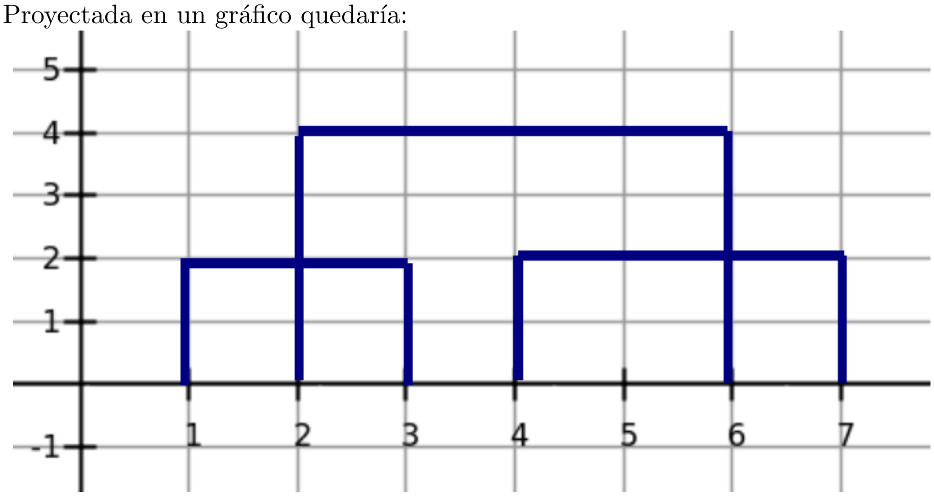
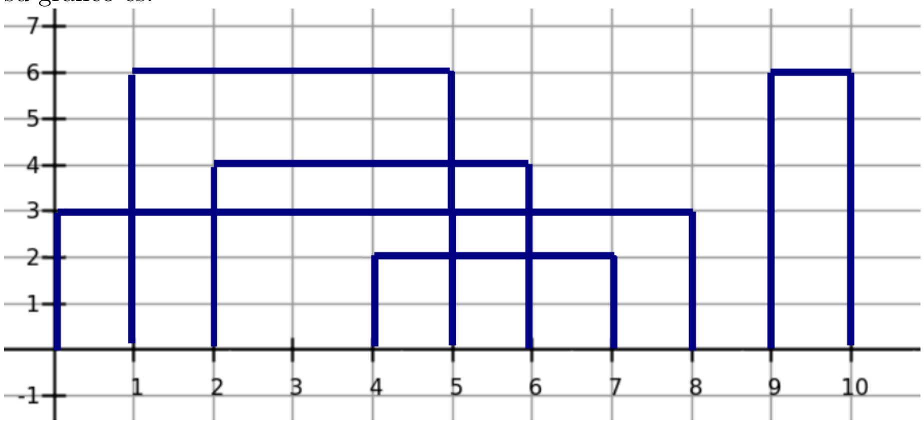
Problema a resolver:

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 que 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.
Vamos a utilizar esta notación para referirnos a los edificios.
Por ejemplo para un entrada de la forma:
lista=1,2,3,4,2,7,2,4,6 y *n=3* tendríamos:
lista= <*1,2,3*>,<*4,2,7*>,<*2,4,6*>
Proyectada en un gráfico quedaría:

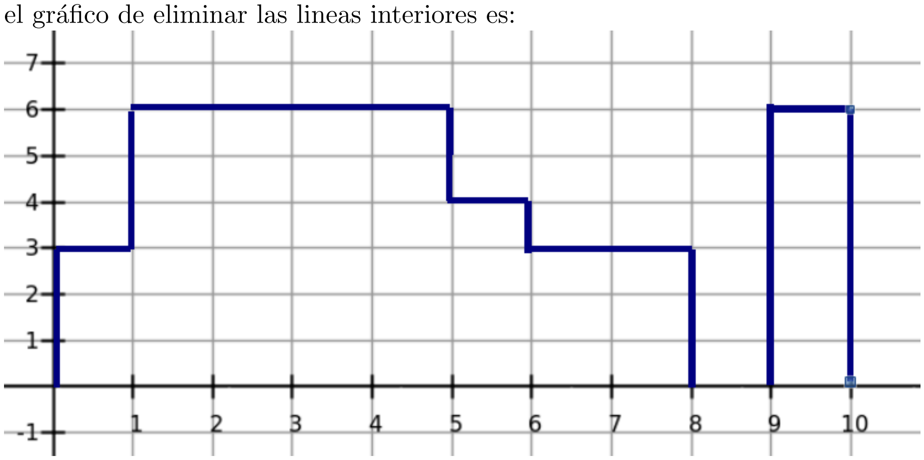


Lo que queremos hacer es "eliminar todas las lineas interiores del gráfico", quedarnos con su contorno (se obtiene el mismo resultado "siguiendo con el dedo el gráfico") para luego poder dar la solución final que explicaremos más adelante.

