

CIS 213: Foundations of Computer Science—Lab 3

Our main goal today is to understand more about how Selection Sort works. To do this, we will study some programs and make changes to them. Then we will use some Unix utilities to assist in graphing the results for easier analysis. The techniques in this lab will be useful for future labs and so ask questions :).

1 Selection Sort Timing

Copy all files from the lab directory (`~/hardnett/pub/cis213/lab2`). Make sure you have the following files:

- `ssort-rand.cc`
- `ssort-rand-vector.cc`

Browse the code. Try to determine how the code works, and then read the description below:

How is the program used? What does it do?

It creates an array of random numbers and sorts it. Optionally, it prints the sorted array of numbers. The number of numbers (the size of the array) is determined by a *command line argument*. Essentially, programs can have arguments (inputs) just the way functions do. In C and C++, this is done via an array called `argv` (argument vector) and the length of that array is `argc` (argument count). All of the things in the array are strings (characters), so if you want a number, you have to convert it using something like `atoi` (ASCII to integer). Do a “man” on `atoi` to learn more.

Compile and run it. Use the “time” command (see the “man” pages) to see how long it takes to sort an array of 5000 elements:

```
host% time ssort-rand 5000 n
```

Also try 6000, 7000, and so forth. How is the time affected?

2 Selection Sort Complexity

Add code to the program to count (1) the number of comparisons and (2) the number of swaps. Report this at the end of the run. Also report the predicted number of each given our analysis of the program.

3 Sorting by Keys

If the things we are sorting are objects with multiple parts, the sorting might be done only on one part, say the person’s name, but not their StudentID, or *vice versa*. The field or data member that we are sorting on is called the *key*. Your book has a box on page 32 talking about this. Make sure you have read it.

Change the program so that instead of sorting an array of integers, it sorts an array of students, where each is defined as follows:

```
class Student {
public:
    int studentID;
    char name[30];
    char grade;
}
```

How does this affect the running time?

4 Using the C++ STL

The C++ STL is the Standard Template Library. The STL contains a collection of generic data structures and algorithms for building programs. The data structures, also called containers, include:

list : a doubly linked list implementation

vector: a dynamic array with bounds checking

queue : a FIFO data structure based on list
 stack : a LIFO data structure based on list
 map : a hashing data structure

These are considered generic because they allow you to have collections of any data types. For example:

```
vector<int> IntArray; // declares an array of integers
list<Student*> StuList; // declares a linked list of Student ptrs
stack<char> MyStack; // declares a stack of characters
```

In this lab we will focus on the vector container. This container is meant to be a safer, more flexible, and easier to use dynamic array. It has member functions that automatically increase the size of the vector versus having to manage the allocations with mallocs and new operators.

Once a vector is defined, you have access to the following member functions:

Operation	Description	Example
resize(int)	resizes the vector to int size	IntArray.resize(25)
push_back(elem)	appends new element to vector and increases the size	IntArray.push_back(5)
elem pop_back()	removes element from the back	X = IntArray.pop_back()
size()	returns the current size of the vector	cout << IntArray.size()
[int]	access to element (overloaded operator)	X = IntArray[i]

Table 1: Vector Operations

The file *ssort-rand-vector.cc* contains an adapted version of the first program that sorts a list of integers. View the contents of the file, and make note of the changes to the original to see where vector support was added.

1. Compare the vector declaration in the new file with array declaration in the *ssort-rand.cc* file.
2. What does the new file use for allocation?
3. What happened to the parameter to the function *SSort(...)*?
4. Did anything change inside of the *SSort(...)* function?
5. Perform the timing test as you did for the original. Compare your results. What differences do you see? Can you explain the differences? (HINT: abstraction)

Turn in your finished program, using the `turnin213` script. Go to your lab2 directory:

```
host% turnin cis213 2
```