Mario López González
Marina Muñoz Cano

COMPARATIVA CONTENDORES DE LA STL

Hemos comprado cuatro tipos de contenedores:

- vector<int>
- list<int>
- set<int>
- unordered_set<int>

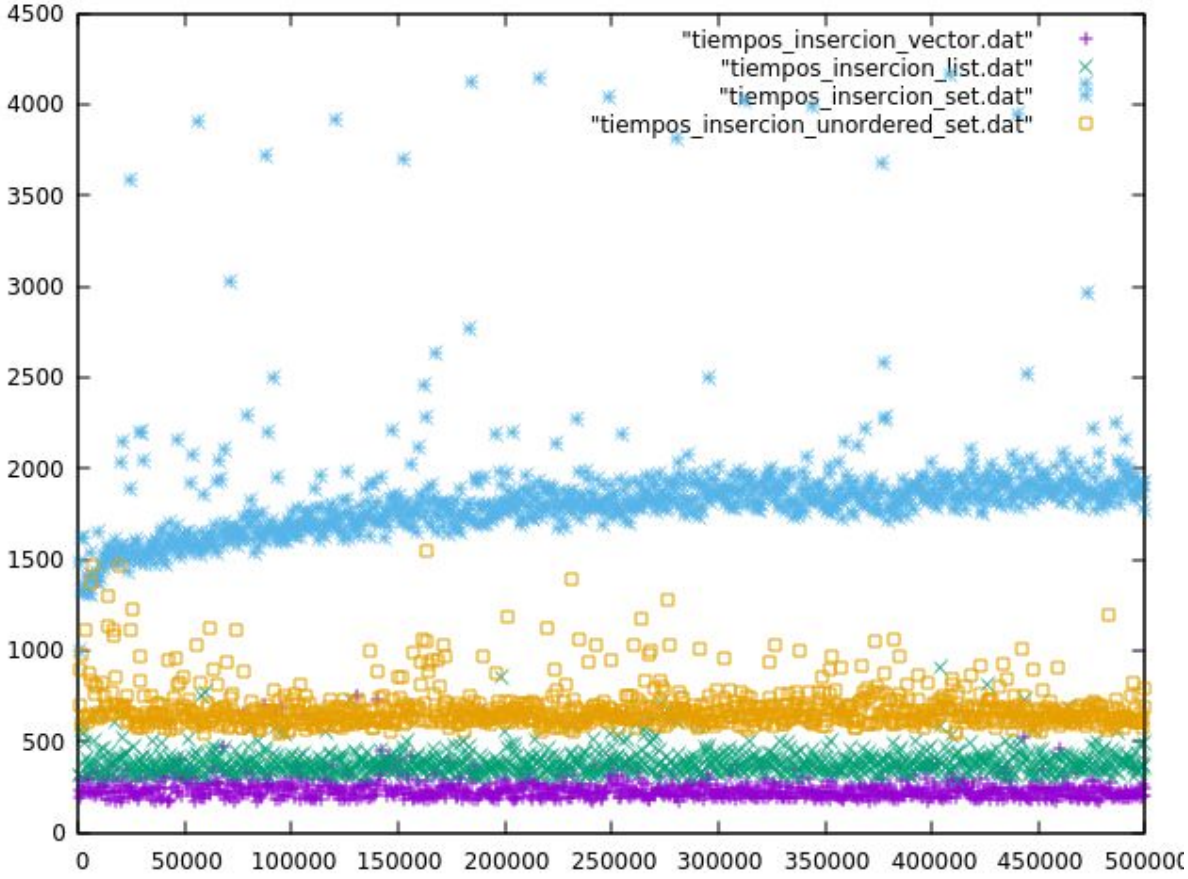
Con las siguientes operaciones:

Inserción

Para la inserción usamos los métodos que describimos a continuación.

- vector y list: push_back() que inserta un elemento al final del contenedor.
- set y unordered_set: insert() inserta un elemento en la posición pasada como parámetro.

Para cada caso hemos realizado ejecuciones con tamaños de 500 a 500.000 con un incremento de 500. A continuación se muestra la gráfica con el resultado de las ejecuciones en cada caso.



Como podemos observar tanto list como vector su tiempo de ejecución es constante, lo que concuerda con la información sobre la complejidad de la operación que se expone en cplusplus.com.

