P1: $T(n) = 2T(\frac{2}{3}n) + n^2$

n^2

|

$\frac{n^2}{4}$

|

⋮

|

$T(1)$ $\frac{n^2}{4^i}$

tree height is $\lg n$, leave num is 1.

$\therefore T(n) = n^2 + \frac{1}{4}n^2 + (\frac{1}{4})^2 n^2 + \cdots + (\frac{1}{4})^{\lg n - 1} n^2 + T(1)$

$$= \sum_{i=0}^{\lg n - 1} (\frac{1}{4})^i n^2 + T(1)$$

$$< n^2 \sum_{i=0}^{\infty} (\frac{1}{4})^i + T(1)$$

$$= n^2 \frac{1}{1 - 1/4} + T(1)$$

$$= \Theta(n^2).$$

Master theorem: $T(n) = aT(n/b) + f(n)$

we can know $a=2$, $b=\frac{3}{2}$ from the question;

$f(n) = n^2$,

$\therefore n^{\log_b a} = n^{\log_{\frac{3}{2}} 2} = O(n^{1.7})$

$\because f(n) = \Omega(n^{\log_{\frac{3}{2}} 2 + \varepsilon})$, $\varepsilon \approx 0.3$

we can have for n is big enough number, c is constant and $c < 1$,

$$af(n/b) \leq 2\left(\frac{n^2}{\frac{9}{4}}\right)$$

$$\leq d \cdot n^2, \quad d = 8/9,$$

$\therefore T(n) = \Theta(n^2).$

P2:

$T(n) = 3T(n/2) + n/\log n$

As we know Integral Theorem

$\sum_{k=1}^{n} \frac{1}{k} \log n + O(1)$

so, we can first calculate leave complexicity, the number of leaves is $3^{\log n} = n$,

$\because$ every leave complexicity is $\Theta(1)$

$\therefore$ total is $\Theta(n)$.

next,

we calculate the intermediate and root nodes, as above, we have $\log n$ levels.

$$g(n) = \sum_{i=0}^{\log n-1} 3^i \cdot \left(\frac{\frac{n}{3^i}}{\log \frac{n}{3^i}} \right) = n \cdot \sum_{i=0}^{\log n-1} \frac{1}{\log n - \log 3^i}$$

$$= n \cdot \sum_{i=0}^{\log n-1} \frac{1}{\log n - i}$$

$$\sum_{i=0}^{\log n-1} \frac{1}{\log n - i} = \frac{1}{\log n} + \frac{1}{\log n-1} + \cdots + \frac{1}{2} + 1$$

$$= \log(\log n) + O(1)$$

$$= \Theta(\log\log n)$$

$$\therefore T(n) = g(n) + \theta(n)$$
$$= n \cdot \theta(\log\log n) + \theta(n)$$
$$= \theta(n \log\log n).$$

let $T(n) \le d \cdot n \log\log n$

$$\therefore T(n) = 3T\left(\frac{n}{2}\right) + \frac{n}{\log n}$$
$$\le 3 \cdot d \cdot \left(\frac{n}{3} \log\log\frac{n}{3}\right) + \frac{n}{\log n} = c \cdot n \log\log\left(\frac{n}{3}\right) + \frac{n}{\log n}$$

For upper bounds, we set $3^k = n$.

$$c \cdot n \log\log\left(\frac{n}{3}\right) + \frac{n}{\log n} \le c \cdot n \log\log n$$

$$\frac{n}{\log n} \le c \cdot n \cdot \left(\log\log n - \log\log\frac{n}{3}\right)$$

$$\frac{1}{k} \le c \cdot \left(\log\frac{k}{k-1}\right)$$

$$1 \le c \cdot \log\left(1 + \frac{k}{k-1}\right)^k$$

$$c \cdot \log\left(1 + \frac{1}{k-1}\right)^k \ge c \cdot \log e \ge 1$$

$$\therefore T(n) \le d \cdot n \log\log n.$$

For lower bounds, we can use same idea to prove.

