1\\2&1&1&3\\1&-1&3&8\end{bmatrix} \to \begin{bmatrix}1&0&0&1\\2&1&1&3\\3&0&4&11\end{bmatrix}$$
$$\to \begin{bmatrix}1&0&0&1\\2&1&1&3\\0&0&4&8\end{bmatrix} \to \begin{bmatrix}1&0&0&1\\2&1&1&3\\0&0&1&2\end{bmatrix} \to \begin{bmatrix}1&0&0&1\\0&1&1&1\\0&0&1&2\end{bmatrix} \to \begin{bmatrix}1&0&0&1\\0&1&0&-1\\0&0&1&2\end{bmatrix}$$

6.2-2: a.
$$L = \begin{bmatrix}1&0&0\\2&1&0\\1&-2&1\end{bmatrix} \quad U = \begin{bmatrix}1&1&1\\0&1&1\\0&0&4\end{bmatrix}$$
$$\begin{bmatrix}1&0&0\\2&1&0\\1&-2&1\end{bmatrix}\begin{bmatrix}y_1\\y_2\\y_3\end{bmatrix} = \begin{bmatrix}2\\3\\8\end{bmatrix} \Rightarrow \begin{aligned} y_1 &= 2 \\ y_2 &= 3-2y_1 = 3-2(2) = -1 \\ y_3 &= 8-y_1+2y_2 = 8-2+2(-1) = 4\end{aligned}$$

$$\begin{bmatrix}1&1&1\\0&1&1\\0&0&4\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix} = \begin{bmatrix}2\\-1\\4\end{bmatrix} \Rightarrow \begin{aligned} x_1 &= 2-x_2-x_3 = 2-(-2)-1 = 3 \\ x_2 &= -1-x_3 = -1-1 = -2 \\ x_3 &= 4/4 = 1\end{aligned}$$

6.2-5: Algorithm BackSubstitution(A[1..n, 1..n], b[1..n])
b[n] ← b[n] / A[n,n]
A[n,n] ← 1
for i ← n-1 to 1
    for j ← n to i+1
        b[i] ← b[i] - b[j] * A[i,j]
    b[i] ← b[i] / A[i,i]
    A[i,i] ← 1
return b

$$M(n) = \sum_{i=n-1}^{1} \sum_{j=n}^{i+1} 1 = \sum_{i=n-1}^{1} i \approx \frac{n(n+1)}{2} \cdot \in \Theta(n^2)$$

6.2-6: $D_G(n) = M_G(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{k=i}^{n+1} 1 \approx \frac{1}{3} n^3$

$D_{BG}(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 \approx \frac{1}{2} n^2$

$M_{BG}(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{k=i}^{n+1} 1 \approx \frac{1}{3} n^3$

$$\frac{T_G(n)}{T_{BG}(n)} = \frac{t_d D_G(n) + t_m M_G(n)}{t_d D_{BG}(n) + t_m M_{BG}(n)} = \frac{C_d \frac{1}{3} n^3 + C_m \frac{1}{3} n^3}{C_d \frac{1}{2} n^2 + C_m \frac{1}{3} n^3}$$

$$= \frac{C_d \frac{1}{3} n^3 + C_m \frac{1}{3} n^3}{C_m \frac{1}{3} n^3} = \frac{C_d + C_m}{C_m} = \frac{C_d}{C_m} + 1 = \frac{3C_m}{C_m} + 1 = 4$$

Better Gauss Elimination is 4 times faster than Gaussian Elimination.

6.2-10: a. $x_1 = \frac{\det A_1}{\det A}$, $x_2 = \frac{\det A_2}{\det A}$, $x_3 = \frac{\det A_3}{\det A}$

$\det A = 1(3-(-1)) - 1(6-1) + 1(-2-1) = 4-5-3 = -4$

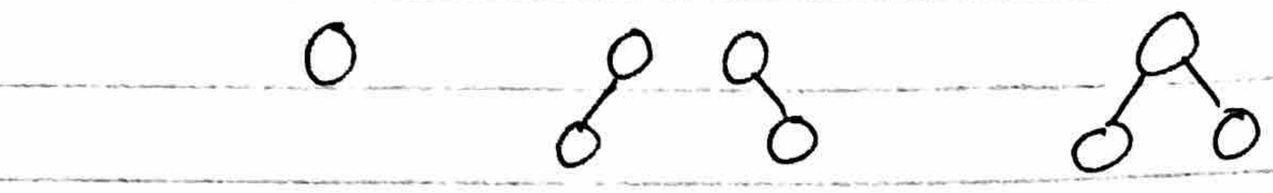
$\det A_1 = 2(3-(-1)) - 1(9-8) + 1(-3-8) = 8-1-11 = -4$

