

CS303 (Spring 2008) - Assignment 2

Due: 01/30/2008

- (1) Here, you are asked to do what we did for Insertion Sort in class, except for Selection Sort. Informally, the way Selection Sort works is as follows: it first finds the smallest element of the array and puts it in the first place, then the second smallest element and puts it in the second place, and so forth. Here is the code to sort an array a with n elements:

```
for (i = 0; i < n-1; i ++)  
{  
    k = i;  
    for (j = i+1; j < n; j ++)  
        if (a[j] < a[k]) k = j;  
    swap (a[k], a[i]);  
}
```

- (a) Prove that Selection Sort correctly sorts the array. Think carefully about what exactly your induction hypothesis is, and how to formulate what the inner loop does.
- (b) Use a big- O type analysis to find the worst-case running time of Selection Sort, in $\Theta(\cdot)$ form. That is, you should have upper and lower bounds.
- (c) Use a big- O type analysis to find the worst-case number of `swap` operations the algorithm uses. Again, give a $\Theta(\cdot)$ form.
- (2) In class, we saw that the worst case running time of Insertion Sort is $\Theta(n^2)$. Above, you calculated that the worst case running time of Selection Sort is $\Theta(f(n))$ for some simple-looking function $f(n)$. In this problem, you should run some experiments to figure out what the hidden constants c_I, c_S for Insertion Sort and Selection Sort are (i.e., the c in the definition of O and Ω).

Implement both Selection Sort and Insertion Sort in C, C++, or Java (your choice), and make sure your code allows you to precisely measure the running time. Figure out what the worst-case input (or pretty bad inputs) for the algorithms looks like, and run the algorithms on worst-case arrays of at least 6 different sizes n . Experiment to make sure you pick the array sizes from a range where you can measure the running times reasonably, i.e., the running times are between 0.1 seconds and one minute or so (if the running time is less than 0.001 seconds, the program will almost certainly output 0 seconds instead).

Plot the running times for both algorithms (say, in micro-seconds as a function of n), and find out the constants c_I, c_S by fitting $c_I n^2$ and $c_S f(n)$ against your running time curves. Also plot those curves to see how well they fit.

Submit (1) a printout of your source code, (2) your plots, (3) the values of the constants with brief explanation (including what type of machine you ran it on, and perhaps other information pertinent to estimating running times).