P3. a: $T(n) = \sqrt{n}\,T(\sqrt{n}) + n.$

$$= n^{\frac{1}{2}}\left[n^{\frac{1}{2}} + n^{\frac{1}{2^2}} T(n^{\frac{1}{2^2}})\right] + n$$

$$= n^{\frac{1}{2}+\frac{1}{2^2}}\left[n^{\frac{1}{2^2}} + n^{\frac{1}{2^3}} T(n^{\frac{1}{2^3}})\right] + 2n$$

$$\cdots$$

$$= n^{\frac{1}{2}+\frac{1}{2^2}+\cdots+\frac{1}{2^i}}\left[n^{\frac{1}{2^i}} + n^{\frac{1}{2^{i+1}}} T(n^{\frac{1}{2^{i+1}}})\right] + i\cdot n$$

let $n^{\frac{1}{2^{i+1}}} = 2$, $i = \log\log n$

then,

$$T(n) = n + n^{\frac{1}{2}+\frac{1}{2^2}+\cdots+\frac{1}{2^i}} T(2) + n\cdot(\log\log n - 1)$$

$$= \Theta(n\log\log n),$$ we have constant c_1, c_2 to represent upper and lower bounds.

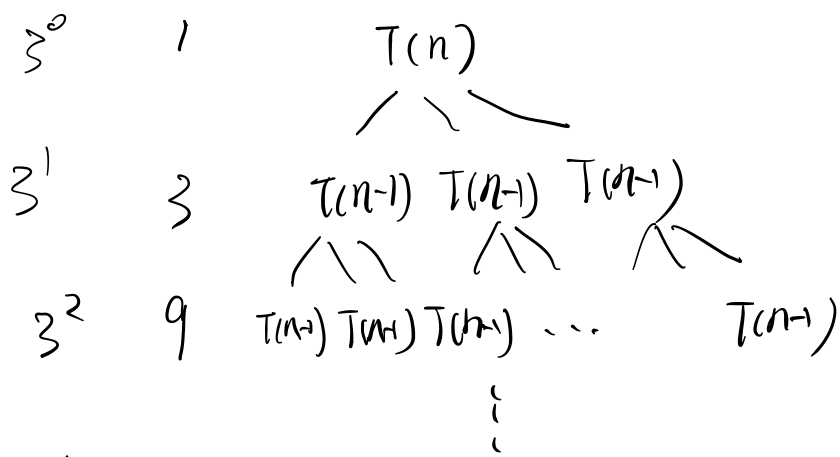
b: $T(n) = 3T(n-1).$

$$T(n) + 3 = 3T(n-1) + 3 = 3(T(n-1) + 1)$$

$$T(n) + 3 = 3^{n-1}(T(1) + 1)$$

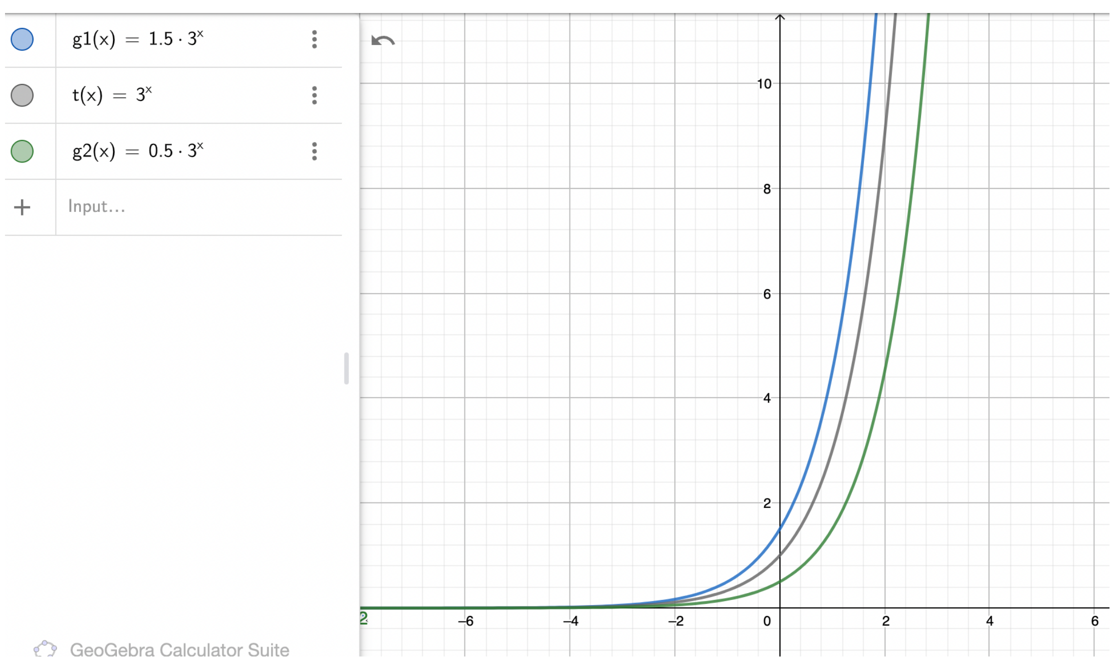
$$\therefore T(n) = 2\cdot 3^{n-1} - 1$$