$\det A_2 = 1(9-8) - 2(6-1) + 1(16-3) = 1-10+13 = 4$

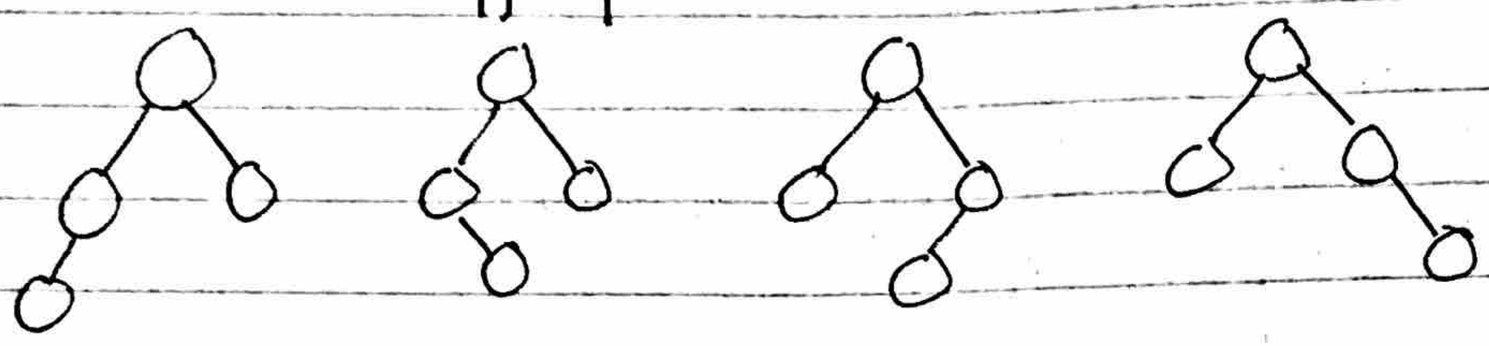
$\det A_3 = 1(8-(-3)) - 1(16-3) + 2(-2-1) = 11-13-6 = -8$

