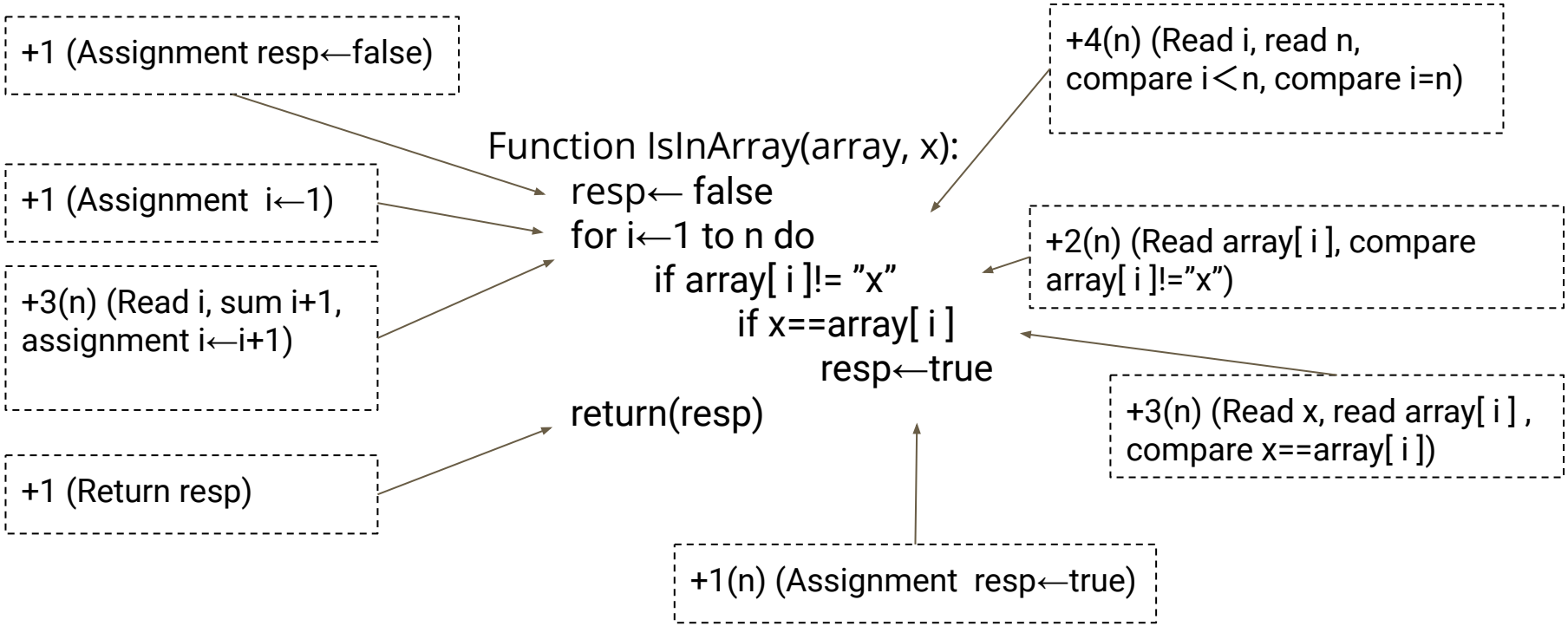
