Análisis de complejidad temporal

Método 1:

	Repetición
int e = Integer.parseInt(in.readLine());	1
for (int i=0; i<e; i++) {	n+1
String[] l = in.readLine().split(" ");	n
int officesPerFloor = Integer.parseInt(l[3]);	n
int totalPeople = Integer.parseInt(l[1]);	n
int totalFloors = Integer.parseInt(l[2]);	n
backend.addEdifice(l[0], totalFloors, officesPerFloor, totalPeople, officesPerFloor * totalFloors);	n
for (int j=0; j<totalPeople; j++) {	n*n+1
String[] l2 = in.readLine().split(" ");	n*n
int destination = Integer.parseInt(l2[2]);	n*n
int targetFloor = (int) (Integer.parseInt(l[2]) - (Math.ceil((float) destination/(float)officesPerFloor)-1));	n*n
backend.getEdifice(l[0]).getFloor(Integer.parseInt(l2[1])).addPerson(l2[0], destination, targetFloor); }}	n*n

T(n)= 1 + (n+1)+n+n+n+n+n+(n*(n+1))+(n^2)+(n^2)+(n^2)+(n^2)

T(n)= 5(n^2)+7n+2
T(n)= O(n^2)

Análisis de complejidad espacial

Método 1:

Tipo	Variable	Tamaño de 1 valor atómico	Cantidad de valores atómicos
Entrada	e l officesPerFloor totalPeople totalFloors l2 destination targetFloor	32 bits 16 bits 32 bits 32 bits 32 bits 16 bits 32 bits 32 bits	1 n 1 1 1 n 1 1
Auxiliar	i j	32 bits 32 bits	1 1
Salida			

Complejidad espacial total = 1+n+1+1+1+n+1+1+1+1 =2n+8= θ(n)

Análisis de complejidad temporal

Método 2:

	Repetición
number = n;	1
occupants = new HashSet<Person>();	1
needGoUp = new LinkedList<Person>();	1
needGoDown = new LinkedList<Person>();	1
offices = new HashMap<Integer, Office>();	1
for (int i=0; i<o; i++) {	n+1
offices.put(oStart-i, new Office()); }	n
this.elevator = elevator;	1
this.edifice = edifice;	1

$$T(n)= 1 +1+1+1+1+(n+1)+n+1+1$$
$$T(n)=2n+8$$
$$T(n)=O(n)$$

Análisis de complejidad espacial

Método 2:

Tipo	Variable	Tamaño de 1 valor atómico	Cantidad de valores atómicos
Entrada	number occupants needGoUp needGoDown offices elevator edifice	32 bits ? bits ? bits ? bits ? bits ? bits ? bits	1 n n n n 1 1
Auxiliar	i	32 bits	1
Salida			

Complejidad espacial total = $1+n+n+n+n+1+1+1 = 4n+4= \theta(n)$