$x_1 = \frac{-4}{-4} = 1$, $x_2 = \frac{4}{-4} = -1$, $x_3 = \frac{-8}{-4} = 2$

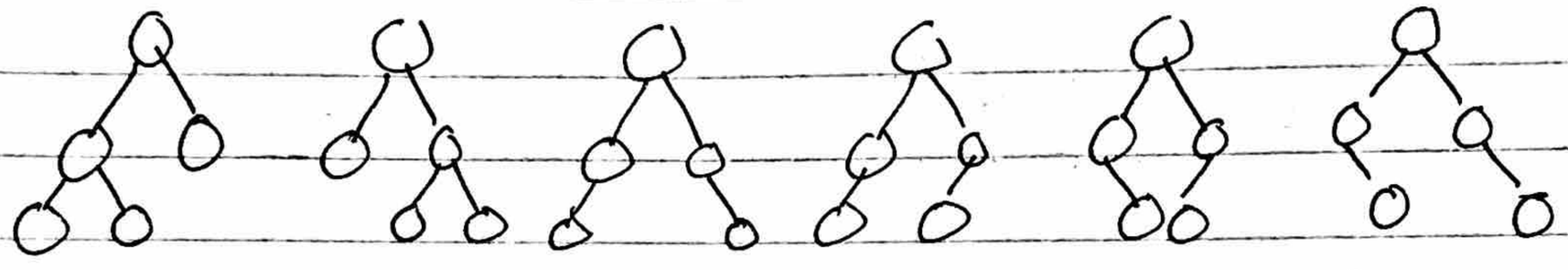
6.3-2: a. n=1 n=2 n=3



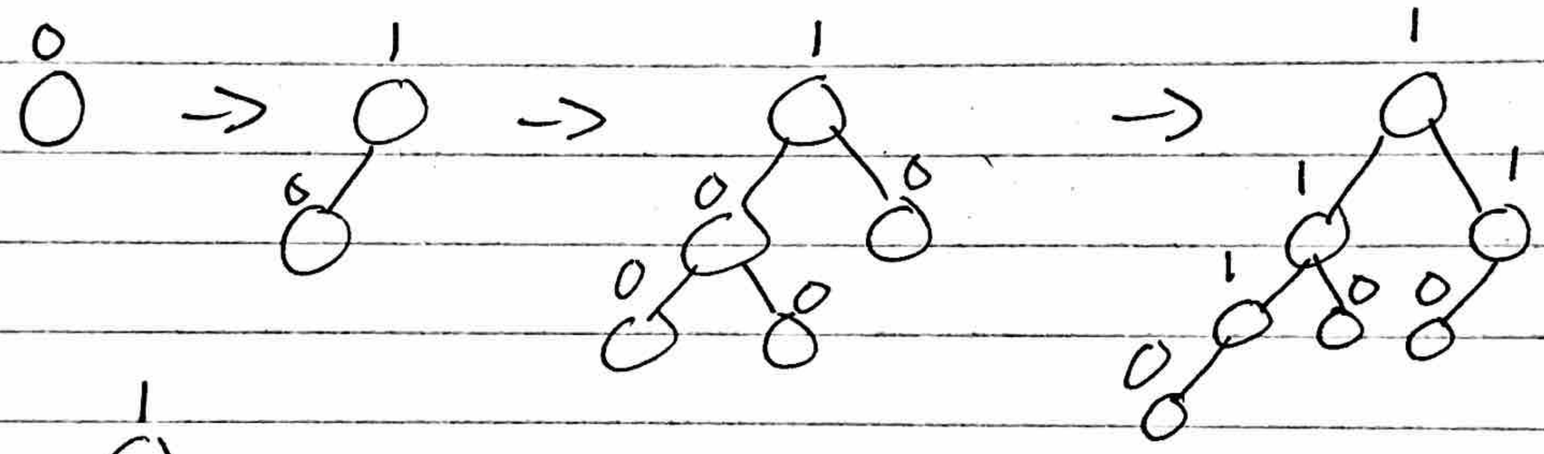
n=4

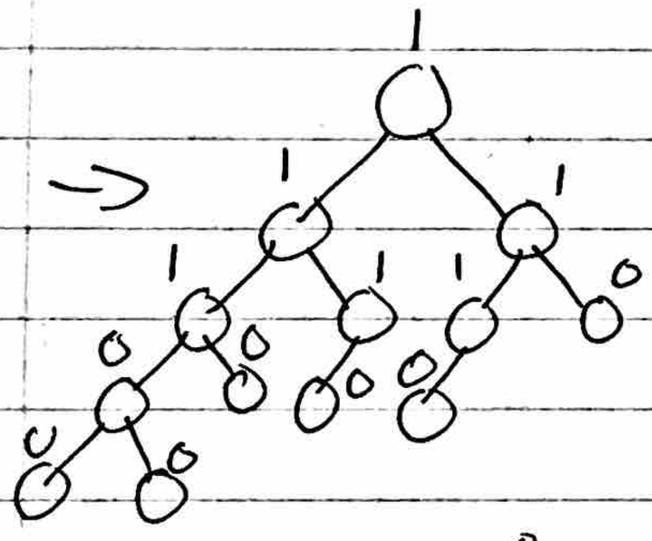


n=5



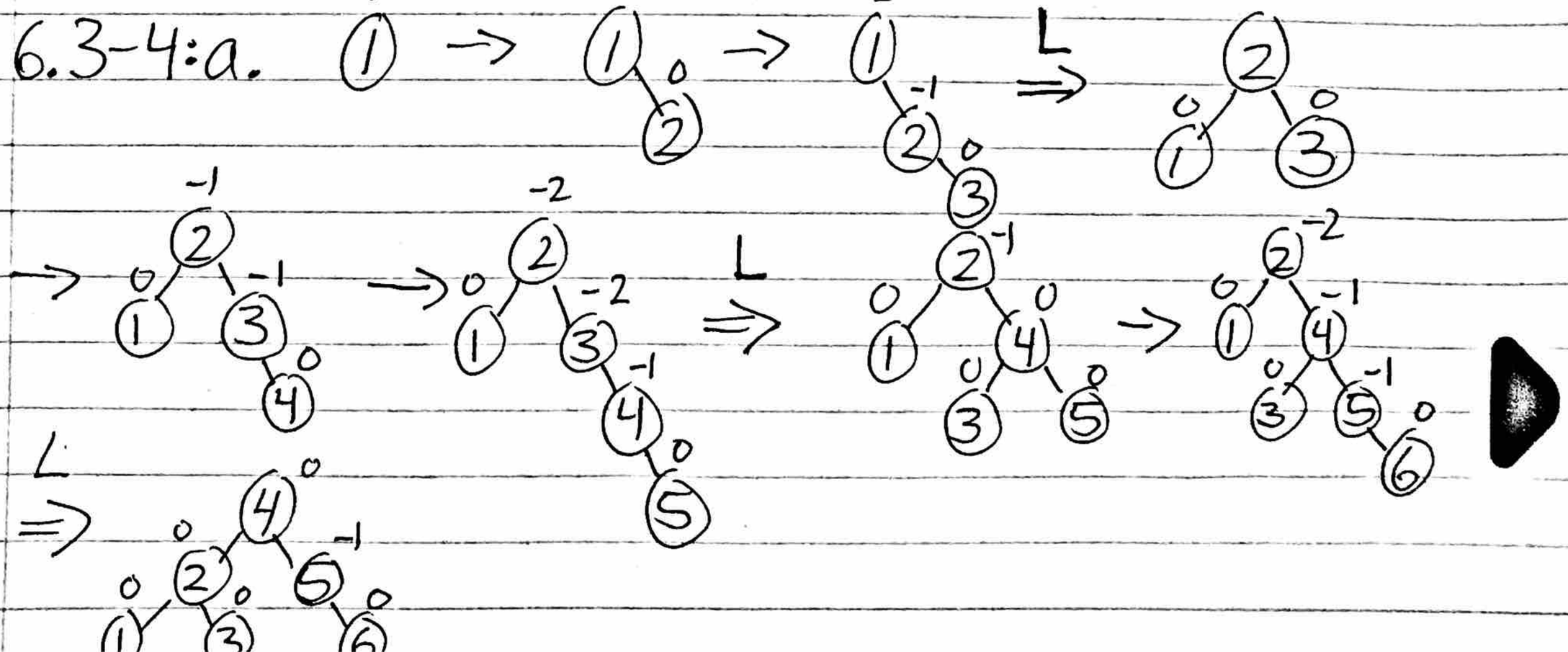
b.