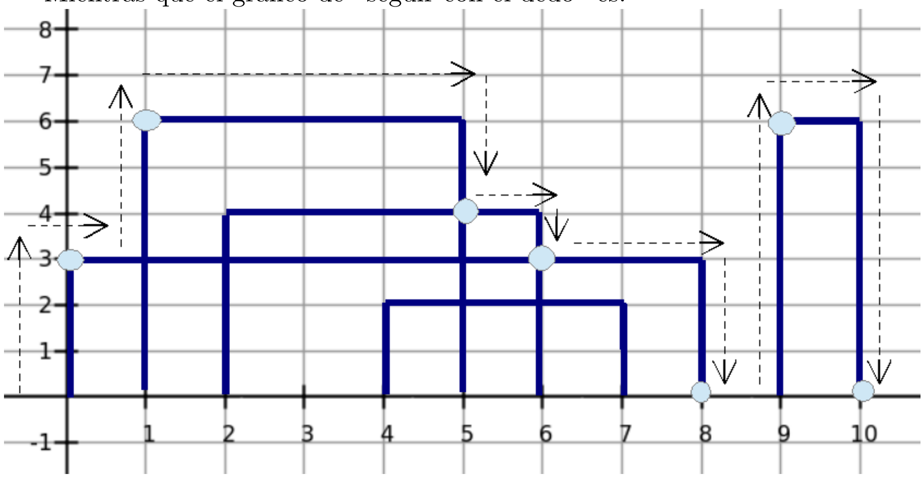
Veamos un ejemplo de eliminación de lineas interiores y el ”procedimiento” de seguir con el dedo el gráfico
Para una $lista = <0,3,8>,<1,6,5>,<2,4,6>,<4,2,7>,<9,6,10>$ con $n=5$,
su gráfico es:



el gráfico de eliminar las lineas interiores es:



Mientras que el gráfico de ”seguir con el dedo” es:



La salida para este ejemplo sería:
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 seguir 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 ”seguir 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 de altura($\uparrow \longrightarrow$ y $\downarrow \longrightarrow$).

Resolución:

Un panorama de la resolución es:
ordenar los edificios de menor a mayor por su *Izq* (en caso que tengan mismo *Izq* es menor el que tiene mayor *Alt*).
Agregar a la solución el primer punto del primer edifico de la lista de edificios ordenada.
Vamos a ir recorriendo los edificios, voy a tener un edificio *comparo* en este recorrido(al principio *comparo* es el primer edificio) y empiezo a recorrer desde el segundo, en caso de haber más de uno,el edificio por el que voy será *siguiente*.
También tengo una *cola* de prioridad (organizada por mayor Alt) donde voy a ir metiendo edificios.
Si *comparo* interseca a *siguiente*:
agrego a la solución el cambio de altura si $siguiente.Alt > comparo.Alt$.
Si $siguiente.Altura >= comparo.Alt$, ahora *comparo* es *siguiente* y encolo a *comparo* en la *cola*.
Si $siguiente.Alt < comparo.Alt$ encolo a siguiente en la *cola* y *comparo* queda donde estaba.