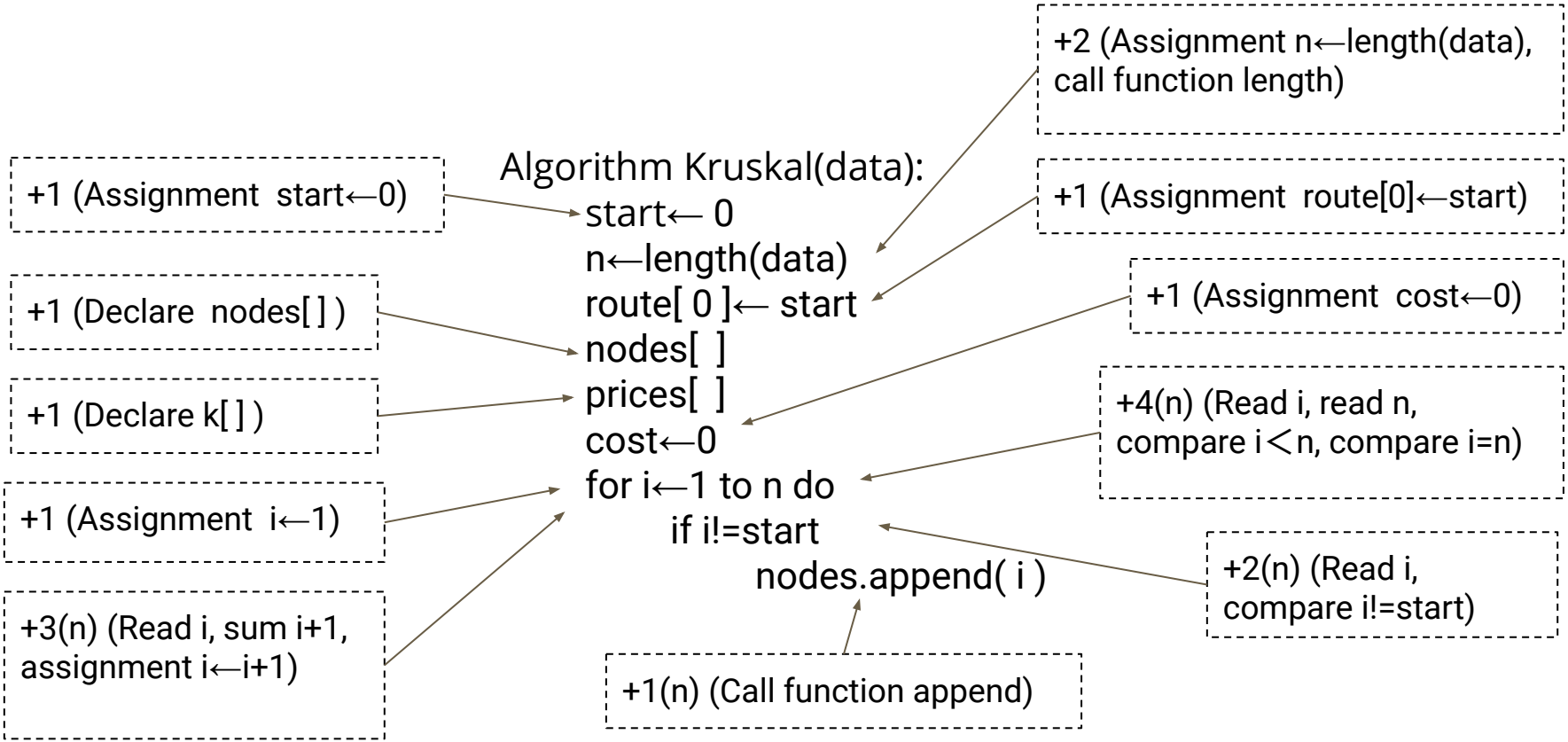
Kruskal
Minimum Spanning Tree

