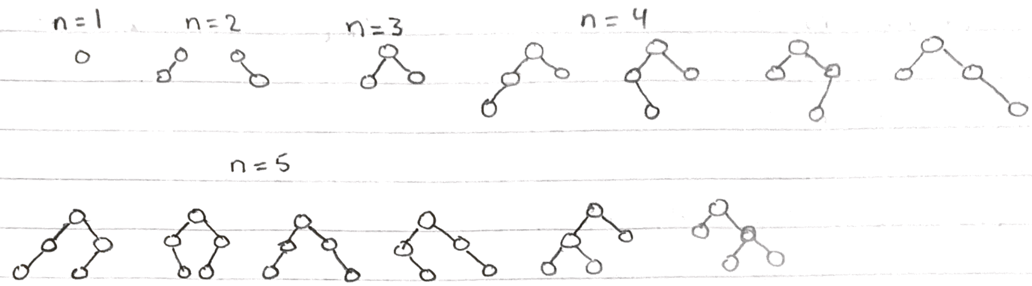
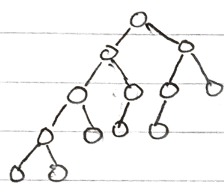
Homework 6

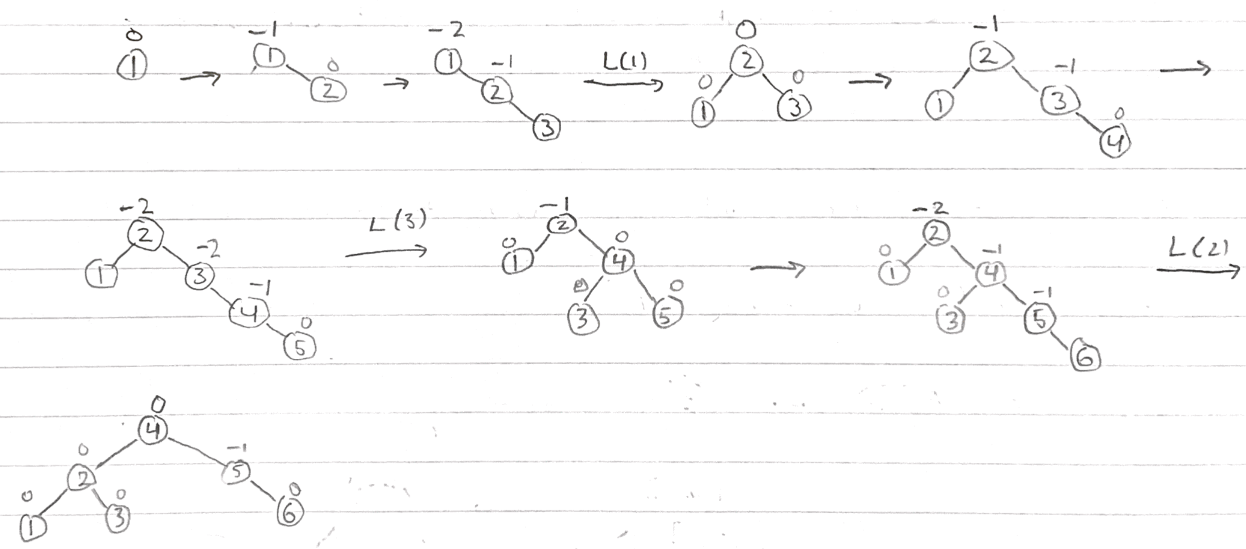
6.3 2a.



b.



