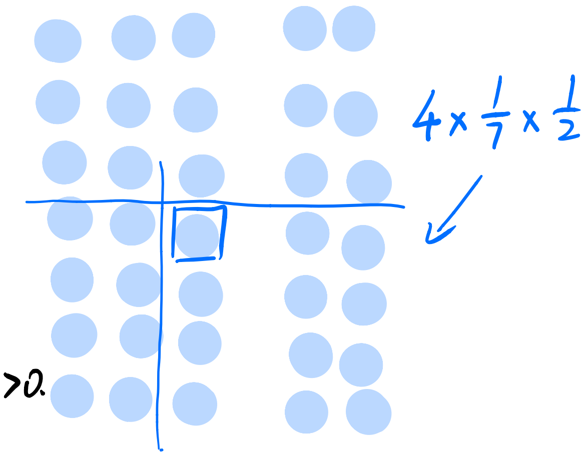
1. For any comparison-based sorting algorithm, the minimum times of comparison is $n-1$, assuming the list size is n. Thus, the smallest possible depth of a leaf in a decision tree is $n-1$.

2. (a). YES.

If we divide input elements into group of 7, then we have:

$T(n) = T(n/7) + T(5n/7) + \theta(n)$



Since we are trying to prove $T(n) \in \theta(n)$,

we assume that $T(n) \leq cn$, where $c > 0$.

$T(n) \leq c \cdot n/7 + c \cdot 5n/7 + \theta(n)$

$\theta(n)$ here is linear, so can be writen as an. $a>0$.

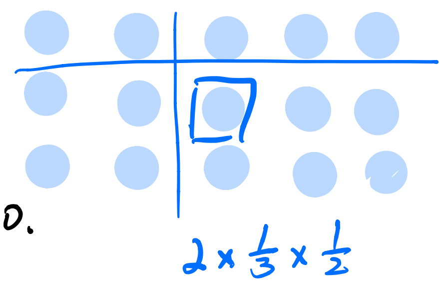
$T(n) \leq c\, 6n/7 + an$

At this point, we only need to search $\frac{6}{7}$ total elements.

$c \cdot 6n/7 + an \leq kn \ (k>0) \Rightarrow c \leq \frac{7}{6}(k-a)$

Since k. a are both constant and are larger than zero, c clearly exist.

(b) Same as above. if we devide group by 3,



we have $T(n) = T(\frac{n}{3}) + T(\frac{2}{3}n) + \theta(n)$

$T(n) \leq c \cdot \frac{n}{3} + c \cdot \frac{2}{3}n + an \leq kn$, where $a>0$. $k>0$.

$cn + an \leq kn$

At this point. it will be meaningless to continue because we still have to search all elements, partition in group of 3 did not improve this problem. Hence, group of 3 is not usable.

3. Assum the median algorithm is function findMedian(list a).

```
Select(A, first, last, i):
    subList[ ] = divideInGroupOfFive(A)
    for n in range(0, subList.size):
        median[n] = findMedian(subList[n])
    mid = findMedian(median)
    p = partition(A, first, last, mid)
    if p.indice == i:
        return p
    elif p.indice <= i:
        return Select(A, A[p.indice + 1], last, i)
    else:
        return Select(A, first, A[p.indice - 1], i)
```

4.

```
Select(X, x_first, x_last, Y, y_first, y_last):
    x_median = x_first + floor((x_last - x_first)/2)
    y_median = y_first + floor((y_last - y_first)/2)
    if x_median == y_median:
        return x_median
    elif x_median > y_median:
        return Select(X, x_median + 1, x_last, Y, y_first, y_median - 1)
    else:
        return Select(X, x_first, x_median - 1, Y, y_median + 1, y_last)
```

$T(n) = T(\frac{n}{2}) + \theta(1)$

Using master theorem, $a=1$. $b=2$

$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 = f(n)$

CASE TWO: $T(n) = f(n) \cdot \lg n = \lg n$

$T(n) \in \theta(\lg n)$

5.

```
Explore(graph G):
    for n in G.vertice:
        if n.marked == False:
            n.marked = True
            n.previsit()
            Stack.push(n)
        for i in range(0, Stack.size()):
            m = Stack.pop()
            m.postvisit()
```

6. Since T is a binary tree, we can label previsit and postvisit of every nodes using DFS algorithm. Then, we only need to examine if $pre(u) < pre(v) < post(v) < post(u)$

7.

```
ExploreAllVertice(graph G, vertice last):
    u = random_choose_vertice(G) except last
    num = 0
    for v in G.vertice except u:
        if v.marked == False:
            v.marked = True
            if connect(u, v) == True:
                num++
    if num == G.vertice_number - 1:
        return u
    else:
        return ExploreAllVertice(G, u)
```

8.

```
ExploreLinearEdge(graph G):
    linearization(G)
    count = 0
    for i in range(0, G.vertice_number):
        if is_directed_edge(v(i), v(i + 1)) = True:
            continue()
            count ++
        else:
            break()
    if count == G.vertice_number - 1:
        return 'There is a directed edge touches each vertices once'
```