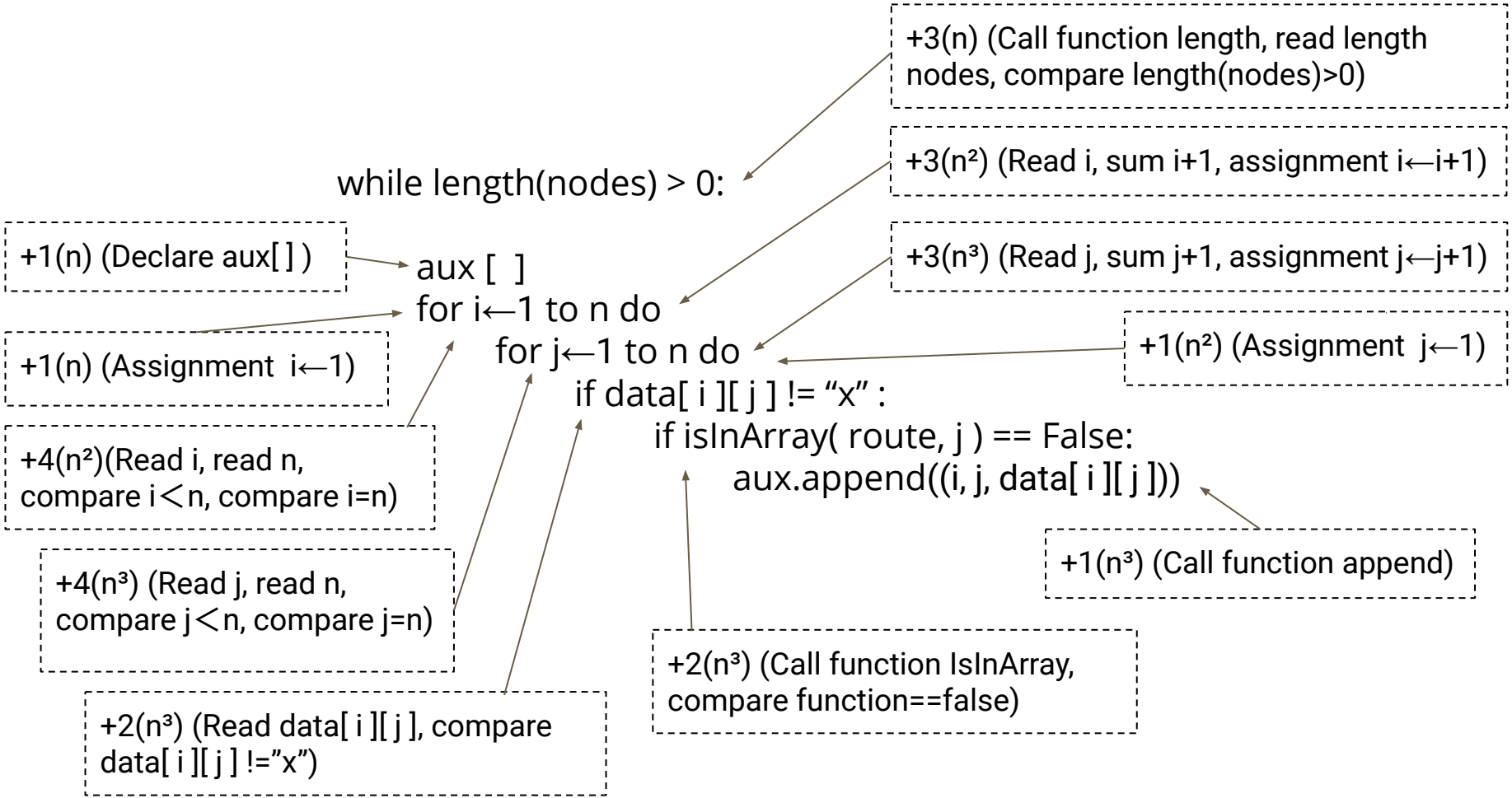
Damaso Reyes Belen
Flores Zenteno Alfonso
Serrano Ramírez Viviana

