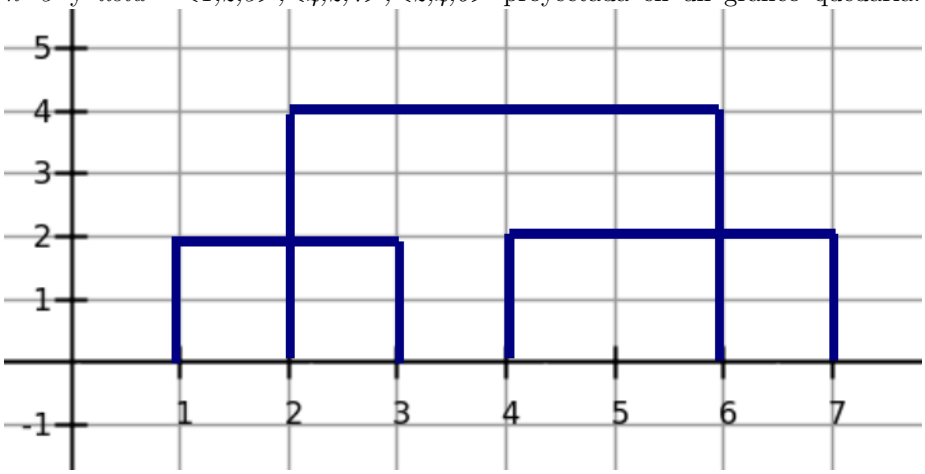
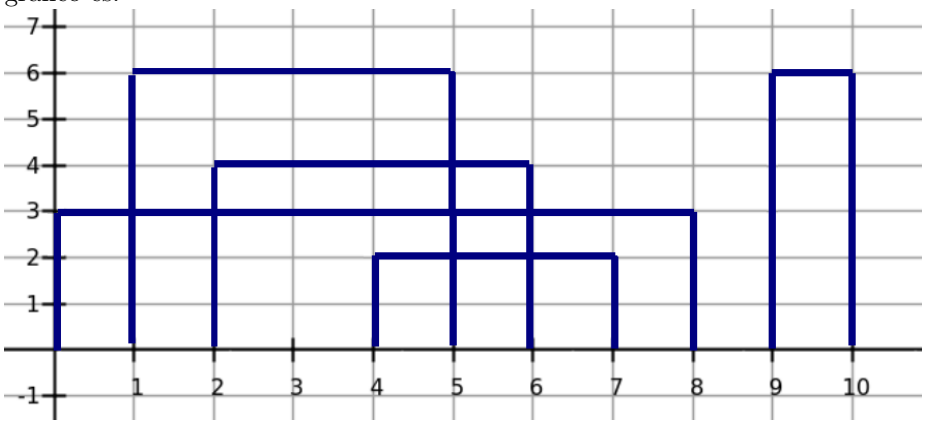
Problema a resolver:

El problema esta dado por la siguiente situación: tenemos en un "lista" con una cantidad $3*n$ de números(n un número fijo).
Para i desde *0* a *n-1*, vamos a decir la posición i en la lista va a ser *Izq* del edificio *i-ésimo*, *i+1* va a ser *Alt* del edificio *i-ésimo* e *i+2* va a ser *Der* del edificio i-ésimo.
A grandes rasgos vamos a tener una lista de n edificios (interpretamos a un edificio como una tupla *<Izq,Alt,Der>*) con una base en común implícita que es 0.
Por ejemplo para un entrada de la forma:
n=3 y *lista= <1,2,3>,<4,2,7>,<2,4,6>* proyectada en un gráfico quedaría:

