Algorithms
Homework 1

Saad Mneimneh
Computer Science
Hunter College and Graduate Center of CUNY

Problem 1: Selection Sort

(a) Write a pseudocode for Selection sort, which in iteration i, finds the index k of the smallest element in $A[i \dots n]$, and swaps $A[i]$ and $A[k]$. Make sure you have two nested loops as in the case for Insertion Sort illustrated in class.

(b) Annotate your pseudocode with the number of times each line is executed, and figure out the running time as a function of n, the length of the array. Confirm that whether the array is sorted or not, the running time of Selection Sort remains quadratic in n.

Note: This is essentially similar to the exercise we did in class for Insertion Sort. It's a little tedious in nature, but if you approach it systematically, it is not too hard to handle.

Problem 2: Another look at Mergesort

For simplicity, we can think of n and k as powers of 2 in this problem, where $n \geq k$.

Assume there is a "magical" data structure that can support the following operations:

- insert an element
- extract the smallest element

Assume further that each of the operations above takes $O(1)$ time (this means constant time), independent of the number of elements in the structure. However, the structure cannot grow beyond k elements, for some constant k.

We are also given an array of n elements that we would like to sort.

(a) Revisit the base case of Mergesort, which is now encountered when the size of a sub-problem is k. Describe how you can benefit from the above data structure to obtain a running time of $\Theta(n \log_2 \frac{n}{k})$ for Mergesort. Illustrate with a recursion tree argument.

(b) Consider a modified version of Mergesort in which we divide a problem of size n into k sub-problems of size n/k each (as opposed to simply 2 sub-problems of size $n/2$ each).

- Describe how we can perform the merging of k sorted sub-problems in $\Theta(n)$ time using the "magical" data structure (in clear English no need for a pseudocode).
- Write the pseudocode for this version of Mergesort (use the merge function on k arrays as a subroutine, no need to actually spell it out in pseudocode). Again, the base case is when the size becomes k.
- Illustrate the recursive tree structure and find the running time as a function of n and k.