Benjamin Miller

Multiset vs. Vectors

Hypothesis:

I expect the vector to be much slower than the multiset. I think that the vector will be slower because it has to do a binary search for every new insertion and basically do what the multiset does, but less efficiently. Although, since vectors are so widely used I think that they will perform well and will not cross the three second mark until over 10,000,000 iterations.

Methods:

For this experiment I used CLion by Jetbrains to run the code. I created a new project in c++ 14 and put my code into the file that ended with ".cpp". I did not do anything special for the compiler or any optimizations. Once the code was added, I simply hit the run button. For the code in the main function I did these steps:

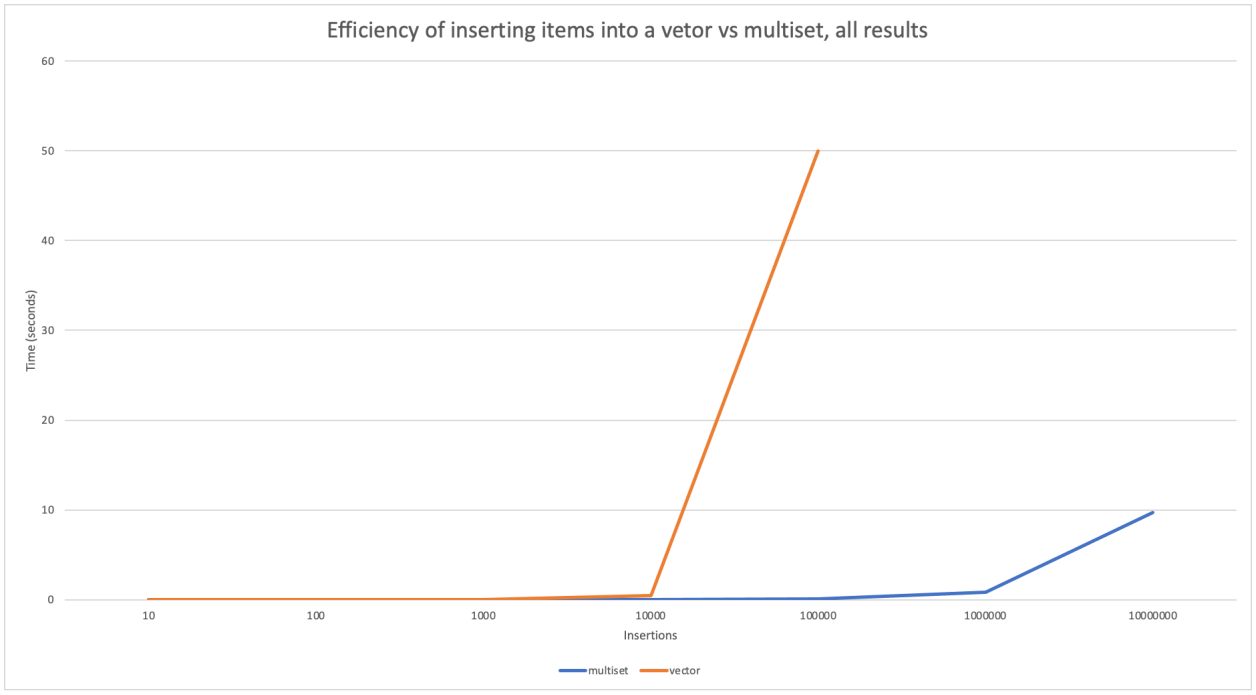
1. Define a variable of n number of insertions, a multiset, and a vector
2. Start the timer and create a loop of n number of insertions
3. Get a random number between 1 and 1000 at the start of each iteration
4. Use a binary search to find the insertion index of the random number for the vector, and then use insert to add the number at that index
5. Record the time it took for all insertions to take place, then increase the number of insertions until the time is greater than 3 seconds
6. Repeat steps 4 and 5 with a multiset and use insert, the binary search is not needed here

The auto generated makefile:

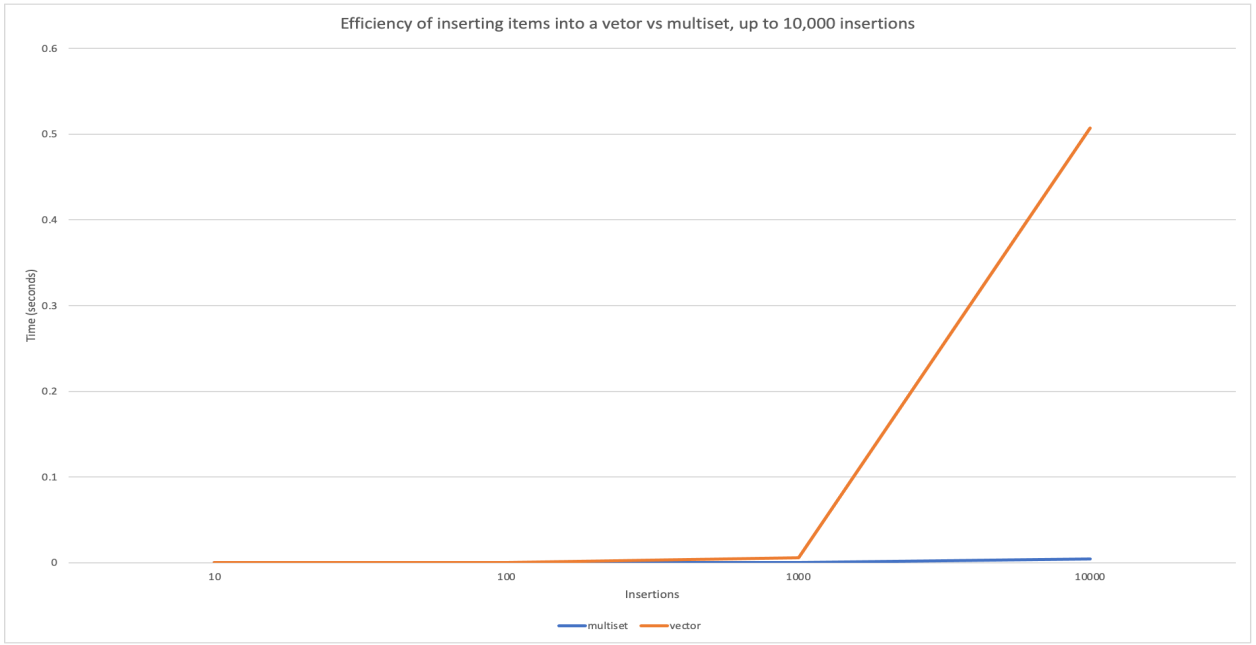
```
cmake_minimum_required(VERSION 3.17)
project(431question4)

set(CMAKE_CXX_STANDARD 14)

add_executable(431question4 main.cpp)
```

```cpp
#include <iostream>
#include <set>
#include <vector>

using namespace std;

// https://www.codegrepper.com/code-examples/cpp/binary+search+tree+c%2B%2B
// This program performs a binary search through an array, must be sorted to work
// Modified to return index instead of if value exists
int binarySearch(vector<int> array, int size, int value)
{
    int first = 0,          // First array element
    last = size - 1,        // Last array element
    middle;                 // Mid point of search
    bool found = false;         // Flag
    while (!found && first <= last)
    {
        middle = (first + last) / 2;      // Calculate mid point
        if (array[middle] == value)       // If value is found at mid
        {
            found = true;
        }
        else if (array[middle] > value)  // If value is in lower half
            last = middle - 1;
        else
            first = middle + 1;           // If value is in upper half
    }
    return first;
}

int main() {
    int insertions = 10000000;
    multiset<int> multisetTest;
    vector<int> vectorTest;

    std::clock_t start_time = std::clock();

    srand(time(0));
    for(int i = 0; i < insertions; i++) {
        int randomNum = rand() % 1000;

        /*
        if (i == 0) {
            vectorTest.insert(vectorTest.begin(), randomNum);
            continue;
        }
        int result = binarySearch(vectorTest, vectorTest.size(), randomNum);
        vectorTest.insert(vectorTest.begin() + result, randomNum);
         */

        multisetTest.insert(randomNum);
    }

    std::clock_t tot_time = std::clock() - start_time;
    std::cout << "Time: "
              << ((double) tot_time) / (double) CLOCKS_PER_SEC
              << " seconds" << std::endl;

    return 0;
}
```

Results:



The results of the experiment. The graph shows the time it took for n number of insertions for a vector (orange) and a multiset (blue). The data for each stops once the time becomes greater than 3 seconds.



The experiment results up to 10,000 insertions. The data is the same as the first graph, but the chosen insertion totals allow for a better view of the time for those smaller totals.

Discussion:

For the data I collected I did not have any real surprises. The biggest one for me was that vectors perform even worse than I thought. I did not have any challenges for this question either. Overall a sorted vector can be pretty efficient up to 10,000 insertions, but after that it becomes too slow to be very useful. The multiset data I collected is new for this problem, but is very similar to question 3, so no surprises for that.

Conclusions:

Under the conditions tested, the vector with binary search is less efficient for all n numbers of insertions compared to a multiset.