0 -1 -2 0

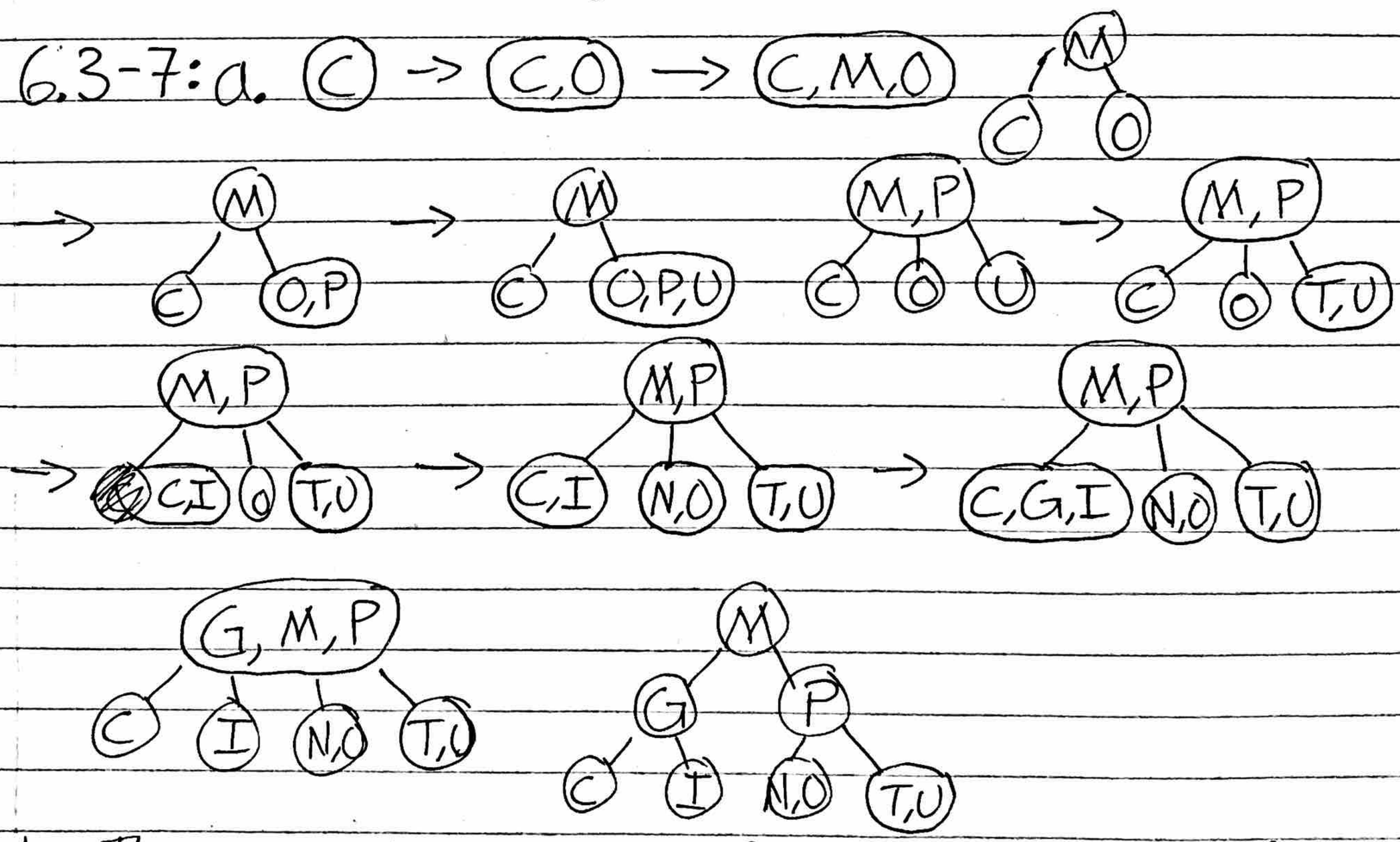
6.3-4: a.