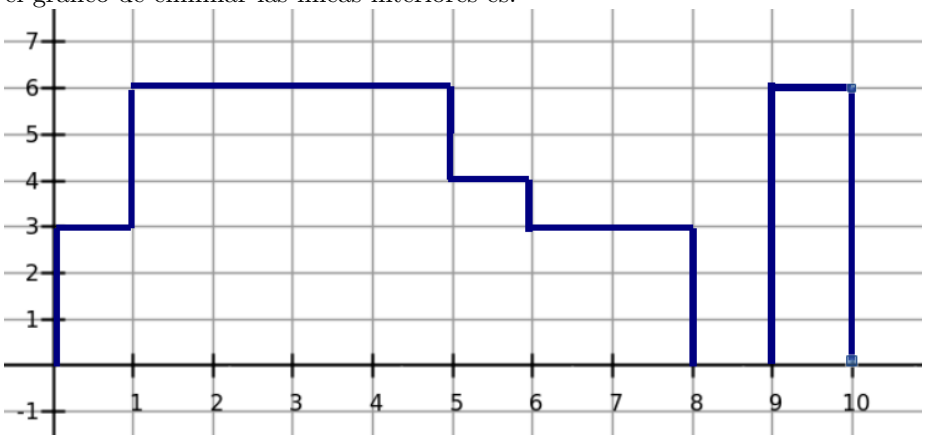


Lo que queremos hacer es "eliminar todas las lineas interiores del gráfico", quedarnos con su contorno (se obtiene el mismo resultado "siguiendo con el dedo el gráfico") para luego poder dar la solución final que explicaré más adelante.

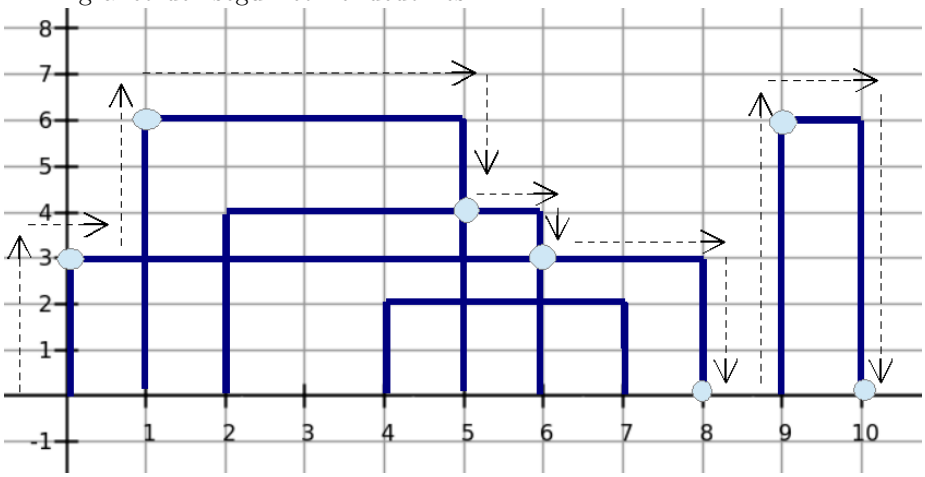
Para una *lista*= *<0,3,8>,<1,6,5>,<2,4,6>,<4,2,7>,<9,6,10>* con *n=5* su gráfico es:



el gráfico de eliminar las lineas interiores es:



El gráfico de "seguir con el dedo" es:



La salida para este ejemplo es *salida=0,3,1,6,5,4,6,3,8,0,9,6,0*

Lo que hago cuando "sigo con el dedo" es:
Empezar con el primer edificio y seguimos el trazo, si me interseco con otro edificio seguir el trazo del edificio con el que me intersequé desde ese punto.
Si no me interseco con nadie pero hay más edificios adelante "siguir con el dedo" los otros. Si no hay más edificios terminé.
Luego de ese contorno voy a obtener la solución final que son los puntos donde hay cambios($\uparrow \longrightarrow$ y $\downarrow \longrightarrow$).

Resolución:

Un panorama de la resolución es:
ordenar los edificios de menos a mayor por su Izq (en caso que tengan la misma izquierda es menor el que tiene mayor altura) y retornar el primer punto del primer edifico.
Vamos recorriendo los edificios(mirando el edificio por el que voy,anterior, y uno mas adelante,siguiente), si anterior interseca a siguiente, encolo anterior y retorno el cambio de altura si siguiente es mayor en altura que anterior.
Si es igual en altura o mayor, ahora anterior es siguiente. Si es menor en altura no hago nada. no vale la pena ponerlo
Si anterior no interseca a siguiente(quiere decir que están "separados",pero puede que antes de anterior haya un edificio que termine después que anterior y sea menor en altura) voy a buscar este edificio en la cola (ordenada por altura) desencolando y mirando el tope:
si lo encuentro, ese edificio ahora es anterior, retorno la intersección.
si la cola es vacía(quiere decir que no había ningún edificio que terminara después que anterior) retorno anterior.Der,0,siguiente.Izq,siguiente.Alt y ahora anterior es siguiente.
Terminé de recorrer los edificios,pero puede que hayan quedado cosas dentro del heap, y son puntos que debería tener la solución Mientras el heap tenga edificios, voy a comparar el tope con anterior:
si tope termina antes que el anterior desencolo, en caso contrario imprimo la intersección. Ahora anterior es el toper de la cola y desencolo.
Hay cosas que en esta descripción no tuve en cuenta, porque son muy especificas, para explicarlo profundamente lo hago con este pseudocódigo:

Sea lista: lista(<Izq,Alt,Der>) y n la cantidad de tuplas en la lista.

```
1:  procedure ResolverEdificios(lista,n)
2:      ordenarlosedificiosporIzq
3:      comparo ← lista[0]
4:      cola ← vacio
5:      imprimo el primer punto(comparo.Izq y comparo.Alt)
6:      for (i ← 1, n − 1) do //voy recorriendo los edificios, si hay 0 edificios que hago?
7:          siguiente ← lista[i]
8:          if (se intersecan comparo y siguiente*0) then
9:              if (siguiente > comparo en altura *1) then
10:                 imprimir cambio de altura
11:                 if (la cola está vacia) then
12:                     cola.encolar(comparo)
13:                 else
14:                     if (comparo no está en el tope del cola) then
15:                         cola.encolar(comparo)
```

```
16:                     end if
17:                 end if
18:                 comparo ← siguiente
19:             end if
20:             if (siguiente == comparo en altura *2) then
21:                 if (la cola está vacia) then
22:                     cola.encolar(comparo)
23:                 else
24:                     if (comparo no está en el tope del cola) then
25:                         cola.encolar(comparo)
26:                     end if
27:                 end if
28:             end if
29:             if (siguiente < comparo en altura *3) then
30:                 cola.encolar(comparo)
31:             end if
32:         end if(no se intersecan comparo y siguiente*4)
33:         if (la cola no está vacia) then
34:             while (cola no vacia) do
35:                 if (primero.cola termina antes que comparo *5) then
36:                     desencolar.cola
37:                 else(primero.cola termina despues que comparo *6)
38:                     imprimir interseccion entre comparo y primero.cola
39:                     comparo ← primero.cola
40:                     desencolar.cola
41:                 end if
42:             end while
43:         else //no pasé a nadie que cortaría a comparo, como no se intersecan
    imprimo ambos puntos
44:             imprimir comparo.Der y 0
45:             imprimir siguiente.Izq y siguiente.Alt
46:             comparo ← siguiente
47:         end if
48:     end for(no hay siguiente,puede que hayan quedado cosas en el cola *7)
    //uso a comparo que es el último edificio con el que haya en el tope de la
    cola
49:     for (la cola no sea vacia, desencolar) do
50:         if (comparo termina antes que primero.cola *8) then
51:             imprimir intersección comparo y primero.cola
52:             comparo ← cola.primero
53:         end if
54:     end for(al último punto no lo imprimo nunca*9)
    //lo imprimo acá
55:     imprimir comparo.Der y 0
56: end procedure
```

Complejidad:

Vamos a ver que la cantidad de veces que encolo es una funcion de n, y así poder ver que la cantidad de veces que desencolo es tambien una funcion de n porque no puedo desencolar más cosas de las que encolo. Vemos en el algoritmo que en *1 y *2,encola si está vacia y si el elemento está en el tope, no lo encolo.(falta demostrar que si quiero encolar un elemento 2 veces el que quiero encolar de nuevo está en el tope, la idea la había sacado angel)Luego en *3 encolo. Despues a lo largo de todo el algoritmo no hago *nunca* un encolar Como todos estos casos son disjuntos (o son <,> o = las alturas de comparo y siguiente) entonces hago 1 encolar en cada edificio en peor caso, con lo cual hago n encolar.