La inserción en set es un poco más costosa que la del resto, la complejidad de esta operación según cplusplus.com es de N*log(size+N), siendo N el número de elementos insertados, por tanto en nuestro caso sería log(size+1), aunque si se insertase en la posición óptima, el tiempo podría llegar a ser constante. Como se observa en la gráfica hay una ligera curva que nos indica que efectivamente se trata de una eficiencia de log(n).

La inserción en unordered_set también se realiza en un tiempo constante, según cplusplus.com en la mayoría de casos, su ejecución es constante, pero en el peor de los casos su ejecución sería una ejecución lineal.

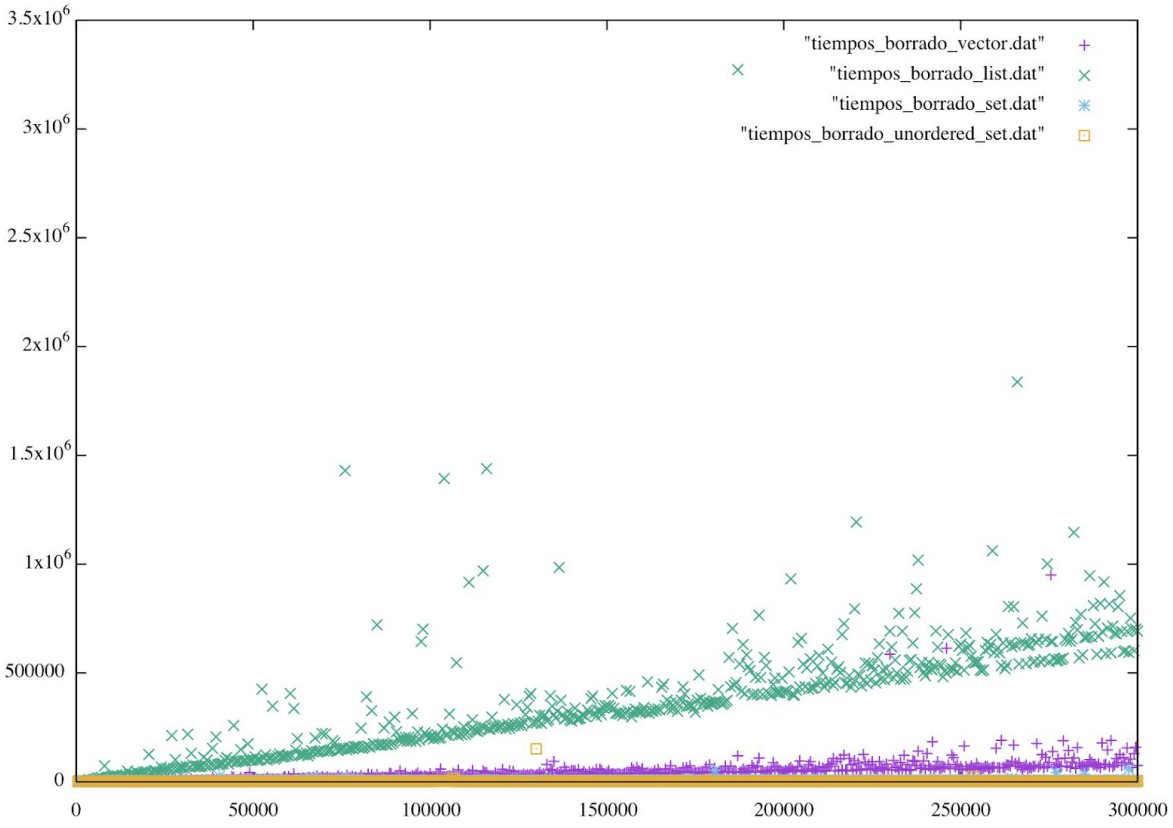
Por tanto podemos concluir que los más eficientes en esta operación son vector, list y unordered_set.