6.3-5: a. Algorithm Range (root)
temp ← root
~~temp ← root~~
while (temp.left != ∅) do
    temp ← temp.left
min ← temp.data
~~while~~ temp ← root
while(temp.right != ∅) do
    temp ← temp.right
max ← temp.data
range ← max - min
return range

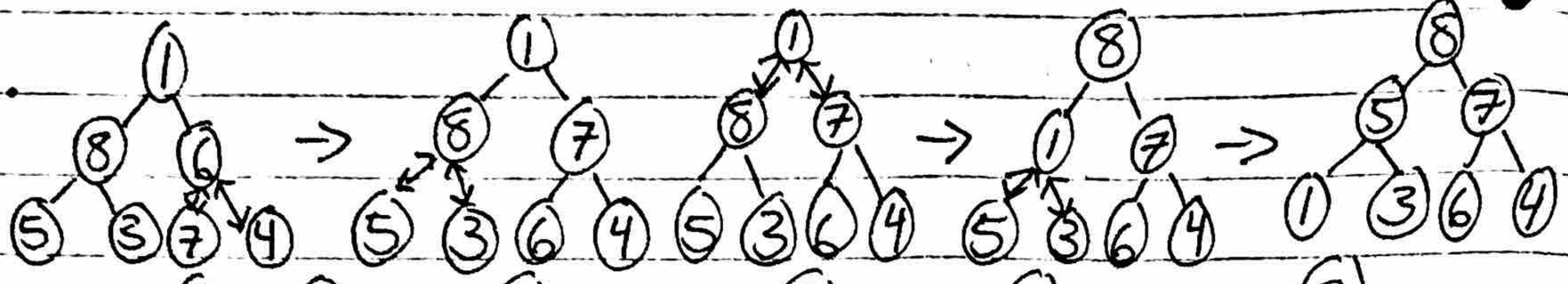
↓ subtraction overhead
$T(n) = \theta(\log n) + \theta(\log n) + \theta(c) = \theta(\log n)$

6.3-7: a. (C) → (C,O) → (C,M,O)