4a. 1, 2, 3, 4, 5, 6

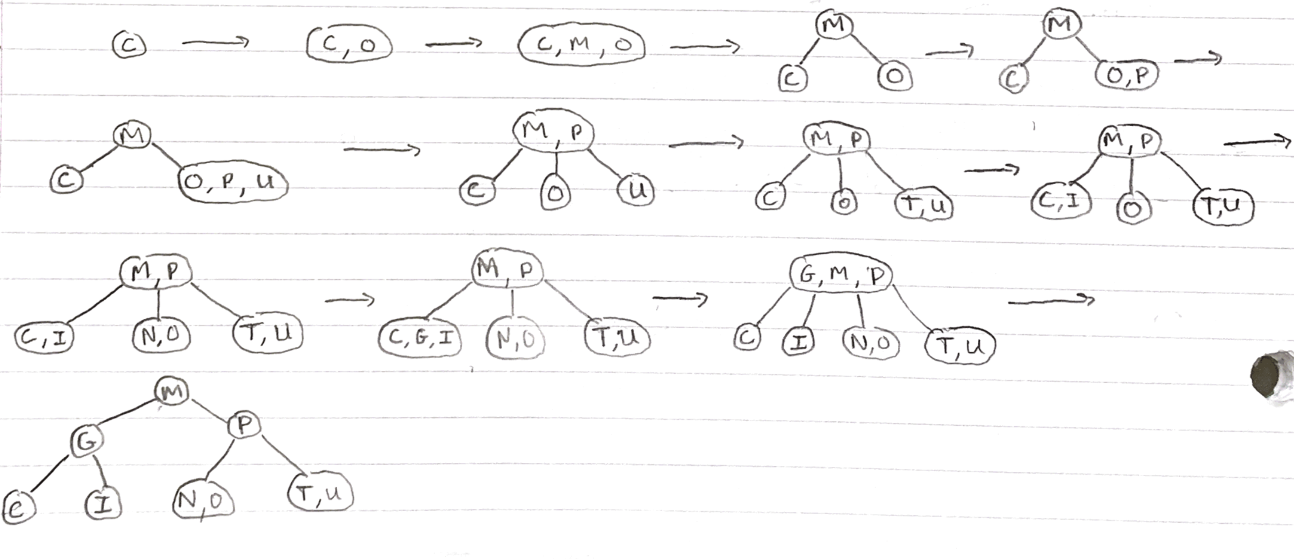


5a. Algorithm AVLRange(root)

```
    // Input: root node of AVL tree
    // Output: the range of the tree
    curr ← root
    while (curr.left != NULL) do
        curr ← curr.left
    min ← curr.data
    curr ← root
    while (curr.right != NULL) do
        curr ← curr.right
    max ← curr.data
    return (max - min)
```

For the worst case, the algorithm will traverse the last level on the leftmost node of the tree to find and set the smallest value to a variable and the algorithm will traverse to the last level on rightmost nodes of the tree to find and set the largest value. Then the algorithm will return the difference of the largest and smallest value to give the range. Thus the worst case efficiency is $\Theta(\log n) + \Theta(\log n) + \Theta(c)$, where c is some constant. Therefore worst case efficiency for this algorithm is $\Theta(\log n)$.

7a.



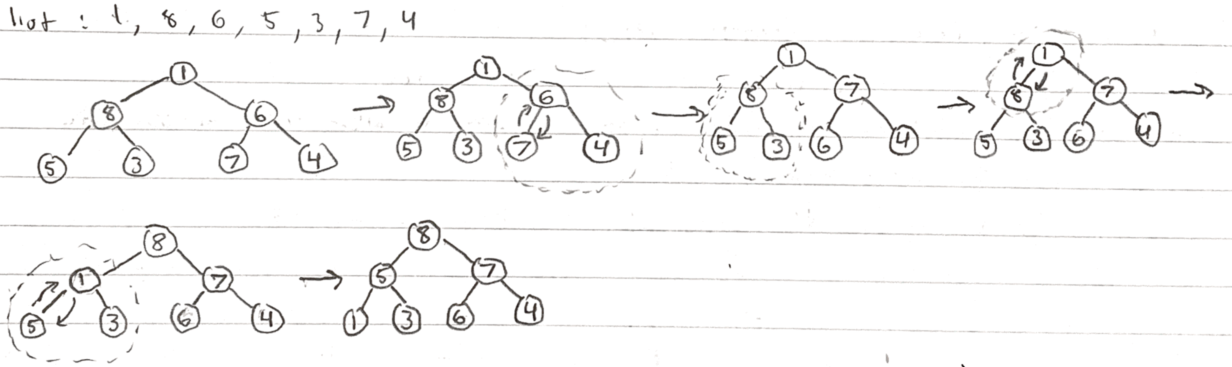
b. The largest number of key comparisons in a successful search are O and U since they are the second elements in a 2 node that is is the last level of the tree. Thus the largest number of key comparisons in a successful search is 4.

Since the probability for searching each key is equal, then the probability of each key getting searched is 1/9, then the average number of key comparisons are