Si *comparo* no interseca a *siguiente*
Quiere decir que están "separados",pero puede que haya un edificio e tal que $e.Izq <= comparo.Izq$, $e.Der > comparo.Der$ y $e.Alt <= comparo.Alt$, si existiera e tendría que estar en la cola.)
Si la *cola* no está vacía voy a buscar a e en la *cola*(si existe, lo encolé cuando fui viendo los edificios anteriores a *siguiente*):
si el tope de la *cola* es e, ahora e es comparo, agrego a la solución la intersección por derecha de e con *comparo*, no lo desencolo porque pude que corte a otros edificios de más adelante.
Si el tope de la *cola* no es e desencolo y sigo buscando. Si la *cola* es vacía(quiere decir que no había ningún edificio que terminara
después que comparo) agrego a la solución el final de *comparo* y el principio de *siguiente*, ahora *comparo* es *siguiente*.
Terminé de recorrer los edificios,pero puede que hayan quedado cosas dentro de la *cola* y parte de la solución no la esté dando.
Mientras la *cola* tenga edificios, voy a comparar el tope con *comparo*:
si $tope.Der < comparo.Der$ desencolo.
En caso contrario, ahora *comparo* es *siguiente* ,agrego a la solución la intersección por derecha entre el tope y *comparo*, desencolo.
Ya no hay más edificios en la *cola*. Agregar a la solución el último punto *comparo.Der,0* y retornar la solución.
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(*números*) de longitud *3*n* y n la cantidad de edificios

```
procedure ResolverEdificios(lista,n)
    pasar a tuplas (<Izq,Alt,Der>) la lista
    ordenar los edificios por Izq, en caso de tener mismo Izq es menor el que tiene mayor Alt
    comparo ← lista[0]
    cola ← vacía
    imprimo el primer punto(comparo.Izq y comparo.Alt)
    for (i ← 1, n − 1) do //voy recorriendo los edificios a partir del segundo
        siguiente ← lista[i]
        if (comparo.Der >= iguiente.Izq(se intersecan)) then
            if (siguiente.Alt > comparo.Alt) then
                imprimir cambio de altura(siguiente.Der, siguiente.Alt)
                si la cola está vacia encolar comparo, sino, encolar comparo si el tope no es comparo
                comparo ← siguiente
            end if
            if (siguiente.Alt == comparo.Alt) then
                si la cola está vacia encolar comparo, sino, encolar comparo si el tope no es comparo
            end if
            if (siguiente.Alt < comparo.Alt) then
                encolar comparo
            end if
        else(no se intersecan comparo y siguiente)
            if (la cola no está vacía) then
                while (cola no vacía) do
                    if (topeCola.Der < comparo.Der) then
                        desencolar
                    else
                        imprimir intersección entre comparo y topeCola(comparo.Der,topeCola.Alt)
                        comparo ← topeCola
                        salir del while
                    end if
                end while
            else //no pasé a nadie que cortaría a comparo, como no se intersecan comparo y siguiente imprimo ambos puntos
                imprimir comparo.Der, 0
                imprimir siguiente.Izq, siguiente.Alt
                comparo ← siguiente
            end if
        end if
    end for//terminé de iterar los edificios,puede que hayan quedado cosas en el cola,uso a comparo que es el último edificio visto con el que haya en el tope de la cola
```

```
    for (la cola no es vacía, desencolar) do
        if (comparo.Der <= topeCola.Der) then
            imprimir intersección por entre comparo y topeCola(comaro.Der,topeCola.Alt)
        end if
        desencolar
    end for
    al último punto no lo imprimo nunca)
    //lo imprimo acá
    imprimir comparo.Der y 0
end procedure
```

Complejidad:

Al principio del algoritmo paso la lista de edificios a una lista de tuplas: cada *3* números voy a formar una tupla(la cantidad de números totales es *3*n*), el costo de crear una tupla es *c*(de asignaciones y crear cosas que son 1 operación) y de ubicarla en una lista es también 1 operación.
Entonces recorro *3*n* números y cada *3* creo un edificio, con lo cuál hago *3*n*(c+1)* para convertir a tuplas los edificios.
Luego de convertirlos a tuplas,ordeno los edificios por Izq(en caso de igual Izq es menor el de mayor Alt), el costo de ordenarlos es *n*log(n)* (porque uso sort de la stl *http://www.cplusplus.com/reference/algorithm/sort/?kw=sort*).
Vamos a ver que la cantidad de veces que encolo es una función de n, y así poder ver que la cantidad de veces que desencolo es también una función de n porque no puedo desencolar más cosas de las que encolo. Vemos en el algoritmo que en **1* y **2*, encola si está vacía o si el tope de la cola no es comparo. Luego en **3* encolo. Después a lo largo de todo el algoritmo no hago *nunca* un encolar.
Como todos estos casos son disjuntos (o son <,> o = Alt de *comparo* y *siguiente*) entonces hago 1 encolar en cada edificio en peor caso, con lo cual hago n encolar.
Veamos ahora la cantidad de operaciones 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*(costo de desencolar)).* Pero si analizamos más finamente, nunca podría para cada paso desencolar todos los edificios.
Recorriendo los edificios ordenados(*línea 3 pseudo*), si voy por el edificio i (i entre *0* y *n-1*), tengo en la cola 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(en caso de existir)* 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 i y *n-1*) ahora en la cola 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(en caso de existir) 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 en la cola(porque los edificios hasta j no están en la cola porque los desencolé), entro al *while* y desencolo *n-1-j* veces.
Si sumo la cantidad de veces que hice desencolar en el recorrido de los edificios es n (suma de los intervalos). Entonces hago n veces desencolar en peor caso 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 del *while* dentro de primer *for* es *n*(costo de encolar)+ n*(costo de desencolar) +n*c(constante)* operaciones
Para el segundo *for*, en peor caso tengo todos los edificios que son n en la cola, y desencolo hasta que se vacia,entonces hace *n*(costo de desencolar) + c1*(constante de guardas y asignaciones)) operaciones