$$\therefore T(n) = \Theta(3^n)$$



$3^0+3^1+3^2+\dots+3^k=3^{k+1}-1$

As we get $T(n)=\Theta(3^n)$,
then there must be constant c_1 c_2 representing upper bounds and lower bounds.



As this image showing that, when $n>n_0$,
we have constant number c_1 and c_2,

lower bounds: $c_1 \cdot g(n) = c_1 \cdot (3^n)$.

upper bounds: $c_2 \cdot g(n) = c_2 \cdot (3^n)$.

P4:

Run Code Problem set3 Save Java

```
public class Dijkstra {
    public static void main(String[] args) {
        String[] vertex={"V1","V2","V3","V4","V5","V6"};
        int[][] matrix=new int[vertex.length][vertex.length];
        int inf=Short.MAX_VALUE;
        matrix[0]=new int[]{0,  16, 22, 30, 41, 59};
        matrix[1]=new int[]{inf,0,  16, 22, 30, 41};
        matrix[2]=new int[]{inf ,inf,0, 17, 23,31};
        matrix[3]=new int[]{inf ,inf,inf,0,17,23};
        matrix[4]=new int[]{inf ,inf,inf,inf,0,18};
        matrix[5]=new int[]{inf,inf,inf,inf,inf,0};
        Graph graph=new Graph(vertex,matrix);
        graph.dijkstra(0);
    }
}

class Graph{
    private String[] vertex;
    private int[][]  matrix;
    private VisitedVertex vv;
    public Graph(String[] vertex,int[][] matrix){
        this.vertex=vertex;
        this.matrix=matrix;
    }
    public void show(){
        for(int[] link:matrix){
            System.out.println(Arrays.toString(link));
        }
    }
    public void dijkstra(int index){
       vv=new VisitedVertex(index,vertex);
       update(index);
       for(int i=1;i<vertex.length;i++){
           int value=vv.updateAll();
```

Output: Finished

```
Finished in 74 ms
[0, 16, 22, 30, 41, 53]
```

```java
            update(value);
        }
        vv.show();
    }
    public void update(int index){
        int len=0;
        for(int i=0;i<vertex.length;i++){
            len=vv.getDis(index)+ matrix[index][i];
            if(!vv.isInVisited_vertex(i)&& len<vv.getDis(i)){
                vv.dis[i]=len;
                vv.pre_vertex.put(vertex[i],""+(index+1));
            }else if(!vv.isInVisited_vertex(i)&&len==vv.getDis(i)){
                String value=vv.pre_vertex.get(vertex[i]);
                vv.pre_vertex.put(vertex[i],""+value+","+(index+1));
            }
        }
    }

}
class VisitedVertex{
    public int[] dis;
    public int[] visited_vertex; // 1 visited, 0 not visited
    public Map<String,String> pre_vertex;

    public VisitedVertex(int index,String[] vertex){
        this.visited_vertex=new int[vertex.length];
        visited_vertex[index]=1;
        this.dis=new int[vertex.length];
        Arrays.fill(dis,Short.MAX_VALUE);
        dis[index]=0;
        this.pre_vertex=new HashMap<>();
        for (String s : vertex) {
            pre_vertex.put(s, "" + 0);
```

```java
        }
    }

    public boolean isInVisited_vertex(int index){
        return visited_vertex[index]==1;
    }
    public int getDis(int index){
        return dis[index];
    }
    public int updateAll(){
        int index = 0;
        int min=Short.MAX_VALUE;
        for(int i=0;i<visited_vertex.length;i++){
            if(visited_vertex[i]==0&&dis[i]<min){
                min=dis[i];
                index=i;
            }
        }
        visited_vertex[index]=1;
        return index;
    }
    public void show(){
        System.out.println(Arrays.toString(dis));
    }
}
```