Veamos ahora la complejidad del while dentro del for(de recorrer los edificios), en peor caso por cada edificio desencolo todos los edificios eso da una complejidad n*(n*log(n)), pero si analizamos más finamente, nunca podría para cada paso desencolar todos los edificios.
Si voy por el edificio i (i entre *0* y *n-1*), tengo en el heap en peor caso i edificios(por lo explicado arriba de la cantidad de encolar), entro en el while(suponiendo que los edificios i e *i+1* no se tocan) y desencolo i edificios(en peor caso desencolo todos los que encolé), sigo avanzando y llego a un edificio j (j entre *0* y *n-1* y es mayor que i) ahora en el heap en peor caso tengo *j-i* edificios(pues los edificios anteriores a i ya no están en la cola y encolé todos desde i hasta j), entro en el while(suponiendo que los edificios j y j+1 no se tocan) y desencolo en peor caso *j-i* edificios. llego al último edificio *n-1* y en peor caso tengo *n-1-j* edificios en el heap(generalizando lo anterior), entro al while y desencolo *n-1-j* veces. Si sumo la cantidad de veces que hice desencolar en el recorrido lineal es n (suma de los intervalos). Entonces hago n veces desencolar

Relacionando la parte de encolar y desencolar en el primer for por cada edificio hago un encolar y una cantidad x de desencolar + c(*constante*) operaciones que tienen costo 1 (asignaciones y guardas) Por lo visto anteriormente la suma de esos x es n, entonces el costo es n*(costo de encolar)+ n*(costo de desencolar) +c(constante)
Solo nos falta analizar el segundo for(cuando salgo del primero), en peor caso tengo todos los edificiosque es n, y hasta desencolo hasta que se vacia,entonces hace n*((costo de desencolar) + c1(constante de guardas y asignaciones)) En la implementación usamos una priority-queue como cola, el costo de encolar(push) es log(n),desencolar(pop) es log(n) y tope(top) es 1.
Al princio del algoritmo ordeno los edificios por Izq, el costo de ordenarlos es n*log(n) (porque uso sort de la stl según este link).

Finalmente la complejidad es
O(n*log(n) de ordenarlos
+ c[*constante*]+n[*encolar*]+n*(log(n))[*desencolar*] del primer for
+ c1[*constante*] + n*log(n)[*desencolar*] del segundo for) $\in$ O(n(log(n)))
**

Correctitud:

El algoritmo pone el primer punto(el edificio más chico por derecha y más alto) y el último(el edificio que termina último) SIEMPRE.(dónde lo hace!!!)
Quiero ver que va a poner los puntos intermedios correctamente.
Tengo los edificios ordenados por izquierda(en caso de igual izquierda es menor el de mayor altura) y accedo a ellos secuencialmente mirando el edificio por el que voy(anterior) y el siguiente.

Si anterior y siguiente se intersecan:
si anterior es mayor a siguiente (grafico A)
tengo que poner el punto de intersección en la solución
Supongamos que ese punto (intersección) no es solución:
caso existe un edificio que empieza entre ia y is, termina después que is y tiene altura entre as y aa (ejemplo gráfico B):
si existiera este edificio tendría que ser haberlo puesto como siguiente , ABS!!!
Entonces el punto es solución.
caso exite un edificio que empieza antes que ia, tiene altura mayor aa y termina después que is (ejemplo grafico C):
Si existiera tendría que haberlo puesto como anterior, ABS!!!
Entonces el punto es soución.

Si no se intersecan anterior y siguiente():
voy a tener en el heap los edificios que terminan después que empieza el anterior porque los fui enconlando (grafico D) si el heap está vacío quiere decir que no hay nadie que corte a anterior,entonces el edificio anterior sigue hasta su base y tengo que seguir con los demás edificios(en caso de tener más) desde siguiente, entonces voy a poner en la solución el final de comparo y el principio de siguiente.
Si el heap no está vacío quiere decir que puede que tenga un edificio que corte a anterior voy a sacar del heap hasta que encuentro uno que interseque a anterior, por como es el heap (está ordenado por mayor altura) este que me interseca es el que tiene mayor altura, entonces este punto de intersección es solución.
Supongamos que este punto de intersección no es solución.
caso (grafico E) exite un edificio que empieza antes que d, termina despues que ad y tiene altura entre aa y ha, entonces este si exitiera, lo tendría que haber encolado cuando reccorría los edificios y tendría que ser el edificio que me daría el heap ABS!!!
Entonces es solución.

NO ME GUSTA NADA ESTA DEMO! igual no es la final