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. *http://www.cplusplus.com/search.do?q=priority*
Finalmente la complejidad es
O(*3*n*(c+1) de crear las tuplas*
*+n*log(n) de ordenarlos*
*+ n*c+n[encolar]+n*(log(n))[desencolar] del primer for*
*+ n*c + n*log(n)[desencolar] del segundo for*) =
0((3(c+1)+c+1+c)*n + 3*n*(log(n))*
$\in$ O(n(log(n))) (acoto por el máximo y saco las constantes)

Correctitud:

El algoritmo pone en la solución (*primerEdificio.Izq,primerEdificio.Alt*).
En la solución al problema, el primer "punto" es *Der, Alt del edificio* que tiene menor *Izq* y mayor *Alt* que todos, ya que es el primer cambio de altura. ¿El primer punto que pone mi algoritmo cumple con dichas propiedades? Veamos, paso los numeros de la entrada a tuplas, ordeno los edificios(en tuplas) por *Izq* (linea 3). Entonces el primer edificio va a ser el de menor *Izq* y el de mayor *Alt*. Con lo cual si, el primer punto cumple con esas propiedades.
Voy iterando los edificios mirando el edificio por el que voy(*siguiente*) y un edificio que voy marcando en ciertos casos, *comparo*. *Comparo* no necesariamente es uno menos que *siguiente*. Al principio del algoritmo *comparo* es el primer edificio y *siguiente* el segundo en caso de haber.
Si *comparo.Der* >= *siguiente.Izq* (se intersecan):
si *comparo.Alt* < siguiente.Alt
tengo que poner el punto de cambio de altura, p, en la solución
Supongamos no tengo que poner a p en la solución:
caso existe un edificio, e que cumple: *e.Izq* < *siguiente.Izq*, *e.Alt* > *siguiente.Alt* y *e.Der* >= *siguiente.Der*(este edificio tapa al punto que quiero agregar a la solución):
(ejemplo grafico C)
Si existiera tendría que haberlo puesto como comparo,pero estoy en el caso donde la altura de comparo es menor que la de siguiente, el edificio no puede existir. Entonces hay que poner a p en la solución.
Si no se intersecan *comparo* y *siguiente*:
(grafico no se intersecan)
voy a tener en la cola los edificios que tienen *Der* mayor a *comparo.Der* porque los fui enconlando, en caso de existir, entre otros.
si la cola está vacía quiere decir que no hay nadie que corte a *comparo*(Gráfico cola vacia grafico),entonces el edificio *comparo* 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 (comparo.Der, 0) y el principio de siguiente(siguiente.Izq, siguiente.Alt).

