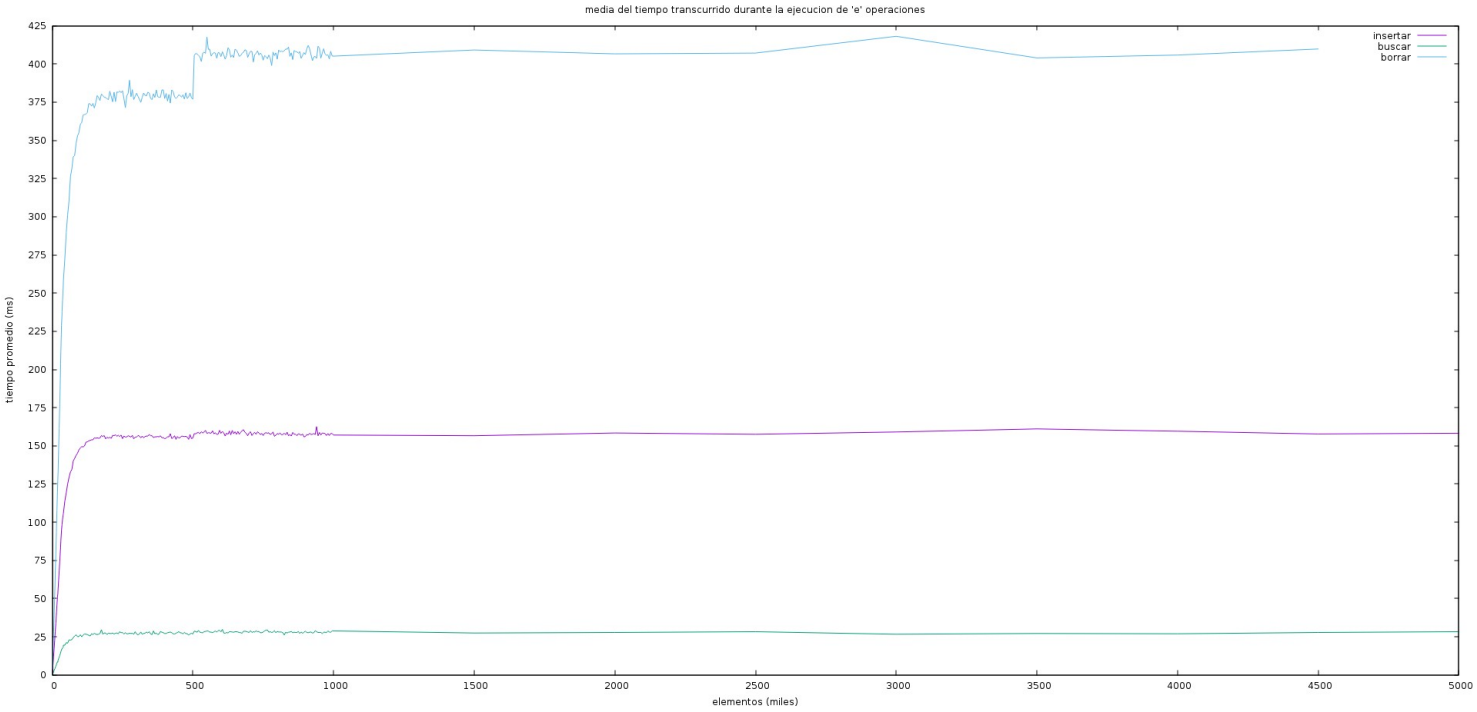
Gráficas de tiempos de ejecución

Una prueba de *e* elementos para cualquier operación está compuesta por dos partes:

- la generación de dos conjuntos, uno de claves y otro de valores
  (tamaño máximo = *e*, al existir la posibilidad de generar claves repetidas)
- las ejecuciones de la operación y su correspondiente medición de tiempo
  (llamadas a la operación = numero de claves únicas, generadas al principio de la prueba)

Se ejecutaron *p* pruebas de *e* elementos para cada operación. El tiempo transcurrido en ejecutarse una prueba se sumó a una variable acumuladora para luego dividirla por *p*. El resultado de esta división es la media del tiempo total al ejecutar *e* llamadas a la operación siendo probada. A continuación se puede ver una gráfica de la relación entre esta media y el numero de elementos:



La operación borrar es la más costosa de las tres, con diferencia. Esto es debido a la necesidad de manipular el árbol aplicando rotaciones y colorflips durante la bajada, que se hace para mantener el invariante de que el nodo actual no es un nodo-2. Gracias a este invariante se puede borrar cualquier nodo haciendo uso del nodo sucesor (si existiese), que se localiza fácilmente con la siguiente función:

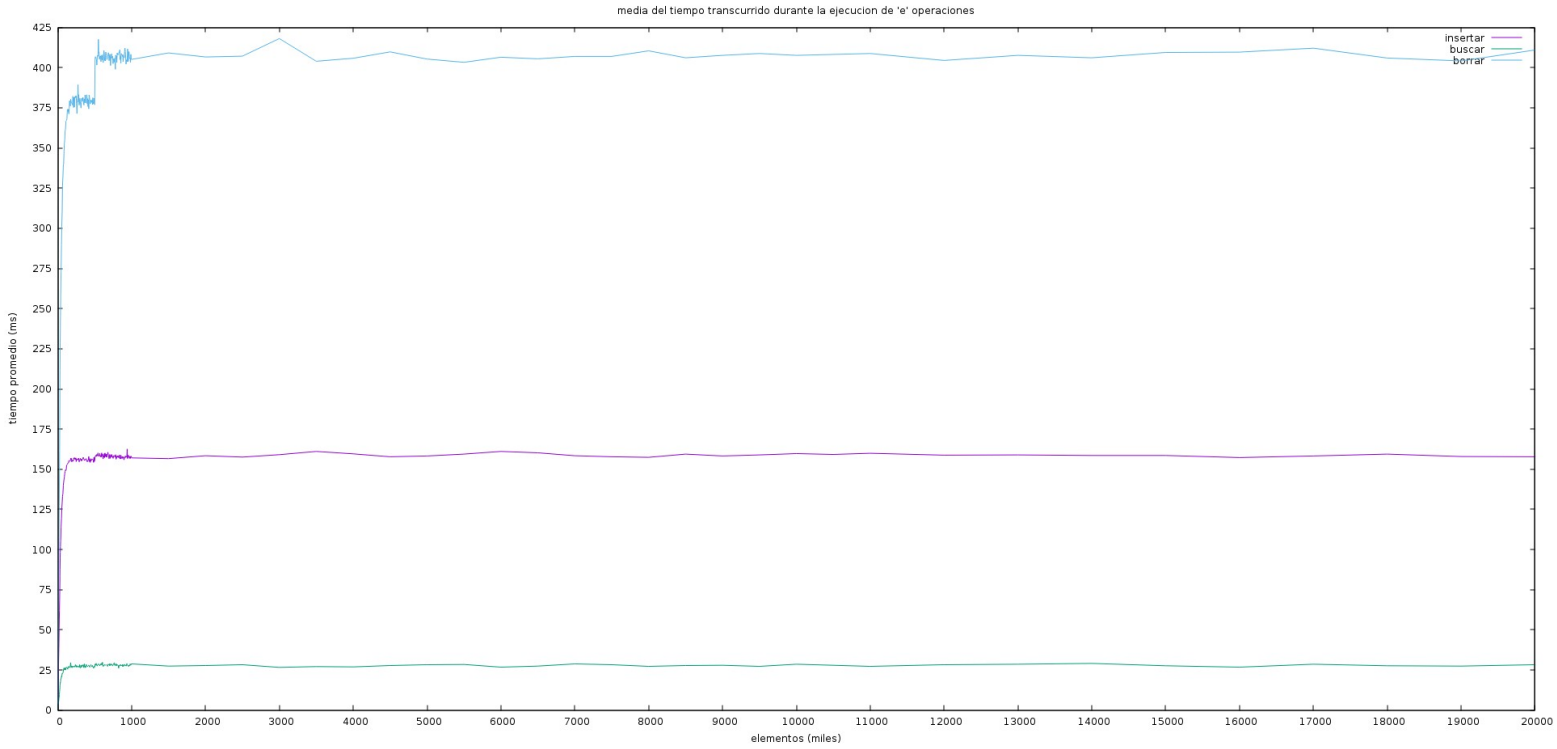
```
nodeptr min_node(nodeptr h)
{ return h->_left != nullptr ? min_node(h->_left) : h; }
```

Por muy sencillo que parezca, las llamadas de manipulación durante la bajada requieren una compensación, que viene a ser la ejecución de fix_up. Esta función se ejecuta con cada nodo durante la subida a la raíz, con lo cual añade aun más comparaciones, más rotaciones, y más colorflips al borrado de un nodo cualquiera, resultando en una llamada mucho más costosa que las de insertar y buscar.

La operación insertar es prácticamente igual a la de un árbol binario de búsqueda. Tras insertar un nodo, se contemplan casos casi idénticos a los d fix_up para mantener el invariante del árbol, igual que al subir de vuelta a la raíz tras haber borrado un nodo.

La operación buscar es igual a la de un árbol binario de búsqueda.

La siguiente gráfica es la misma que la anterior, pero cubre un rango más amplio.



