COMP132 – Computer Science II
Fall 2015

Homework #9
Sorting and Searching
Solutions

1. In this problem you will investigate the running time and asymptotic class of several different implementations of the Selection Sort algorithm.

a. Run the `SelectionSortExperiment` program contained in the `comp132.examples.sorting` package of the `132SampleCode` project. Perform the following tasks based on the results reported:

i. Give a formula describing the running time of the selection sort implementation in the `iterativeSort` method as a function of the problem size (n). To get this formula you can use the process that we used in class:

- Copy and paste the results into Excel.
- Create an x-y scatter plot of the data.
- Add a polynomial trend line to the graph with its equation.

$f(n) = 2x10^{-9}n^2 - 4x10^{-7}n + 0.0013$

Note: Since you are running the program on a different computer that may be faster or slower than mine, you will likely get different constants before the n^2, n and at the end.

ii. Express the running time from part i using Big-O notation.

$O(n^2)$

Note: Regardless of the constants you got in part i, this answer should be $O(n^2)$.

b. Repeat part a using the recursive implementation of selection sort that is provided in the `recursiveSort` method of the `SelectionSort` class.

i.

$f(n) = 3x10^{-9}n^2 – 6x10^{-8}n - 0.0004$

Note: Since you are running the program on a different computer that may be faster or slower than mine, you will likely get different constants before the n^2, n and at the end.

ii.

$O(n^2)$

Note: Regardless of the constants you got in part i, this answer should be $O(n^2)$.

c. Briefly compare the running times and the asymptotic classes of the two distinct implementations of selection sort. Which sort is faster? Do they belong to the same asymptotic class?

The constant on n2 is larger on the recursive sort than it is on the iterative sort. Thus, as n get larger the function f(n) for the recursive sort will be larger than for the iterative sort. Thus, the iterative implementation is faster than the recursive implementation.

However, both the iterative and recursive implementations both belong to the $O(n^2)$ asymptotic class.

Note that the functions f(n) were different on your machine, my machine and the classroom machine as well as for the iterative and recursive implementations. But in all cases the resulting function was a polynomial of order 2 and thus was $O(n^2)$. This should help to illustrate that the asymptotic class is really a property of the algorithm rather than the particular implementation.

2. True or False: An implementation of a O(n) algorithm will always run faster than an implementation of a $O(n^2)$ algorithm, regardless of the problem size. Briefly but fully explain your answer.

False: For small problem sizes (i.e. small values of n) it is possible that the implementation of the $O(n^2)$ algorithm will run more quickly than the O(n) algorithm implementation. This could be because the constants in the f(n) for the O(n) algorithm might be much larger than those in the f(n) for the $O(n^2)$ algorithm. For example:

$f_1(n) = n^2 + 2$ -> $O(n^2)$
$f_2(n) = 10n + 2$ -> O(n)

In this case, as long as $n < 10$, the function $f_2(n)$ is larger than $f_1(n)$ and thus this implementation of the O(n) algorithm is slower than the $O(n^2)$ algorithm. However, once n is sufficiently large ($n > 10$) the O(n) algorithm implementation will be faster.

3. For each of the following running times give the name of the asymptotic class to which the algorithm belongs (e.g. constant, linear, etc…) and express the running time using Big-O notation.

a. $3n^2-2n+273$ **$O(n^2)$**
b. $14n + 22\ n \lg n + 1$ **$O(n \lg n)$**
c. $0.0001n + 100^{123}$ **$O(n)$**
d. $(n - 4)(2n - 5)$ **$O(n^2)$**
e. $123456 + 10 \lg n$ **$O(\lg n)$**
f. $2^{32}-1$ **$O(1)$**

4. Imagine a program that processes 5000 input values in 10 seconds. How long would it take the program to process 20,000 input values if the algorithm implemented by the program is:

a. A O(n) algorithm.

When the input to a O(n) algorithm doubles in size the running time approximately doubles in length. Thus, because the input doubled in size twice, from 5000 to 10,000 to 20,000 the running time would also double twice, from 10 to 20 to 40. So the running time for n=20,000 would be approximately 40 seconds.

b. A $O(n^2)$ algorithm.

When the input to a $O(n^2)$ algorithm doubles in size the running time approximately quadruples. So the running time for n=20,000 would be approximately 160 seconds.

5. Consider the following program that swaps the maximum and minimum values in an array.

```
public void swapMaxAndMin(int[] vals) {
  int mindex = 0;
  int maxdex = 0;

  for (int i=1; i<vals.length; i++) {
    if (vals[i] < vals[mindex]) {
      mindex = i;
    }
    if (vals[i] > vals[maxdex]) {
      maxdex = i;
    }
  }

  int tmp = vals[mindex];
  vals[mindex] = vals[maxdex];
  vals[maxdex] = tmp;
}
```

a. Give a function describing the operation count for this program if an array access is counted as the basic operation.

		i=1	i=2	i=3	...	i=n-1	After loop
f(n)	=	4 +	4 +	4 +	...	4 +	4

= 4 * (n-1) + 4
= 4n – 4 + 4
= 4n

b. Express the function from part a using Big-O notation.

O(n)

6. We know that once the problem becomes large enough a O(n) algorithm will be faster than a $O(n^2)$ algorithm. Consider the following running time equations for programs implementing two different algorithms for the problem:

A: $f(n) = 2n^2 - 3n + 70$
B: $f(n) = 25n + 100$

How large would the problem need to be in order for program B to be faster than program A? Show your work and explain how you found your solution.

To solve this problem we need to determine the value of n for which the running time of program A is equal to the running time of program B. Then, for all problem sizes larger than that n, program B will be faster than program A. So setting the equations equal and solving gives:

$2n^2 - 3n + 70 = 25n + 100$
$2n^2 - 3n - 25n + 70 - 100 = 0$
$2n^2 - 28n - 30 = 0$

This quadratic has two roots: 15 and -1. A negative problem size makes no sense so 15 is the root of interest.

So, for all $n > 15$ program B will be faster than program A.