Si la cola no está vacía quiere decir que puede que tenga un edificio que corte a *comparo* por *Der*. Grafico que corte a comparo
Voy a buscar un edificio que interseque a *comparo* por *Der*(si lo encuentro no busco más),mirando el tope, en este caso no voy a desencolar. Si el tope no es lo que busco desencolo.
Por como es la cola este que quería encontrar es el que tiene mayor *Alt* de los edificios que tienen menor *Alt* que *comparo*. Entonces este punto,p, de intersección lo agrego a la solución.
Supongamos que no tengo que agregar a p a la solución.
caso existe un edificio,e, que cumple que: *e.Izq* < *comparo.Izq*, *e.Der* > *comparo.Der* y

comparo.Alt $<=$ *e.Alt* $<$ *topeCola.Alt*
(grafico E)
Entonces este si existiera, lo tendría que haber encolado cuando recorría los edificios y tendría que ser el edificio que me daría el tope de la cola Absurdo!. Entonces tengo que agregar a p a la solución.
Siempre recorro los edificios mirando *comparo* y *siguiente*, en algún momento me voy a quedar sin edificios, *comparo* va a quedar en algún edificio y la cola va a tener algún estado (vacía o con edificios dentro)
ejemplos de estados de la cola
Ahora voy a trabajar con *comparo* y la *cola* (ahora en la cola están mis "siguientes") Si hay edificios en la cola: Quiere decir que uno de esos edificios interseque en Der a comparo.
Hasta que esté vacía la cola voy a buscar el edificio que interseque a *comparo* por *Der*,mirando el tope, si no está el tope desencolo. Si lo encuentro ahora ese es comparo y desencolo.
Por como es la cola este que quería encontrar es el que tiene mayor *Alt* de los edificios que tienen menor *Alt* que *comparo*. Entonces este punto,p, de intersección lo agrego a la solución.
Supongamos que no tengo que agregar a p a la solución.
caso existe un edificio,e, que cumple que: *e.Izq* $<$ *comparo.Izq*, *e.Der* $>$ *comparo.Der* y
comparo.Alt $<=$ *e.Alt* $<$ *topeCola.Alt*
(grafico E)
Entonces este si existiera, lo tendría que haber encolado cuando recorría los edificios y tendría que ser el edificio que me daría el tope de la cola Absurdo!. Entonces tengo que agregar a p a la solución.

A lo sumo voy a poner un punto en cada iteración, si pongo ese punto, entonces va a ser entre *comparo* y el edificio E de mayor Alt que lo pase por Der. Supongo que ese punto de intersección no es valido, entonces existe otro edificio *E'* de mayor Alt que E y que termine después que *comparo* (*E'* tapa al punto de intersección)
Si *E'.Alt* $>$ *comparo.Alt*, entonces llegamos a un absurdo ya que *E'* debería ser *comparo* porque solo cambio a *comparo* cuando el que saque de la cola lo pasa por *Der* y tiene mayor Alt, Absurdo!. Si la altura de *E'.Alt* $<=$ comparo.Alt, entonces como la altura de *E'* es mayor que la de E (para que tape el punto de intersección), tendría que ser *E'* el tope de la cola y no E, Absurdo.Con lo cual el punto es valido.
Demostramos que todos los puntos que pone nuestro algoritmo son válidos.
Ahora tenemos que demostrar que esos puntos son todos los de la solución, osea que no olvidamos ninguno. Sea S la solución (de puntos ordenados) al problema, quiero ver que todos los puntos de S están en la solución que dí. El primer punto sabemos que lo vamos a poner siempre porque está al principio del algoritmo El resto de los puntos exceptuando el primero, podemos dividirlos en *ascendentes* (cuando el punto anterior es mas bajo), *descendentes* (cuando el punto anterior es mas alto) y cambios de altura a 0. Dado cualquier punto descendente de S con