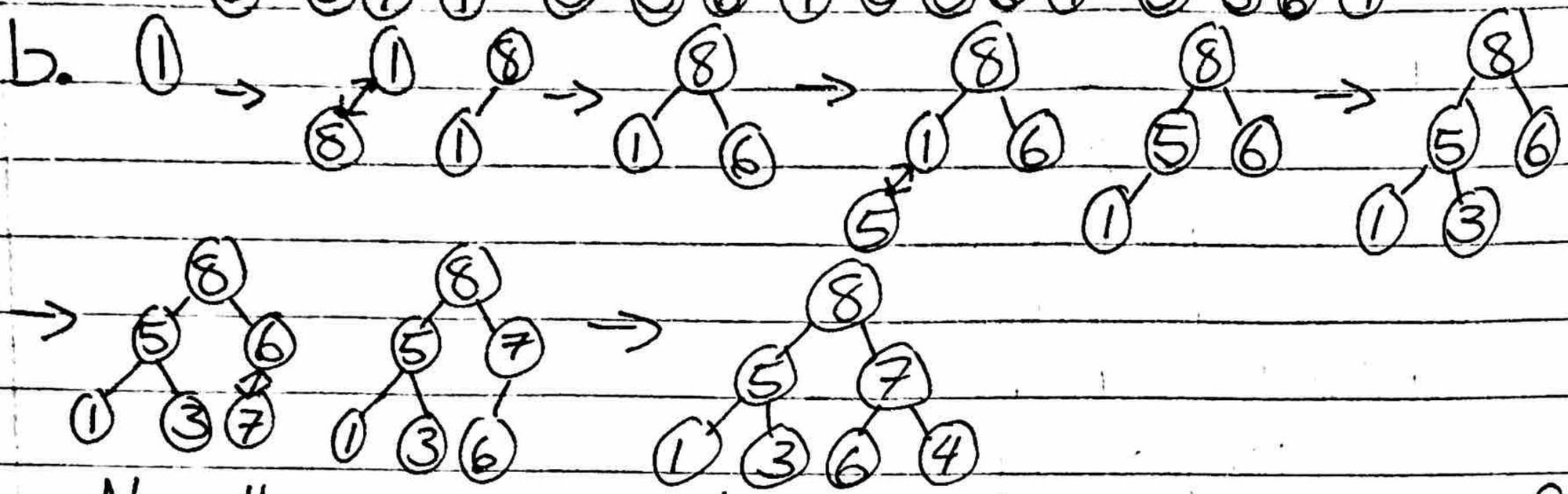


b. The largest number of comparisons is 4 for O & U.
$C_{avg} = \frac{1}{9}C(C) + \frac{1}{9}C(O) + \frac{1}{9}C(M) + \frac{1}{9}C(P) + \frac{1}{9}C(U) + \frac{1}{9}C(T) + \frac{1}{9}C(I) + \frac{1}{9}C(N) + \frac{1}{9}C(G) = \frac{1}{9}(3) + \frac{1}{9}(4) + \frac{1}{9}(1) + \frac{1}{9}(2) + \frac{1}{9}(4) + \frac{1}{9}(3) + \frac{1}{9}(3) + \frac{1}{9}(3) + \frac{1}{9}(2)$
$= 25/9 = 2.\overline{78}$

6.4-1:a.

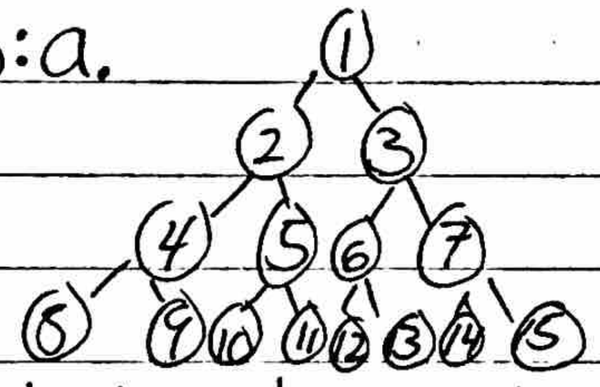


b.



c. No, there is no rules on the magnitudes of children nodes besides being less than their parents so these two algorithms can produce different orderings for them without violating the requirements of a heap.

6.4-3:a.



h=0 n=1
h=1 n=2-3
h=2 n=4-7
h=3 n=8-15

smallest: 2^h largest: $2^{h+1}-1$

b. $\lfloor \log_2(1) \rfloor = \lfloor 0 \rfloor = 0$ ✓

$\lfloor \log_2(2) \rfloor = \lfloor 1 \rfloor = 1$ ✓ $\lfloor \log_2(3) \rfloor = \lfloor 1.58 \rfloor = 1$ ✓

$\lfloor \log_2(4) \rfloor = \lfloor 2 \rfloor = 2$ ✓ $\lfloor \log_2(7) \rfloor = \lfloor 2.81 \rfloor = 2$ ✓

$\lfloor \log_2(8) \rfloor = \lfloor 3 \rfloor = 3$ ✓ $\lfloor \log_2(15) \rfloor = \lfloor 3.91 \rfloor = 3$ ✓

Therefore $h = \lfloor \log_2 n \rfloor$.

6.4-5:a. Minimum Key Deletion for a heap

Step1: Sequentially search $H[\lfloor n/2 \rfloor + 1..n]$ for position min that holds the smallest key

