IB9N7 C++ for Quantitative Finance
Worksheet 14

Sorting and searching

18 February 2016
(Week 20)

Objectives for this lab session

By the end of this session, you should have completed the following:

- Look at some examples of sorting and searching.

Exercises

Exercise 1: Set using vectors

The aim of this exercise is to create an object which represents an accumulating set of `int`s. It allows us to remember whether a number has been seen before. It should have two public member functions (i.e. these functions form its interface).

```
void add(int a); //add a to the set
bool isPresent(int a) const; //return whether a is in the set.
```

(a) Create a class `Set1` which implements this interface, by storing all the values which have been added in a `std::vector<int>` data member. `add` should just `push_back` into this member.

(b) Create a class `Set2` which implements this interface, by storing all the values which have been added in a `std::vector<int>` data member which is sorted in ascending order. `add` should insert the new value into the right place, and `isPresent` should check efficiently. Hint: implement both of these functions using `lower_bound`.

Exercise 2: Vectors and lists

(a) Setup a vector which contains random integers between 0 and 20 inclusive by copying (and adapting if you want) this:

```
#include<random>
#include<vector>
int main(){
    std::vector<int> a;
    std::mt19937 gen;
    std::uniform_int_distribution<int> uid(0,20);
    for(int i=0; i<200; ++i){
        a.push_back(uid(gen));
    }
}
```

(b) Sort your vector into ascending order.

(c) Calculate the median of your vector. (Hints: it is easier to do this without using iterators to begin with. It helps that your vector is already in sorted order.)

Extra: consider how you might do this if the vector was not originally sorted.

(d) Now that your vector is sorted, insert the number '10' in the appropriate place to maintain the sorted invariant.

(Note that you should wrap this feature into a function instead of doing it directly in your `main` function.)

(e) Remove all odd numbers from your vector.

Take care with respect to iterator invalidation.

Hint: The slow but safe way to do this is to restart your search at the beginning of the vector once you have removed an element. Once you have done that, try to find a more efficient but equally valid method.

(f) Determine whether the numbers 15 or 20 each appear in your vector (just "yes" or "no" will do). You may experiment with the `std::find` algorithm if you want.

(g) If the above numbers appear, multiply **all** instances of them by 2, replacing the original copies in the vector.

(h) Now try to do all of the above using a list instead of a vector.

When would you use a list instead of a vector?

Which is easiest or most appropriate for calculating the median?