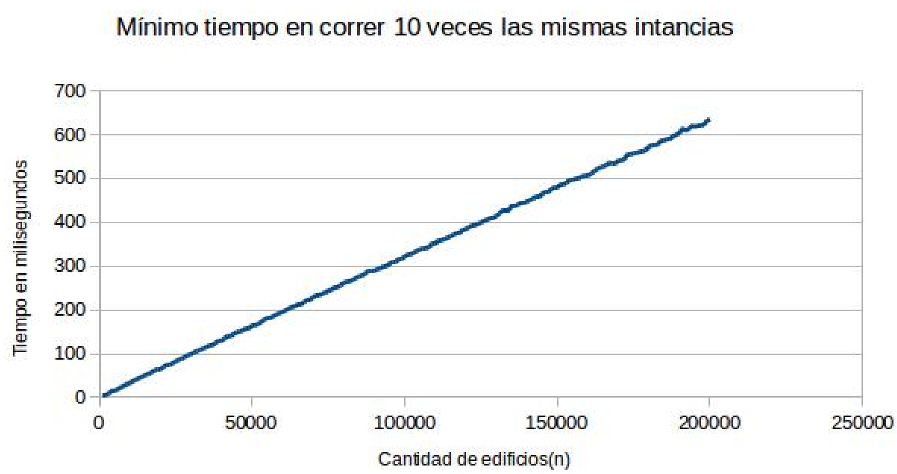
altura distinta de 0, quiero ver que está en la solución que devuelve el algoritmo. Como la altura es distinta de cero, entonces el punto debe ser la intersección entre dos edificios *E1* (mayor altura) y *E2* (menor altura). Si podemos probar que eventualmente va a valer *comparo = E1*, entonces estaríamos probando que vamos a poner el punto en la solución ya que vimos que, como en la cola tenemos todos los edificios menores que *comparo* que terminan despues que *comparo* (entre otros más que también pueden estar en la cola), vamos a desencolarlo y agregar el punto a la solución.
Supongamos que no existe ningun estado del algoritmo en el cual *comparo = E1*, entonces seguro existe un edificio *E'* tal que tapa a *E1* ya que, nuestro algoritmo cambia el valor de *comparo* cuando encuentra un *siguiente* tal que la altura de *siguiente* es mayor a la de *comparo* o, cuando estoy desencolando pero cuando desencolo me quedo con los que terminan despues de *comparo* y, si *E'* tapa totalmente a *E1* entonces *E1* va a ser desencolado pero no va a terminar después que *comparo* y por lo tanto comparo no va a cambiar de valor. Pero, si existe *E'* tal que tapa a *E1*, entonces particularmente va a tapar al punto de interseccion denotado por *E1* y *E2*, entonces ese punto no va a ser solucion y habiamos supuesto que si lo era. Llegamos a un absurdo proveniente de suponer que existe una solucion denotada por dos edificios *E1* y *E2* y que *E1* es distinto a *comparo* para cualquier estado del algoritmo. Por lo tanto, *E1 = comparo* para algun estado del algoritmo y entonces eventualmente vamos a llegar a identificar el punto de intersección cuando desencolemos.

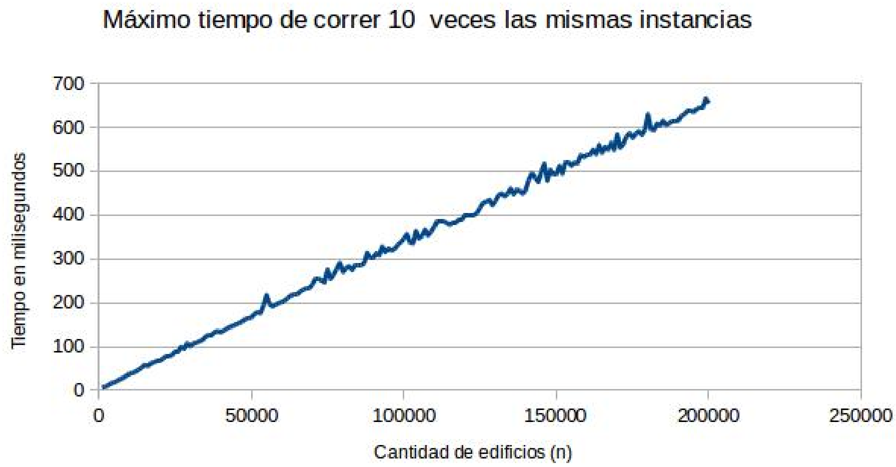
Ahora, queremos ver que vamos a agregar todos los cambios de altura a cero. Dado que en cada iteración me fijo si el *siguiente* edificio interseca a *comparo* y, en el caso de que no lo interseque y la cola vacía entonces agrego el punto, si pasa esto quiere decir que no existe ningún edifició que termine después de *comparo* y empiece antes de que finalice *comparo*. Por lo tanto *comparo* es el último edificio antes del espacio denotado por el cambio de altura a 0. Entonces el punto es efectivamente una solución.
Si la cola tiene edificios, quiere decir que pueden existir edificios que terminan después que *comparo* y, como voy a actualizar el valor de *comparo* al edificio más alto de la cola que termine despues que el edificio que contiene "ahora" *comparo*, si efectivamente existía un espacio entre dos edificios (un cambio de altura a 0), entonces lo voy a detectar. ya que voy a repetir este procedimiento por cada edificio que no esté contenido por otro (porque va a ser comparo).
Luego para los puntos ascendentes: Como por cada iteración nos fijamos si la altura de *comparo* es menor a la altura de *siguiente* y, en el caso de que se intersequen, agregamos el punto (*siguiente.Izq, siguiente.Alt*) a la solución.
Además, en el caso de que haya un cambió de altura a 0 y existe un edificio *siguiente*, ya demostramos que ese cambio de altura lo agregamos. Además de agregar ese cambio de altura a 0, agregamos el cambio de altura a la altura de *siguiente*, que representa un punto de altura creciente. Ya que vimos que los cambios de altura a 0 los agregamos todos y que el algoritmo verifica siempre si *comparo* tiene altura menor que un edificio *siguiente* que lo interseca, vemos que no puede existir un punto de crecimiento que no lo agreguemos porque estamos viendo los dos casos donde aparece dicho punto.

