CPS 130 Homework 2 - Solutions

1. (CLRS 2.2-4) How can we modify almost any algorithm to have a good best-case running time?

   Solution: Hardcode the solution for some particular input, and check whether the input is that one. If so, simply output the hardcoded solution. One could also check whether the problem is already solved (when sorting, for instance, check first if input is already sorted).

2. (CLRS 3.1-3) Explain why the statement 'The running time of algorithm A is at least $O(n^2)$' is content free.

   Solution: This just checks that you understand what an upper bound is. The statement is saying "the running time is at least at most n^2", but says nothing about the upper bound, which could be $O(n^3)$, $O(2^n)$, etc. We cannot extract any information regarding lower bounds either, so the statement is meaningless and content free.

3. CLRS 2-4 (The inversion problem).

   Solution:

   a.

$(\mathbf{i}, \mathbf{j})$	$\mathbf{A[i] > A[j]}$
$(1,5)$	$2 > 1$
$(2,5)$	$3 > 1$
$(3,5)$	$8 > 1$
$(4,5)$	$6 > 1$
$(3,4)$	$8 > 6$

   b. The reverse-ordered list $\langle n, n-1, \ldots, 2, 1\rangle$ has the most inversions. The number of inversions is

   $$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}.$$

   c. Let $f(n)$ be the number of inversions of array $A[1..n]$. The running time of insertion sort given A is $\Omega(f(n))$. Referring to page 8 of CLR, we can express the run time of insertion sort as

   $$T(n) = c_\alpha \sum_{j=2}^{n} (t_j - 1) + c_\beta n + c_\gamma.$$

   The number of times the contents of the while loop executes for index j is $(t_j - 1)$. The loop only executes if $0 < i < j$ and $A[i] > A[j]$, which means (i,j) is an inversion. The loop removes only one inversion at a time, so in $(t_j - 1)$ iterations, all inversions for element $A[j]$ have been removed. The total number of inversions is

   $$f(n) = \sum_{j=2}^{n} (t_j - 1).$$

The total runtime can now be expressed as

$$T(n) = c_\alpha f(n) + c_\beta n + c_\gamma > c_\alpha f(n) = \Omega(f(n)).$$

Therefore, the running time of insertion sort is lower bounded by the number of inversions.

d. Execute merge sort on the array A. Assume sorting is done from left to right. MERGE(A, p, q, r) merges arrays $L = A[p..q]$ and $R = A[q+1..r]$. Since L and R are sorted, there are no inversions in L or R, only inversions between L and R. Refer to page 12 CLR for the outline of MERGE. In merging L and R, we select the smallest element from the top of stacks L and R to put into our sorted list. To count inversions, every time we select an element from list R, we increment a counter by the number of unselected elements in list L. The running time is identical to merge sort or $\Theta(n \lg n)$. To show correctness, we must show that every inversion is counted exactly once. Since L and R are sorted, there are no inversions in L or R, only inversions between L and R. If an element r from list R is selected ahead of k items in list L, then r is less than the remaining k elements $l_1, l_2, \ldots, l_k$ in list L, corresponding to k inversions. Selecting r to be placed next in the sorted merged list removes all k inversions. No new inversions are created in the merging step, since the resulting array $A[p..r]$ is sorted and therefore has no inversions. No inversion can be counted twice, since removing an inversion sorts L and R into one combined list, and inversions can only exist between lists. Our new modified merge routine counts all inversions correctly in $\Theta(n \lg n)$ time.

4. (part of CLRS 3-3) Order the following expressions by their asymptotic growth and **justify your answer**.

$$2^n, n!, (\log n)!, n^3, e^n, 2^{\log_2 n}, n \log n, 2^{2^n}, n^{\log \log n}.$$

Solution: $2^{\log_2 n} < n \log n < n^3 < (\log n)! < n^{\log \log n} < 2^n < e^n < n! < 2^{2^n}$.

- One typical way to compare the growth of two functions is to compute their limit when n goes to ∞, that is, $\lim_{n\to\infty} \frac{f(n)}{g(n)}$. If the limit is finite, then f and g have the same order of growth ($f = \Theta(g)$). If the limit is ∞ then $f > g$ ($f = w(g)$). If the limit is zero, than $f < g$ ($f = o(g)$).
- What you should remember is that any (positive) polynomial growth faster than any logarithm, and any (positive) exponential growth faster than any polynomial, that is:
$$\log^a < n^b < c^n, \text{ for any } a, b > 0 \text{ and any } c > 1$$
- One other trick you can use is to take the logarithm. For instance, $n^{\lg \lg n} = (\lg n)^{\lg n} < 2^n$ because $\lg n \lg \lg n < n$.
- Typically you can use the Stirling formula to prove bounds involving $n!$:
$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \leq n! \leq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{1/12n}$$
so that comparing $n!$ and 2^{2^n} is the same as taking the logarithm of both and comparing $n \lg n$ and 2^n. The first is smaller than the second, since $n \lg n$ is polynomially bounded and 2^n is exponential.