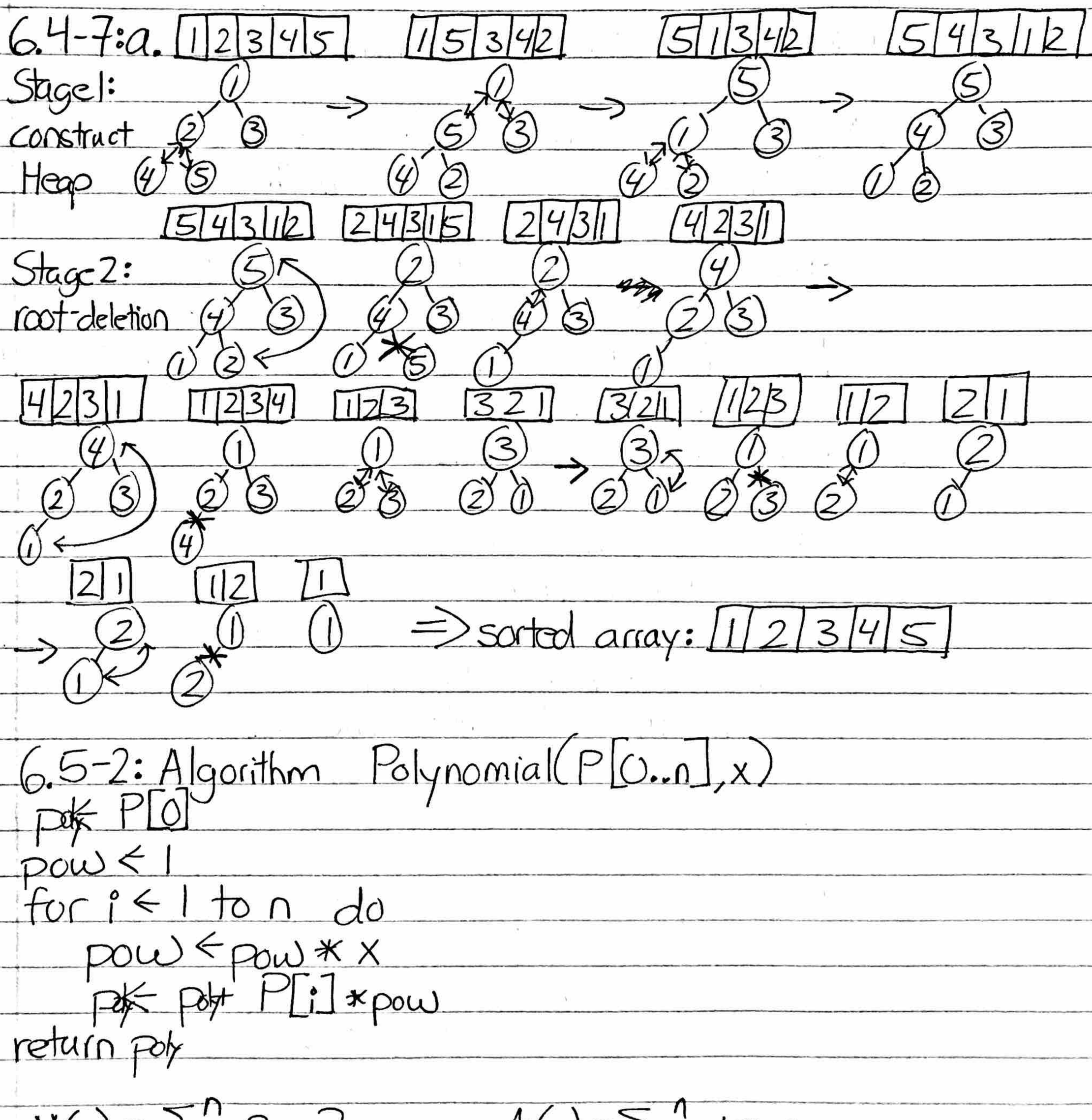
step2: Swap H[min] and H[n]

step3: Delete H[n] and decrease heap size by 1

step4: Heapify and shift H[min] as needed to satisfy the parental dominance property.

The time efficiency of searching for the smallest element in H[⌊n/2⌋+1..n] is O(⌊n/2⌋) which is of class O(n) and deleting the minimum key has a constant over head c, so the overall efficiency is: $\Theta(\log n) + \Theta(c) = \Theta(\log n)$.

6.4-7:a.



