CSCI 262  
Data Structures  
20 – Sorting

**Sorting**  
- Input: a list of elements, e.g. integers  
- Output: a list of the input elements in sorted order

Why do we study this problem?  
- Teaching example  
  – Algorithm design  
  – Algorithm analysis  
- Sorting is also useful for all sorts of applications

A simple solution:  
- Find the minimum element in the list  
- Swap it with the first element in the list  
- Sort the sublist after the first element

This sorting algorithm is named “selection sort”.

---

**Selection Sort Illustrated**  
5 7 2 10 3 1 6  
- Find min element  
7 2 10 3 1 6  
- Swap with first element  
7 2 10 3 5 6  
- Sort the remaining elements

**Selection Sort Code**
```cpp
template <typename T>
void selection_sort(Vector<T> & vec) {
    int n = vec.size();
    for (int left = 0; left < n; left++) {
        int right = left;
        for (int j = left + 1; j < n; j++) {
            if (vec[j] < vec[right]) right = j;
        }
        swap(vec[left], vec[right]);
    }
}
```

**Analyzing Selection Sort**  
Recall we want to count basic computer steps...

```cpp
template <typename T>
void selection_sort(Vector<T> & vec) {
    int n = vec.size();
    for (int left = 0; left < n; left++) {
        int right = left;
        for (int j = left + 1; j < n; j++) {
            if (vec[j] < vec[right]) right = j;
        }
        swap(vec[left], vec[right]);
    }
}
```

What is \( x \)? Ans: \( n - left - 1 \).  
How do we add these up?
Analyzing Selection Sort

Things we can easily count:
1 step (line 3)
4n steps (lines 5 and 10)

Things that are trickier:
$n - \text{left} - 1$ (different value of left each time)

Putting it all together, we have:
Cost of selection sort is

\[ 1 + 4n + n(n - 1)/2 \]

What is the “big-O” of this expression?

Analyzing Selection Sort

Just have to count carefully:
1st time through:
left = 0, so $n - \text{left} - 1 = n - 1$
2nd time through:
left = 1, so $n - \text{left} - 1 = n - 2$
... 
Last time through:
left = $n - 1$, so $n - \text{left} - 1 = 0$

Visual Analysis

Preceding pages were very rigorous in counting
Sometimes, a visual approach is simpler:

Cost: $O(1 + 2 + \ldots + n)$

Analysis Complete

Selection sort is $O(n^2)$

Can we do better?
(Yes)

Sorting in Standard Library

Sorting in the C++ standard library works with
iterators
- Iterators let you step through a collection item by item
- In some ways like pointers (and pointers into an array are considered iterators for that array)
- Basis of range-based for loop
- Mostly, a topic for another time

For now, just note that you can sort any random access collection (Vectors, strings, arrays)
sort

Sorting a Vector:

#include <algorithm>

Vector<int> vec = {17, 42, 100, -3, 50};
sort(vec.begin(), vec.end());
cout << vec << endl;

sort

Sorting a string:

#include <algorithm>

string s = "Hello, world!";

sort(s.begin(), s.end());
cout << s << endl;

sort Notes

• Elements of container must be comparable using "<" 
  – Depending on application, may be able to overload "<" for items to be sorted 
  – Otherwise, have to supply a separate bool valued function as a third 
    parameter to sort:

    bool rev(int a, int b) {
        return b < a;
    }

    int main() {
        Vector<int> foo = {16, 4, 23, 1, 2, 17, 6};
        sort(foo.begin(), foo.end());
        cout << foo << endl;
        sort(foo.begin(), foo.end(), rev);
        cout << foo << endl;
        return 0;
    }