Big-Oh Notation

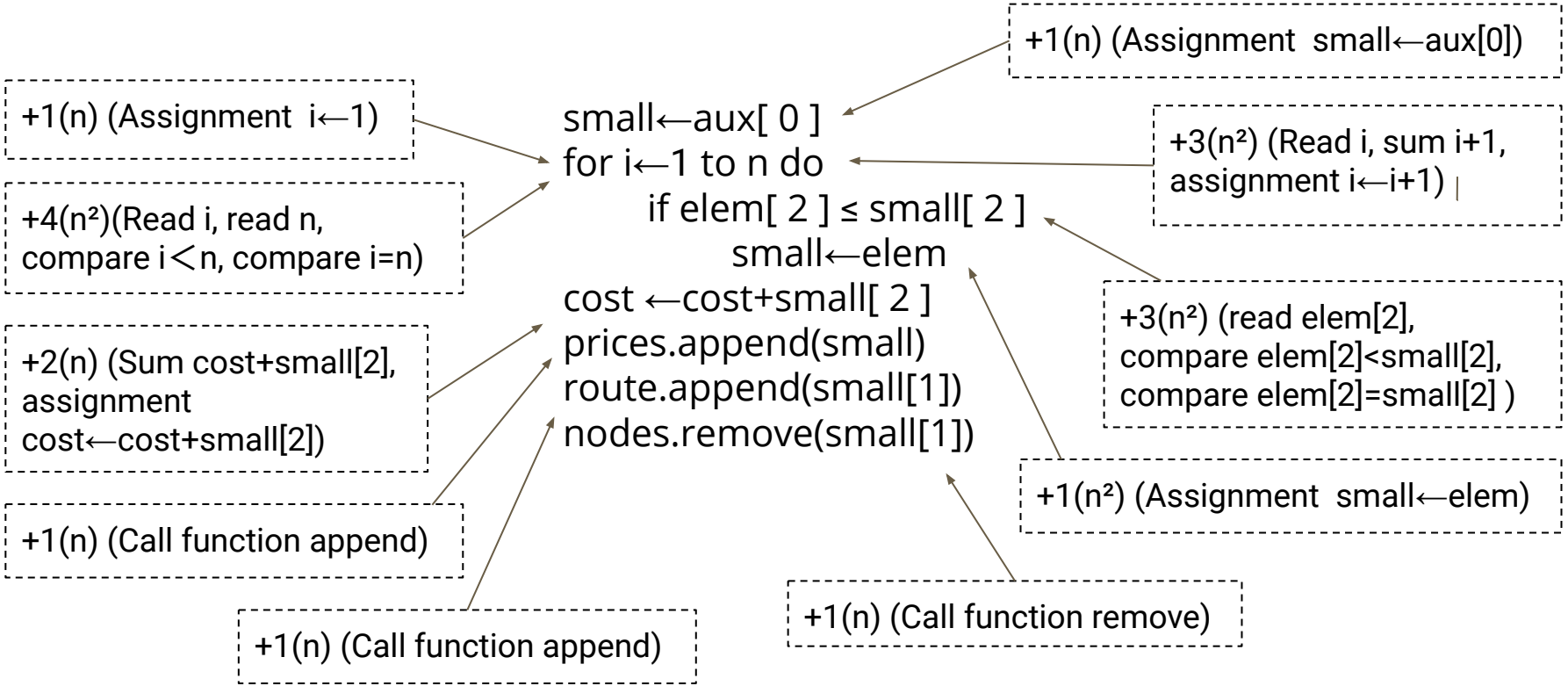
Step 1: Counting the basic operations

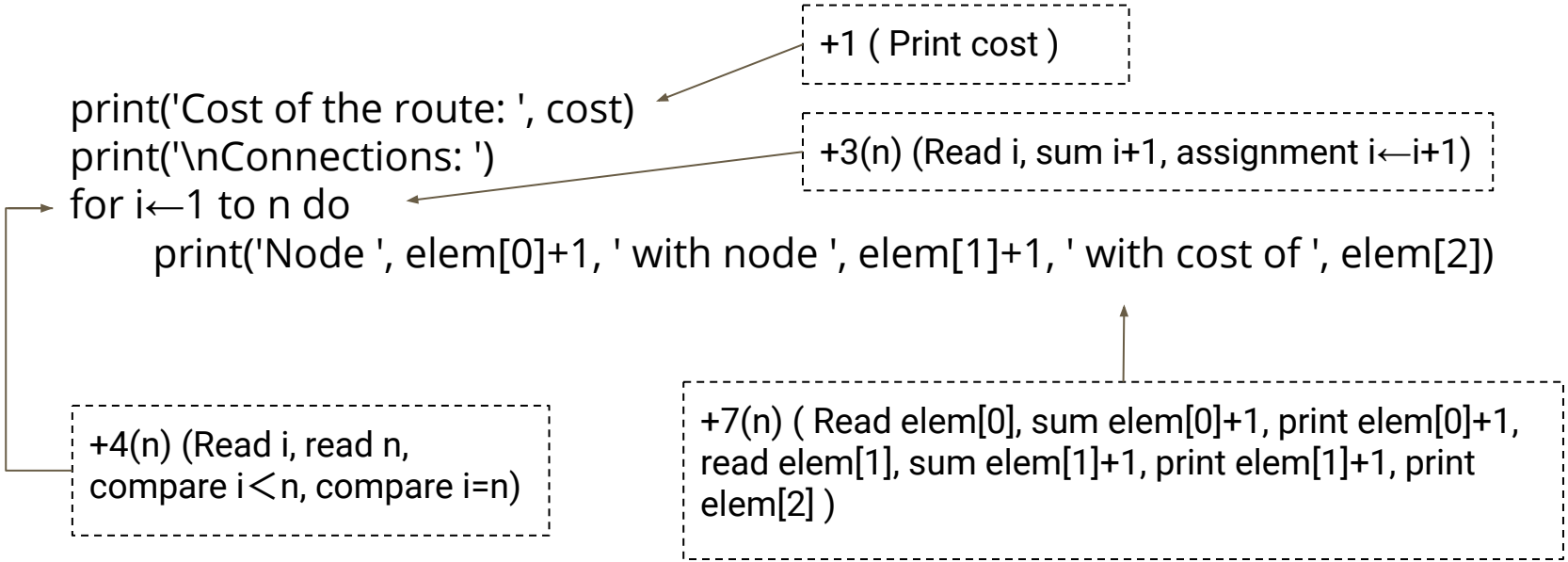



Algorithm Kruskal(data):
start← 0
n←length(data)
route[0]← start
nodes[ ]
prices[ ]
cost←0
for i←1 to n do
if i!=start
nodes.append(i)
+1 (Assignment start←0)
+2 (Assignment n←length(data), call function length)
+1 (Assignment route[0]←start)
+1 (Declare nodes[ ])
+1 (Assignment cost←0)
+1 (Declare k[ ])
+4(n) (Read i, read n, compare i＜n, compare i=n)
+1 (Assignment i←1)
+2(n) (Read i, compare i!=start)
+3(n) (Read i, sum i+1, assignment i←i+1)
+1(n) (Call function append)


+3(n) (Call function length, read length nodes, compare length(nodes)>0)
+3(n²) (Read i, sum i+1, assignment i←i+1)
+3(n³) (Read j, sum j+1, assignment j←j+1)
+1(n²) (Assignment j←1)
while length(nodes) > 0:
+1(n) (Declare aux[ ])
aux [ ]
for i←1 to n do
for j←1 to n do
if data[i][j] != “x” :
if isInArray(route, j) == False:
aux.append((i, j, data[i][j]))
+1(n) (Assignment i←1)
+4(n²)(Read i, read n, compare i<n, compare i=n)
+1(n³) (Call function append)
+4(n³) (Read j, read n, compare j<n, compare j=n)
+2(n³) (Call function IsInArray, compare function==false)
+2(n³) (Read data[i][j], compare data[i][j] !=”x”)

Same While...


+1(n) (Assignment small←aux[0])
