Solution to HW 5

March 10, 2013

1. (a) Given 2n numbers, we can choose n numbers for the first list and put the other n numbers in the second list. Since there is no question of sorting we have $\binom{2n}{n}$ possible ways to divide 2n numbers into two sorted lists, each with n numbers.

   (b) The number of comparisions required for merging two sorted arrays is atleast $log\binom{2n}{n}$
   Using Stirlings approximation:
   $lg(\binom{2n}{n}) = lg(\frac{(2n)!}{n!n!})$
   $\implies lg\frac{\sqrt{2\pi 2n}(\frac{2n}{e})^{2n}e^{\alpha_{2n}}}{(\sqrt{2\pi n}(\frac{n}{e})^n e^{\alpha_n})^2}$ (where $\frac{1}{l2n+1} < \alpha_n < \frac{1}{l2n}$ and $\frac{1}{l2(2n)+1} < \alpha_{2n} < \frac{1}{l2(2n)}$)
   $\implies lg(\sqrt{2\pi 2n}(\frac{2n}{e})^{2n}e^{\alpha_{2n})-2lg((\sqrt{2\pi n}(\frac{n}{e})^n e^{\alpha_n})}$
   $\implies 2n - \frac{1}{2}lgn + [(\alpha_{2n} - 2\alpha_n)lge - \frac{1}{2}lg\pi]$
   $\implies 2n - \frac{1}{2}lgn + d$ (where $d = [(\alpha_{2n} - 2\alpha_n)lge - \frac{1}{2}lg\pi]$)
   Since $\frac{1}{2}lgn + d$ is $o(n)$, the number of comparisions needed is atleast $2n - o(n)$.

   (c) Suppose the original arrays are $(a_1, a_2, ...a_n)$ and $(b_1, b_2, ...b_n)$ and that a_i and b_j are in sorted order. Without loss of generality lets assume that $a_i < b_j$. Suppose no comparision is made between a_i and b_j. Since comparingb_j to any element other than a_i is the same as comparing it to a_i (As a_i and b_j are consecutive.)¿ we end up following the same path through the decision tree and reach the same node. If we run a merging algorithm on the arrays we get an incorrectly sorted array. Since this is a contradiction an comparision is maded between a_i and b_j.

   (d) Any two consecutive entries in the sorted array come from opposite arrays. Since there are $2n-1$ such such pairs atleast $2n-1$ comparisions must be made.

2. When the input array is already sorted in increasing order, HEAPSORT takes $\Omega(nlgn)$ time since each of the $n-1$ calls to MAX-HEAPIFY takes $\Omega(lgn)$time.

3. IN radix sort we start with the least significant digits first. So at the i^{th} iteration of radix sort, the numbers are sorted with respect to the i^{th} least significant digits.
   So, we can output any number as soon as we consider all its digits we can put it in the right place in the sorted array.
   k is the total number of digits on all numbers.
   n is the size of the list.
   To output tha numbers whose$length = i$ after i^{th} iteration, we need to check the length of each number before putting it in a bucket at round$i+1$.
   If numberlenth is less than i then move the number to the result list.
   So the time complexity is $O(n + k)$

4. (a) Compare each red jug with each blue jug. Since there are n red jugs and$n$ blue jugs, that will take $\theta(n^2)$ comparisons in the worst case.

   (b) We can view the computation of the algorithm in terms of a decision tree. Every internal node is labeled with two jugs (one red, one blue) which we compare, and has three outgoing edges (red jug smaller, same size, or larger than the blue jug). The leaves are labeled with a unique matching of jugs. The height$h$ of the decision tree is equal to the worst-case number of comparisons the algorithm has to make to determine the matching. And there are $n!$ different matchings. So, $3^h \geq n! \geq (n/e)^n, h = \Omega(nlgn)$. So any algorithm solving the problem must use $\Omega(nlgn)$comparisons.

   (c)
```
MATCH-JUGS(R, B)
        if |R| = 0
        return
        if |R| = 1
        let R = {r} and B = {b}
        output (r, b)
        return
        else r  a randomly chosen jug in R
        compare r to every jug of B
        B1  the set of jugs in B that are smaller than r
        B2  the set of jugs in B that are larger than r
        b  the one jug in B with the same size as r
        compare b to every jug of R  {r}
        R1  the set of jugs in R that are smaller than b
        R2  the set of jugs in R that are larger than b
        output (r, b)
        MATCH-JUGS(R1, B1)
        MATCH-JUGS(R2, B2)
```

   This algorithm is similar to quicksort, and we arrive at the solution of $O(nlgn)$ comparisons. The worst case is $T(n) = \theta(n^2)$.