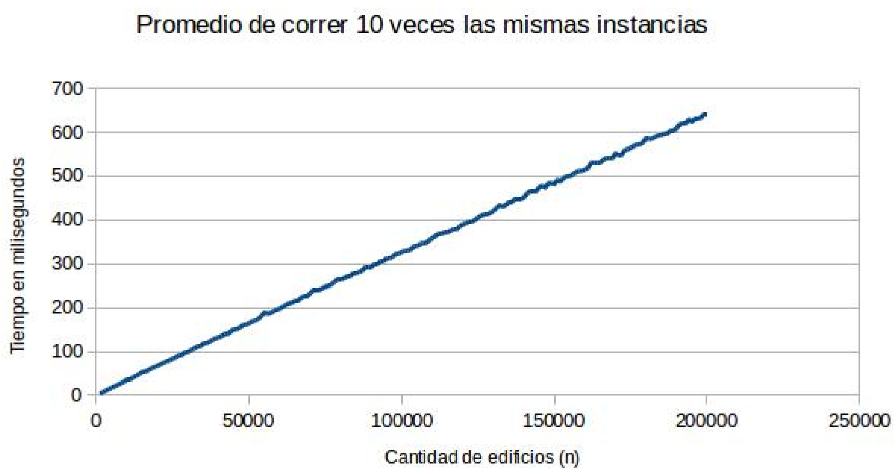
Con lo dicho anteriormente ya quededa demostrado que todos los puntos de la solución los da el algoritmo.

Experimentación:

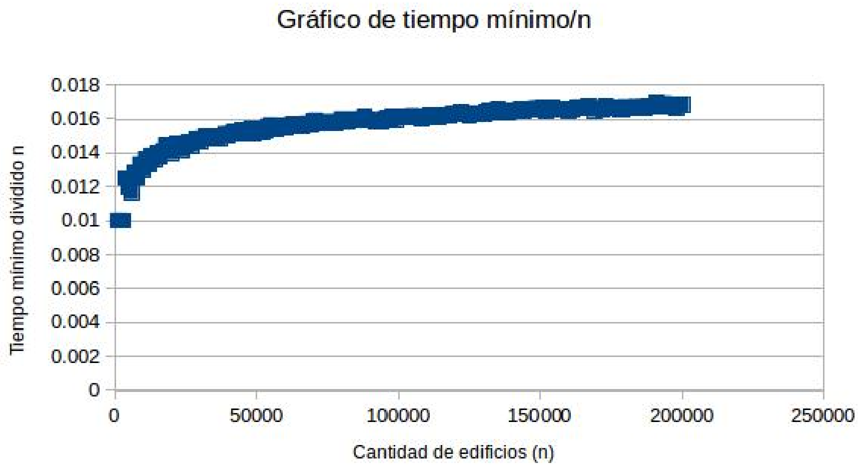
Se han generado 200 instancias donde la instancia $n_i = n_{i-1} + 1000$ y $n_1 = 1000$. Por cada instancia, vale que para todo edificio e generado, $e.Izq < e.Der$ y, tanto $e.Izq$ como $e.Der$ y $e.Alt$ se han elegido de manera aleatoria en el rango [0, 49]. El rango puede sonar arbitrario y, si bien lo es porque podríamos haber elegido otro, también resulta razonable desde el punto de vista que permite que, cuando n es muy grande, entonces los edificios se superpongan más, generandose así más intersecciones entre ellos y haciendo que el algoritmo deba trabajar más para decidir cual intersección es un punto de la solución y cual no lo es.
Se ha corrido el algoritmo 10 veces sobre cada una de las instancias. Luego, se calculó el tiempo mínimo, máximo y promedio por cada instancia, a continuación mostramos los gráficos correspondientes:



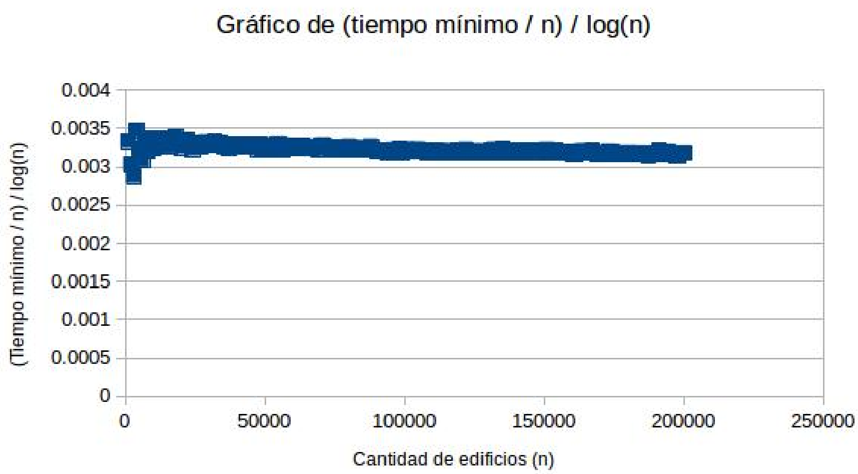




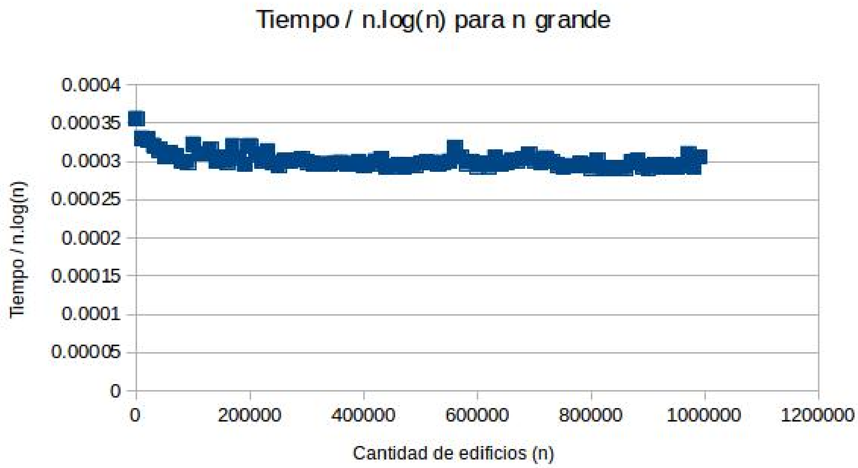
Dado que nuestro algoritmo va a correr paralelamente con otros procesos, en algunas ejecuciones se va a demorar más que en otras. Tomando el tiempo mínimo como el más significativo (es decir, como la ejecución en la cual fue "menos interrumpido"), si dividimos ese tiempo por n, obtenemos el siguiente gráfico:



Con los gráficos anteriores no podíamos decir mucho ya que, aparentemente aproximan a una recta, pero como al dividirlos por n obtenemos una función que creciente y no una constante, efectivamente teníamos (en el gráfico de tiempo mínimo) una función perteneciente a $\Omega(n)$.
Si nuevamente dividimos esta última función, esta vez por $log(n)$, obtenemos lo siguiente:



Vemos que los puntos parecieran tender al valor 0.003. Si recapitulamos, lo que hicimos fue tomar los valores de tiempo mínimo, dividirlos por n y a ese valor dividirlo por $log(n)$ y llegamos a valores que tienden a un número constante $k \simeq 0.003$.
Si aumentamos aún más el n máximo y hacemos este mismo análisis, para instancias de hasta $n = 991000$ obtenemos el siguiente gráfico:



en el cual podemos observar que el gráfico sigue tendiendo a una constante $\simeq 0.003$.
Concluimos entonces que los tiempos tomados concuerdan con la complejidad teórica calculada $O(n.log(n))$