Borrado

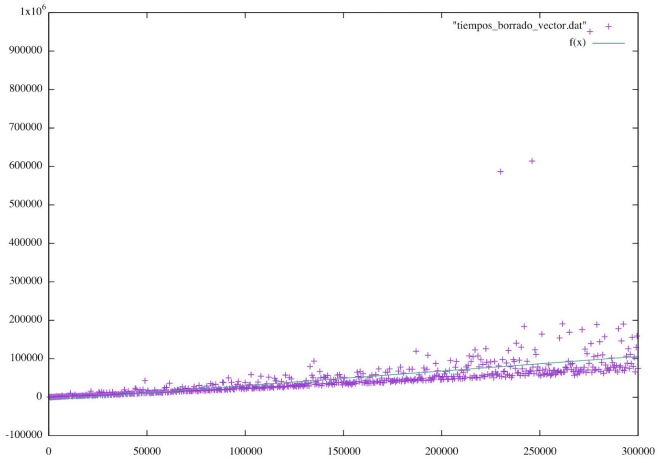
Para el borrado usamos los siguientes:

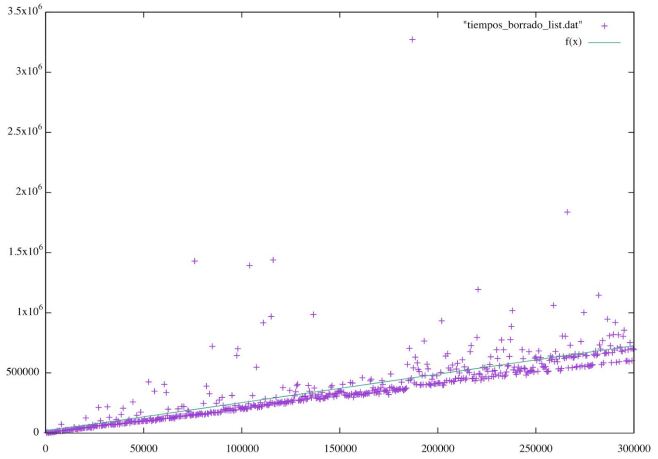
- vector, set y unordered_set: erase(N/2) que elimina el elemento en la posición N/2 (o de clave N/2 en el caso de set y unoredered_set).
- list: se usa también erase() con la diferencia de que tenemos que mover el iterador previamente hasta la posición que queremos borrar.

Para cada caso hemos realizado ejecuciones con tamaños de 500 a 300.000 con un incremento de 500. A continuación se muestra la gráfica con el resultado de las ejecuciones en cada caso.

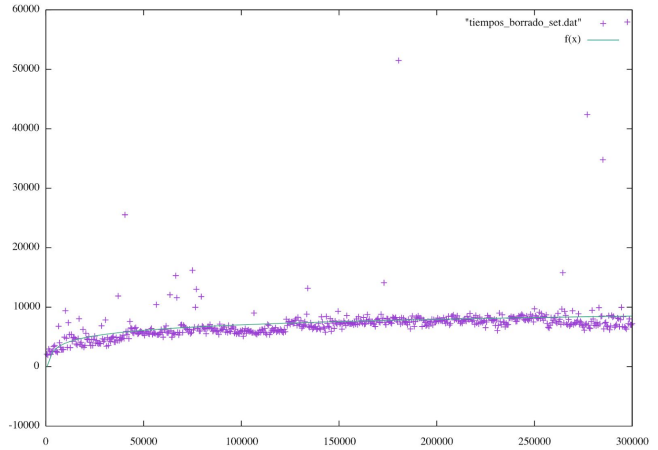


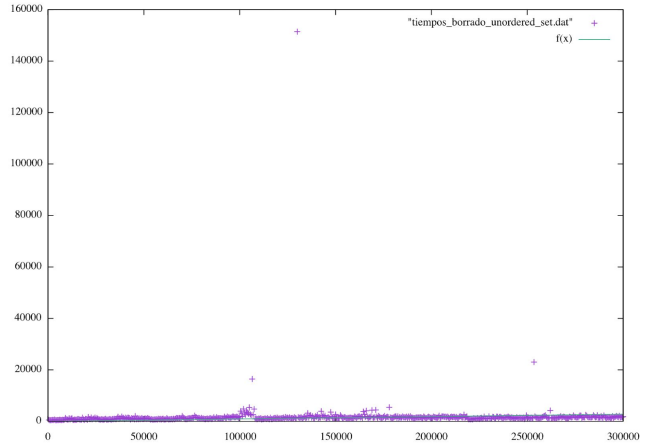
Tanto list como vector según cplusplus.com se realizan en un tiempo lineal (O(n)). Como se puede observar en las gráficas mostradas a continuación.





