# CS360 - Assignment 1

1.2-2 Note that $lg()$ represents $log_2()$

$$\begin{aligned} 8n^2 &= 64nlg(n) \\ \Rightarrow n &= 8lg(n) \end{aligned}$$

This equation cannot be solved analytically, instead we will tablulate values to approximate the answer:

$$\begin{array}{cc} n = 16 & 8lg(16) = 32 \\ n = 32 & 8lg(32) = 40 \\ n = 64 & 8lg(64) = 48 \\ n = 40 & 8lg(40) = 42.6 \\ n = 44 & 8lg(44) = 43.7 \end{array}$$

Hence for $n < 44$ insertion sort is faster than merge sort.

1.2-3 Again the equation $2^n = 100n^2$ cannot be solved analytically so tabulate values:

$$\begin{array}{cc} n = 10 & 2^{10} = 1024 < (100)(10)^2 = 10000 \\ n = 20 & 2^{20} = 1048576 > (100)(20)^2 = 40000 \\ n = 15 & 2^{15} = 32768 > (100)(15)^2 = 22500 \\ n = 14 & 2^{14} = 16384 < (100)(14)^2 = 19600 \end{array}$$

Hence for $n \leq 14$ the exponential algorithm will be faster.

2.2-3 On average, half of the elements will have to be checked (assuming the element being searched for is equally likely to be any element in the array). This can be shown since each element will have $1/n$ probability of being the desired element and for the $i^{th}$ element will have had to have searched $i$ elements. Thus the average over all the elements can be written as

$$\begin{aligned} \frac{1}{n}\sum_{i=1}^{n} i &= \frac{1}{n}\left(\frac{n(n+1)}{2}\right) \\ &= \frac{n+1}{2} \end{aligned}$$

$\Rightarrow T(n) \approx c\frac{n+1}{2} = \Theta(n)$ on average.

The worst case is when the element is *not in* the array resulting in every element being checked.
$\Rightarrow T(n) \approx c(n+1) = \Theta(n)$ worst case (since we must also check that there are no more elements).

In both the average and worst case, the running time of linear search will grow linearly, i.e. $\Theta(n)$.

2-2 (a) For correctness we need to show not only that the algorithm terminates with $A'[1] \leq A'[2] \leq A'[3] \leq ... \leq A'[n]$, but additionally that the elements of $A'$ are simply a permutation of he elements of $A$, i.e. that $A'$ contains *all* the elements of $A$.

(b) The loop invariant for the loop 2-4 is that $A[j]$ is the smallest element in $A[j...n]$ and that $A[j...n]$ is a permutation of those elements.
*Initialization* - For $j = n$, $A[j...n] = A[n]$ is a single element which trivially satisfies the loop invariant.
*Maintenance* - By the loop invariant, at any iteration $A[j]$ is the smallest element in $A[j...n]$, so the only action that can occur within the loop is $A[j-1]$ is exhanged with $A[j]$ if $A[j] < A[j-1]$ (line 4). Hence $A[j-1]$ will be the smallest element in $A[j-1...n]$ and since $A[j...n]$ was a permutation, $A[j-1...n]$ must also be. Therefore after the iteration $A[j-1]$ is the smallest element of $A[j-1...n]$ and $A[j-1...n]$ is a permutation.
*Termination* - The loop terminates when $j = i$ which by the loop invariant gives $A[i]$ is the smallest element of $A[i...n]$ and $A[i...n]$ is a permutation.

(c) The loop invariant for the outer loop is that $A[1...i]$ are the smallest elements from $A[1...n]$ in sorted order with $A[i+1...n]$ containing the remaining elements.
*Initialization* - For $i = 1$ $A[1...i] = A[1]$ with only a single element so it trivially satisfies the loop invariant.
*Maintenance* - For a given iteration $i$, $A[1...i]$ are the smallest (sorted) elements. From part (b), after the inner loop executes, $A[i+1]$ is the smallet element from $A[i+1...n]$ and $A[i+1...n]$ is a permutation of those elements. Therefore $A[1...i+1]$ contains the smallest (sorted) elements and $A[i+2...n]$ is a permutation of the remaining elements.
*Termination* - The loop terminates when $i = n$ which by the loop invariant states $A[1...i] = A[1...n]$ are sorted (and that $A[i+1...n] = \emptyset$ are the remaining elements). Therefore, the entire array is sorted.

(d) For this version of bubble sort (fixed iteration loops), the worst case will swap at every iteration of the inner loop. Hence the outer loop executes $n$ times ($i \to 1...n$) while the inner loop executes $n - i$ times ($j \to n...i + 1$) giving

$$\begin{aligned} T(n) &= \sum_{i=1}^{n}(n-i) = \sum_{i=1}^{n} n - \sum_{i=1}^{n} i = n^2 - \frac{n(n+1)}{2} \\ &= n^2 - \frac{n^2}{2} - \frac{n}{2} = \frac{n^2}{2} - \frac{n}{2} = \Theta(n^2) \end{aligned}$$

Therefore worst case running time is asymptotically the same as insertion sort - $\Theta(n^2)$.