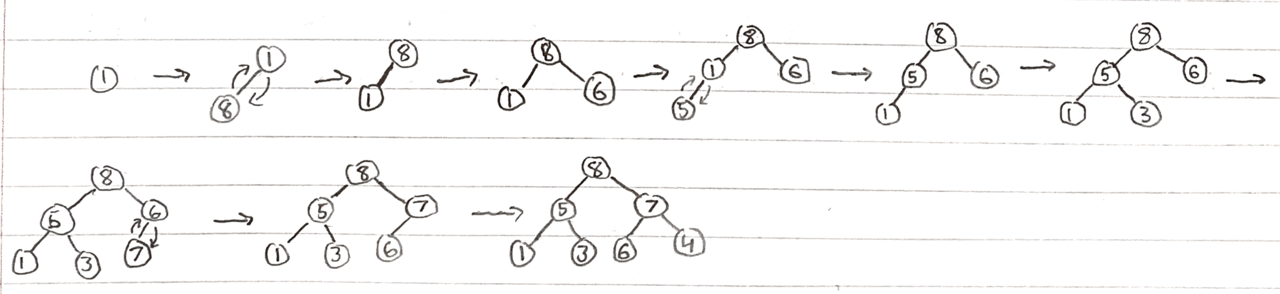
$$\tfrac{1}{9}(C)+\tfrac{1}{9}(O)+\tfrac{1}{9}(M)+\tfrac{1}{9}(P)+\tfrac{1}{9}(U)+\tfrac{1}{9}(T)+\tfrac{1}{9}(I)+\tfrac{1}{9}(N)+\tfrac{1}{9}(E)$$
$$=\tfrac{1}{9}(3)+\tfrac{1}{9}(4)+\tfrac{1}{9}(1)+\tfrac{1}{9}(2)+\tfrac{1}{9}(4)+\tfrac{1}{9}(3)+\tfrac{1}{9}(3)+\tfrac{1}{9}(3)+\tfrac{1}{9}(2)$$
$$= 25/9$$
$$= 2.8$$

Average number of key comparisons in successful search is 2.8

6.4 1a. list : 1, 8, 6, 5, 3, 7, 4



b.



c. It is not always true that the bottom-up and top-down algorithms yield the same heap for the same input.

3a. number of nodes in a complete tree is $2^{h+1}-1$. Thus $max(h) = \sum_{i=0}^{h+1} 2^i - 1 = (2^{h+1}-1)$. Therefore the maximum nodes with height h in binary tree is $2^{h+1}-1$

The minimum number of elements is one more than the number of elements in a complete tree of height h, $min(h) = \sum_{i=0}^{h} 2^i - 1 + 1 = (2^h - 1) + 1 = 2^h$

Therefore the minimum number of nodes with height h in heap tree is 2^h and maximum is $2^{h+1}-1$.

b. $2^h \leq n < 2^{h+1} - 1$

$h \leq \log_2 n < h+1 - \log_2(1)$

$h \leq \log_2 n < h+1$

Thus it is shown that height of heap is not exceeding $\log_2 n$ so it is proven that $h = \lfloor \log_2 n \rfloor$.

5a. Algorithm MinDeletion ($H[0, \ldots n-1]$)

1. scan heap/array for the minimum element
2. Exchange the last-inserted element $H[n-1]$ with the minimum element
3. Decrease the size of the heap array by 1 and up heap the replaced node to restore max heap property

The time efficiency of the algorithm will be linear $\Theta(n)$ because the overall time of the algorithm is $\Theta(n) + \Theta(1) + O(\log n)$ since it is linear to search for the minimum element and $\Theta(n)$ is bigger than $O(\log n)$.

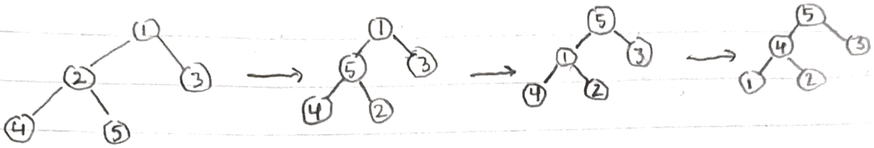
7a.

unsorted

1 2 3 4 5

sorted

∅

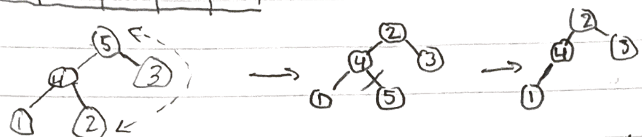


unsorted

5 4 3 1 2

sorted

∅

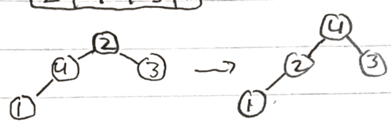


unsorted

2 4 3 1

sorted

5

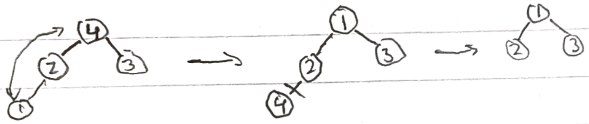


unsorted

4 2 3 1

sorted

5



unsorted

1 2 3

sorted

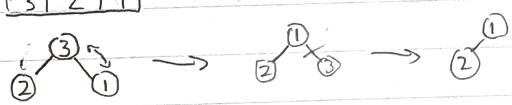
4 5

sorted

4 5

unsorted

3 2 1

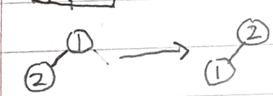


unsorted

1 2

sorted

3 4 5



unsorted

2 1

sorted

3 4 5

unsorted

1

1

sorted

2 3 4 5

unsorted

∅

sorted

1 2 3 4 5

6.5 2. Algorithm PolyEvaluation (P[0...n], x)

```
// Input: Array of coefficients of a polynomial of degree n
// Output: the value of the polynomial at the point x
p ← P[0]
power ← 1
for i ← 1 to n do
    power ← power * x
    p ← p + P[i] * power
return p
```

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

Number of additions: $A(n) = \sum_{i=1}^{n} 1 = n$

3a. Horner's rule will be twice as fast, if only multiplications need to be taken into account, because it makes n multiplications versus 2n multiplications required by the brute-force algorithm.