El tiempo de borrado de set según cplusplus.com es constante si lo que le pasamos como parámetro a erase es una posición (iterator). Si el argumento es un valor de clave el tiempo es logarítmico. Si borramos entre dos posiciones el tiempo sería lineal. Al pasarle un valor, como observamos en la gráfica el tiempo experimental tiende a log(n).





En el caso de unordered_set, el borrado de un elemento se realiza en un tiempo constante. Como se especifica en cplusplus.com el tiempo de borrado es lineal en el número de elementos que se quieren borrar (N en el peor caso), como en este caso eliminamos un sólo elemento, efectivamente el tiempo es constante.

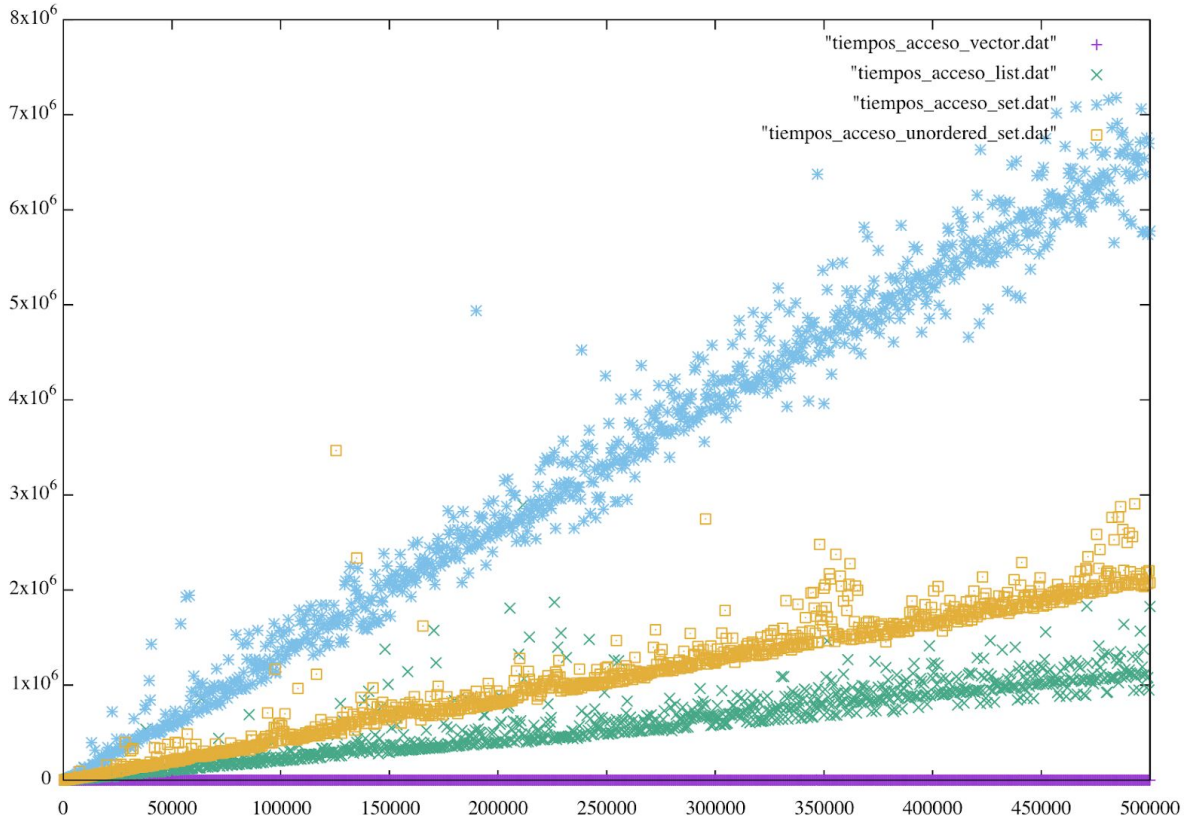
Por tanto, en el borrado de un elemento el más eficiente es unordered_set, seguido de set y los menos eficientes serían list y vector.

Acceso

Para el acceso usamos los métodos descritos a continuación:

- vector: operator[] consultando la posición N/2
- list, set y unordered_set: incrementamos la posición de un iterador hasta N/2 usando advance() y accedemos al elemento mediante el operator* de la clase iterator.