6.5-2: Algorithm Polynomial(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
```

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

6.5-3: a.(i) Horner = $M(n)+A(n)=M(n)=n$

Brute-force = $M(n)+A(n)=M(n)=2n$

$\frac{\text{Brute-force}}{\text{Horner}} = \frac{2n}{n} = 2$ Horner's is twice as fast!

(ii) Horner = $M(n)+A(n)=n+n=2n$

Brute-force = $M(n)+A(n)=2n+n=3n$

$\frac{\text{Brute-force}}{\text{Horner}} = \frac{3n}{2n} = 3/2$ Horner's is 3/2 times faster!

b. No, because Horner's rule doesn't use any more memory than the brute-force method.

6.5-4: a.

coeff:	3	-1	0	2	5
p: x=-2		(-2)(3)+(-1)=-7	(-2)(-7)+0=14	(-2)(14)+2=-26	(-2)(-26)+5=57

b. The quotient of the division of $p(x)=3x^4-x^3+2x+5$ by $x+2$ is $3x^3-7x^2+14x-26$ with a remainder of 57.

6.6-2: In a max-heap a parent node has a value greater than or equal to the values of its children. Therefore to produce a min-heap using the max-heap algorithm we simply reverse this property so that the children of a node are less ~~than~~ greater than or equal to their parent node.

6.6-4: a. Algorith Cycle3(A)

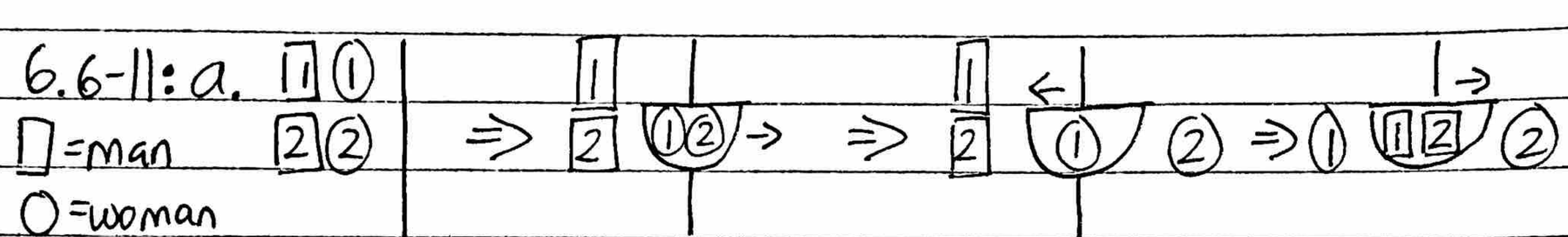
```
for i<=0 to n
    for j<=0 to n
        if(A[i,j]!=0 & A^2[i,j]>0)
            return true
return false
```

$\in \Theta(n^2)$

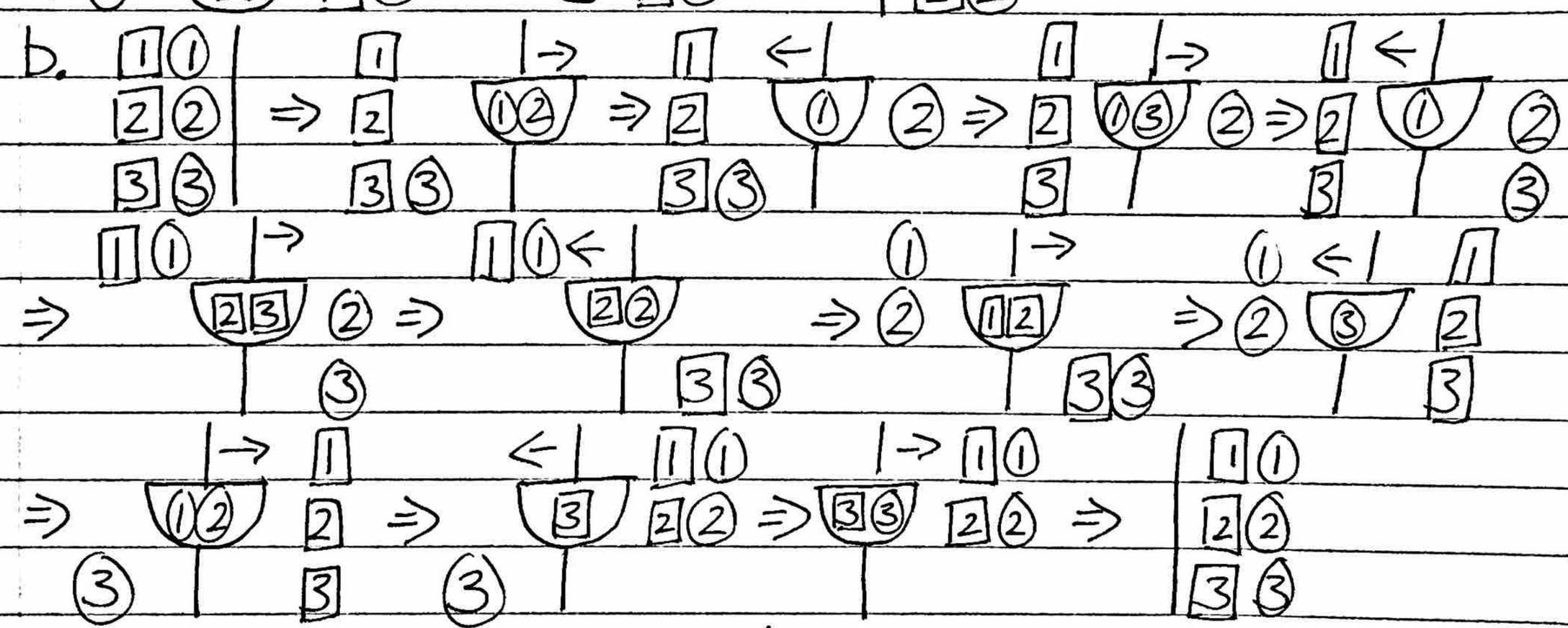
b. No, since it is possible to have a cycle of length 3 without having a back edge to the vertex's grandparent, such as a-b-e-a.

6.6-9: The edge-coloring problem can be reduced to a vertex-coloring problem by making a new graph where the vertices represent the edges of the original graph. We then connect two vertices in the new graph by an edge only in the case where these vertices represented two edges with a common endpoint in the original graph.

6.6-11: a. [1](1) [2](2)
□=man
○=woman



b. [1](1) [2](2) [3](3)



c. This problem does not have a solution for $n \geq 4$ because there would always occur an instance where at least one wife is on a side without her husband where other husbands are present.