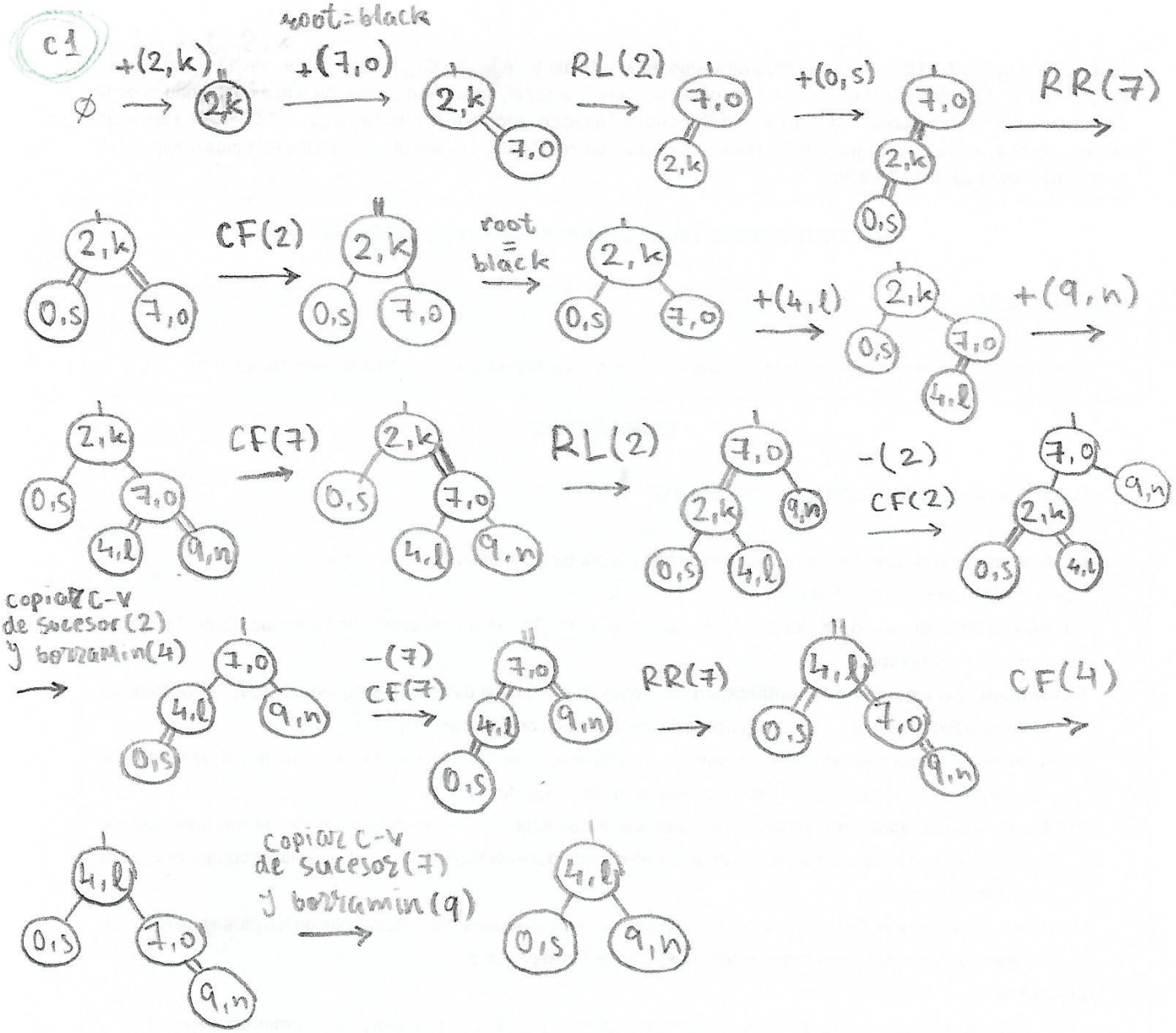
Para las operaciones de borrar e insertar se puede ver un escalón en los tiempos entre 500.000 y 505.000 elementos. En la curva de la búsqueda no existe este escalón, y en la del borrado es un escalón exagerado. No estoy seguro de la posible causa de este salto, pero supongo que tendrá alguna relación con el formato de prueba de las operaciones.

Todas las operaciones presentan algo parecido a una curva logarítmica, y, basándome en los datos, estoy casi seguro de que todas las operaciones tienen un peor coste de O(n log(n)).

Casos de prueba

A continuación se encuentra la secuencia de operaciones de inserción para cada caso de prueba. Después de leer todos los valores del fichero, se borran algunos elementos. Se representan gráficamente todas las operaciones que modifican el árbol. Los vínculos rojos se representan con una doble linea entre nodos, en lugar de una, que representa un vínculo normal.
RR/RL = rotateRight / rotateLeft, CF = colorFlip

cargar c1.txt -> borrar(2) -> borrar(7)



cargar c2.txt -> borrar(3)