+1(n) (Assignment i←1)
small←aux[0]
for i←1 to n do
+3(n²) (Read i, sum i+1, assignment i←i+1)
+4(n²)(Read i, read n, compare i＜n, compare i=n)
if elem[2] ≤ small[2]
small←elem
+3(n²) (read elem[2], compare elem[2]<small[2], compare elem[2]=small[2])
cost ←cost+small[2]
prices.append(small)
route.append(small[1])
nodes.remove(small[1])
+2(n) (Sum cost+small[2], assignment cost←cost+small[2])
+1(n²) (Assignment small←elem)
+1(n) (Call function append)
+1(n) (Call function remove)
+1(n) (Call function append)


+1 (Print cost)
print('Cost of the route: ', cost)
print('\nConnections: ')
+3(n) (Read i, sum i+1, assignment i←i+1)
for i←1 to n do
print('Node ', elem[0]+1, ' with node ', elem[1]+1, ' with cost of ', elem[2])
+4(n) (Read i, read n, compare i＜n, compare i=n)
+7(n) (Read elem[0], sum elem[0]+1, print elem[0]+1, read elem[1], sum elem[1]+1, print elem[1]+1, print elem[2])

Step 2: Estimate execution time

$$t(n) = 3+3+8+1+12n+10n+5n+7n+14n+8n^2+11n^2+12n^3$$

$$=12n^3+19n^2+48n+15$$

$c>0, c \in \mathbb{R}$

$\therefore$ The highest number of basic operations could be $12n^3+19n^2+48n+15$

Step 3: Applying the definition of Big-Oh

With $c = 94$ and $n_0 = 1$:

$$12n^3+19n^2+48n+15 \leq 94n^3 \quad \text{for } n \geq 1$$

$\therefore$ Kruskal(data) is $O(n^3)$