Cost analysis of the algorithm (c_3 → problem_2.py)

```
class Kruskall:
    def __init__(self,size):                                    #c1=1
        self.size = size                                        #c2=1
        self.edges = []                                         #c3=1
        self.vertex = ['']*size                                 #c4=n

    def createEdgeList(self,weight,u,v):
        if 0 <= u < self.size and 0 <= v < self.size:          #c5=1
            self.edges.append((u,v,weight))                    #c6=1

    def createVertexList(self,vertex,d):
        if 0 <= vertex < self.size:                            #c7=1
            self.vertex[vertex] = d                            #c8=1

    def find(self,parent,i):
        if parent[i]==i:                                      #c9=1
            return i                                          #c10=1
        return self.find(parent,parent[i])                    #c11=log n

    def union(self,parent,rank,x,y):
        xRoot = self.find(parent,x)                           #c12=log n
        yRoot = self.find(parent,y)                           #c13=log n
        if rank[xRoot] < rank[yRoot]:                         #c14=1
            parent[xRoot]=yRoot                               #c15=1
        elif rank[xRoot]>rank[yRoot]:                         #c16=1
            parent[yRoot]= xRoot                              #c17=1
        else:
            parent[yRoot]=xRoot                               #c18=1
            rank[xRoot]+=1                                    #c19=1

    def kruskalProblem(self):

        self.edges=sorted(self.edges, key=lambda item:
item[2])#c20=E.log E
        parent = [i for i in range(self.size)]                 #c21=n
        rank = [0]*self.size                                   #c22=n

        result = []                                            #c23=1

        for u,v,weight in self.edges:                         #c24=E
            x = self.find(parent,u)                           #c25=E.log n
```

```
            y = self.find(parent,v)                              #c26=E.log n
            if x != y:                                                   #c27=E
                result.append((u,v,weight))                              #c28=E
                self.union(parent,rank,x,y)     #c29=E.log n

        mstResult=[]                                                 #c30=1

        for(u,v,weight) in result:                                   #c31=V-1
            mstResult.append((self.vertex[u],self.vertex[v],weight))
#c32=V-1

        return mstResult                                              #c33=1
```
*n representa o número de vértices no grafo
*E representa as arestas

- Basic operation:
  $C_{11}, C_{12}, C_{13} = log\, n$
  $C_{25}, C_{26}, C_{29} = E.\, log\, n$

- Time complexity calculation:
  $T(n) = (c_{11} + c_{12} + c_{13}) . (log\, n) + (c_{25} + c_{26} + c_{29}). (E.\, log\, n)$
  $T(n) = 3log\, n + 3(E.\, log\, n)$
  $T(n) = E.\, log\, n$
  $T(n) \in O(E.\, log\, n)$

- Solving the recurrence:
  $T(n) = E.\, log\, E + E.\, log\, n, \ n >= 1, \ T(0) = O(E.\, log\, E)$
  $T(1) = E.\, log\, E + E.\, log$