Para cada caso hemos realizado ejecuciones con tamaños de 500 a 500.000 con un incremento de 500. A continuación se muestra la gráfica con el resultado de las ejecuciones en cada caso.



El acceso a un vector con el operador [] se realiza en tiempo constante según cplusplus.com, como se puede observar en el gráfico, efectivamente es así.

En el resto de casos la función advance() que incrementa la posición de un iterador en la cantidad que le indiquemos hemos comprobado que se realiza en un tiempo lineal como especifica cpluscplus.com.

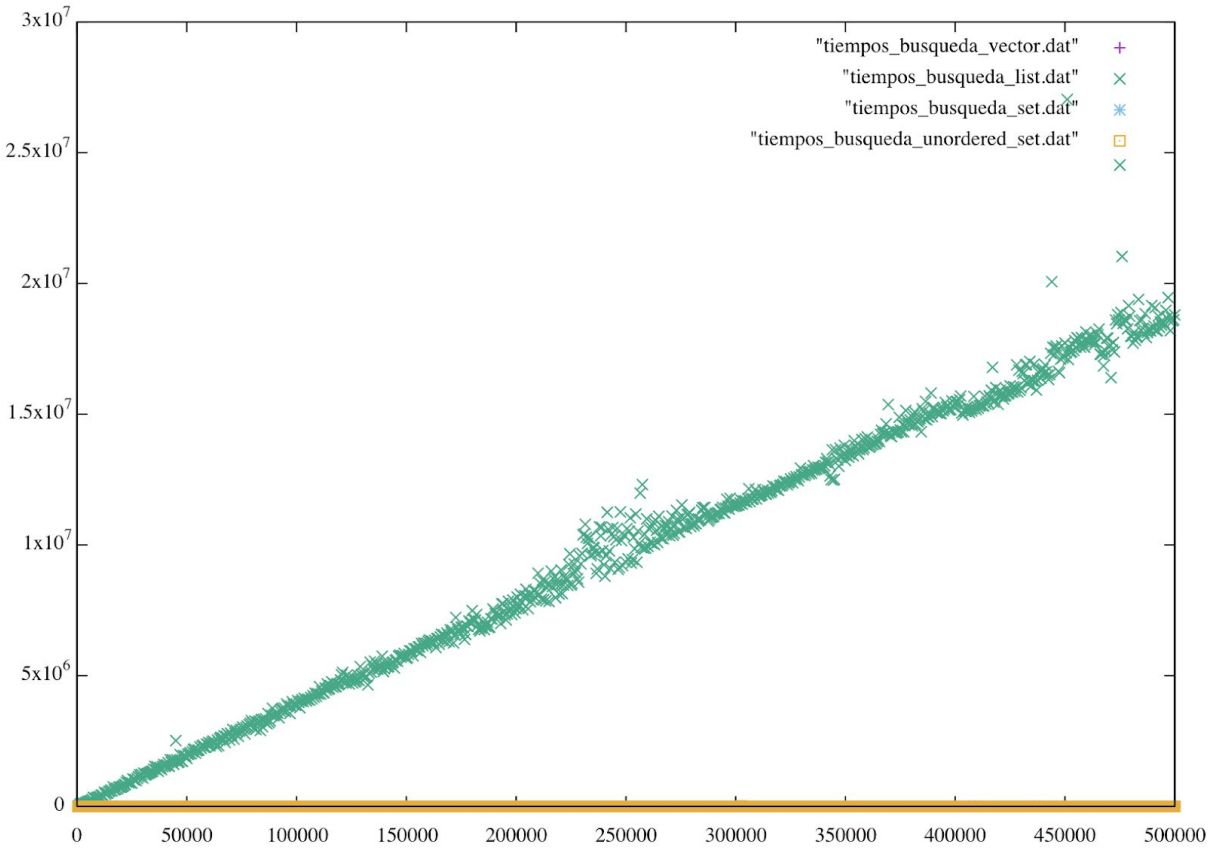
Por tanto, podemos concluir que para el acceso de elementos que no se encuentren al principio o al final del contenedor, el mejor sería vector, ya que lo realiza en tiempo constante, seguido del resto cuyo tiempo es lineal.