If one addition takes about the same amount of time as one multiplication, then Horner's rule will be about $(2n+n)/(n+n) = 1.5$ times faster.

b. No, Horner's rule is not more efficient at the expense of being less space efficient than the brute-force algorithm because Horner's rule doesn't use any extra memory.

4a. $p(x) = 3x^4 - x^3 + 2x + 5$ at $x = -2$

coefficients	3	-1	0	2	5
$x = -2$	3	$-2(3)+(-1) = -7$	$-7(-2)+0 = 14$	$14(-2)+2 = -26$	$-2(-26)+5 = 57$

The value at $x = -2$ is $\boxed{57}$

b. The intermediate numbers generated by the algorithm in the process of evaluating $p(x)$ at some point x_0 turns out to be the coefficients of the division of $p(x)$ by $x+2$ while the final result, in addition to be $p(x_0)$, is equal to the remainder of the division. Thus the quotient of the division of $p(x) = 3x^4 - x^3 + 2x + 5$ by $x+2$ is $3x^3 - 7x^2 + 14x - 26$ and the remainder is 57.

6.6 2 The algorithm that is used to create a max heap can be used to create a min heap by reversing the condition where the values are compared with each other so that the parent node contains the value which is less than or equal to the values of the child node

4a. Algorithm CycleThree (A[0..n, 0..n])

```
// Input: Adjacency matrix of a graph
// Output: True if the graph contains a triangle subgraph. Otherwise, false.
A² ← Strassen Multiplication (A, A)
for i ← 0 to n-1 do
    for j ← 0 to n-1 do
        if (A[i,j] != 0 && A²[i,j] > 0)
            return true
return false
```

Since Strassen Multiplication takes $O(n^{2.807})$ time and nested loop takes $O(n^2)$ to find the cycle, thus overall running time of the algorithm is $O(n^{2.807})$ which is less than $O(n^3)$

b. The given algorithm is not correct because a DFS tree may not contain back edge to a grand parent of a vertex which forms a cycle of length 3.

9. The edge-coloring problem can be reduced to a vertex coloring by creating a new graph which its vertices represent the edges of the graph and connect two vertices in the new graph by an edge if and only if these vertices represent two edges with a common endpoint in the original graph. The solutions for the vertex-coloring problem for the new graph solves the edge-coloring problem for the original graph.

11a. for $n=2$ (H_1, W_1) & (H_2, W_2)

$H_1 W_1 \; H_2 W_2 \longrightarrow$

$H_1 H_2 \longrightarrow W_1 W_2$

$H_1 W_1 H_2 \longleftarrow W_2$

$W_1 \longrightarrow H_1 \; H_2 W_2$

$H_1 W_1 \longleftarrow H_2 W_2$

$\longrightarrow H_1 W_1 \; H_2 W_2$

b. for $n=3$ (H_1, W_1), (H_2, W_2) & (H_3, W_3)

$H_1 W_1 \; H_2 W_2 \; H_3 W_3 \longrightarrow$

$H_1 \; H_2 \; H_3 W_3 \longrightarrow W_1 W_2$

$H_1 W_1 H_2 \; H_3 W_3 \longleftarrow$

$H_1 \; H_2 \; H_3 \longrightarrow W_1 W_2 W_3$

$H_1 W_1 \; H_2 \; H_3 \longleftarrow W_2 W_3$

$H_1 W_1 \longrightarrow H_2 W_2 \; H_3 W_3$

$H_1 W_1 \; H_2 W_2 \longleftarrow H_3 W_3$

$W_1 \; W_2 \longrightarrow H_1 \; H_2 \; H_3 W_3$

$W_1 \; W_2 \; W_3 \longleftarrow H_1 \; H_2 \; H_3$

$W_3 \longrightarrow H_1 W_1 \; H_2 W_2 \; H_3$

$H_3 W_3 \longleftarrow H_1 W_1 \; H_2 W_2$

$\longrightarrow H_1 W_1 \; H_2 W_2 \; H_3 W_3$

c. The problem does not have a solution for every $n \geq 4$, because there are more than 3 husbands and 3 wives and at some point and onward there will always exist a situation where one wife remains without her husband on the same side. Thus breaking constraint.