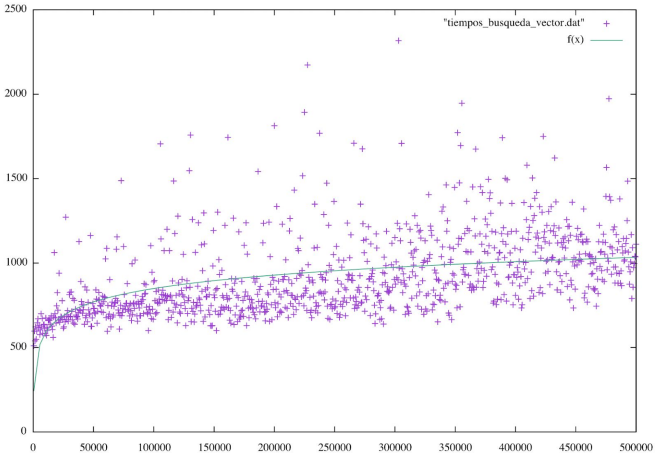
Búsqueda

Para el acceso usamos los métodos a continuación:

- vector: usamos la función binary_search() de algorithm
- list: recorremos con un for desde begin() hasta encontrar el elemento buscado (end() si no lo encontramos).
- set y unordered_set: usamos el método find().

Para cada caso hemos realizado ejecuciones con tamaños de 500 a 500.000 con un incremento de 500. A continuación se muestra la gráfica con el resultado de las ejecuciones en cada caso.

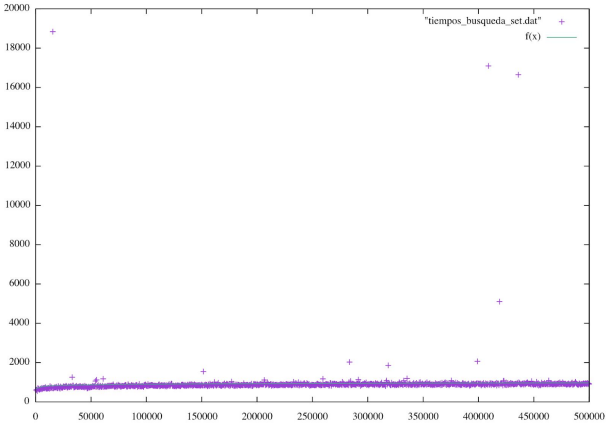


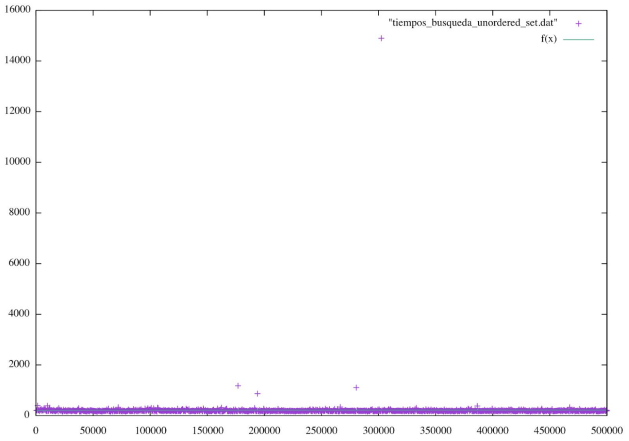


En el caso de la búsqueda binaria su eficiencia es de $O(\log_2 N)$. En la gráfica podemos observar que aunque las muestras están muy dispersas se concentran en la zona que corresponde con la función $\log_2 N$.

En list podemos observar en la gráfica en la que hemos representado todas las operaciones que al tener que recorrer la lista de principio a fin obtenemos un tiempo lineal.

En set el método find() según cplusplus.com tiene complejidad logarítmica. Como se observa de forma experimental efectivamente se ajusta a un a curva logarítmica.





El método find() de la clase unordered set, es de tiempo constante en la mayoría de casos y lineal en el peor de los casos. En nuestro caso, como se puede observar en la imagen, la ejecución es de tiempo constante.

Por lo expuesto previamente, podemos determinar que el mejor contenedor para búsqueda de elementos es unordered_set puesto que su tiempo de ejecución es constante. En segundo lugar, se encuentran vector y set con un tiempo logarítmico. En último lugar, el más costoso es list con un tiempo de ejecución lineal.