CSL 356 Solutions to Tutorial Sheet 2

1. Given a binary tree of N leaf nodes, show that the average distance from leaf nodes to the root node is $\Omega(\log N)$ for *any* binary tree.
   Hint: Suppose the minimum tree has x nodes in the left subtree and $N - x$ in the right subtree - minimize average path length wrt x.

   Sol
   Claim: The balanced tree attain minimum path length. Proof by induction,

   Base case: true for 2 nodes
   Suppose it holds for all $i \leq N$, then for N,
   let x and $N - x$ be the optimum
   By IH, $\log x$ and $\log(N - x)$ must be the heights of the left and right subtree. Total path length
   $P_x = x(\log x + 1) + (N - x)(\log(N - x) + 1)$
   $\frac{d}{dx} P_x = 0 \implies (\log x + 1) + 1 - \log(N - x) + -1 + (N - x)\frac{1}{N-x}(-1) = 0$
   $\implies \log x = \log(N - x) \implies x = N/2$

   i.e balance tree attains minimum

2. If we are given a number x, let N_x denote the number of elements in a heap $\leq x$. Show how to output all elements $\leq x$ in $O(N_x)$ steps in a heap.
   In a binary min-heap of n elements, prove that the smallest k elements are present in a subtree rooted at the root (i.e. these elements are not arbitrarily scattered in the heap).
   Sol
   Algorithm: if the value of a node N_v is less than x then report N_v and search both left and right children, otherwise don't search below N_v.

   Claim: Every node that is visited is either output or it's parent is output. So the size of the output is at least 1/3 (number of nodes visited).

   This is sometime called filtering search.

3. For n distinct elements $x_1, x_2 \ldots x_n$ with positive weights $w_1, w_2 \ldots w_n$ such that $\sum_i w_i = 1$, the *weighted median* is the element x_k satisfying

   $$\sum_{i|x_i<x_k} w_i \leq 1/2 \; \sum_{i|x_i \geq x_k, i \neq k} w_i \leq 1/2$$

   Describe an $O(n)$ algorithm to find such an element. Note that if $w_i = 1/n$ then x_k is the (ordinary) median.

   Sol
   Choose the unweighted median M in $O(n)$ time and determine by adding the weights of the elements less than M, which portion, the weighted median lies ? Accordingly search recursively in that portion $T(n) \leq T(n/2) + O(n) \implies T(n) = O(n)$.

4. Given two sorted arrays A and B of sizes m and n respectively, design an algorithm to find the median in $O(polylog(m + n))$.
   (You can do this in exactly $O(\log(m + n))$ steps).
   Can you generalize it to m sorted arrays ?

   Sol
   Let the arrays be of size m and n (wlog $m \leq n$). Denote the median of A and B by m_A and m_B

respectively.

Case 1: $m_A < m_B$
Then all elements smaller than m_A has at least $n/2+m/2$ elements larger.i.e rank is strictly less than $n/2+m/2$ as well as all the elements in A smaller than m_A. Similarly m_B is larger than $n/2+m/2$ elements(and elements in B are larger than m_B).

Case 1: $m_A \geq m_B$
Elements in B less than m_B and elements in A larger than m_A need not be considered. So in both cases we can eliminate $\min\{n/2, m/2\}$ elements less than (greater than) the combined median. i.e one of the arrays is halved. This implies $O(\log m + \log n)$ bound which is $O(\log(n+m))$.

5. Use a divide and conquer based approach to find the maximum and minimum element among a set of n numbers in $3n/2$ comparisons.
Can you do any better ?

6. * We want to sort n integers in the range $0..2^{b-1}$ (b bits each) using the following approach. Let us assume that b is a power of 2. We divide each integer into two $b/2$ bit numbers - say x_i has two parts x'_i and $x"_i$ where x'_i is the more significant part. We view the more significant bits as buckets and create lists of $b/2$ bit numbers by associating the lower significant $b/2$ bit numbers with the bucket with the more significant bits. Namely $x"_i$ is put into the list corresponding to x'_i. To merge the list we now add the $b/2$ bit numbers corresponding to the non-empty buckets to the earlier list (to distinguish, we can mark them). We can now sort the list of $b/2$ bit integers recursively and output the merged list by scanning the sorted elements. Note that this list can have more than n numbers since we added the buckets also. Suggest a method to avoid this blow up (since it is not good for recursion) and analyze this algorithm.

Sol
Let us assume that b is a power of 2. We divide each integer into two $b/2$ bit numbers - say x_i has two parts x'_i and $x"_i$ where x'_i is the more significant part. We would like to implement a MSB first radix sort by sorting both the x'_is and x''_is "together". Suppose we have sorted them - how do we reconstruct the final sorted list. Assume that we maintain information about the MSB/LSB and also the induces i with each x'_is and x''_i. From the sorted list we bucket the x''_is (so we are not looking at information on the sorted x'_is(the non-empty bucket). The obvious problem is that in the recursive call we must sort $2n$ $b/2$ numbers. To prevent blow up in the problem size, from each bucket we remove the min element, say b_i^m before making the recursive call. Therefore we have removed number of elements equal to number of buckets, that prevents the blow up
$T(n,b) \leq T(n,b/2) + O(n)$
$\implies T(n,b) = O(n \log b)$ $T(n, \log n)$ is $O(n)$ for base case.

7. Consider a job scheduling problem where each job J_i has a start and a finish time (s_i, f_i). Two jobs cannot run simultaneously and once started, a job must run to its completion (i.e. we cannot split a job into parts). Given a set of jobs
(i) If we schedule greedily in increasing order of finish times can we maximize the number of jobs completed ? Justify.
(ii) If job J_i is associated with a profit p_i (≥ 0), can you apply a greedy algorithm to maximize the profit (of all completed jobs) ? Justify.

Sol
Consider jobs $J_1 = (1,10), J_2 = (1,4), J_3 = (3,12)$. This is not a matroid max and sets have different sizes.

Consider the optimal sequence. If we substitute the first job by the smallest finishing time, then the number of jobs completed cannot change. So there is a solution with a smallest finish time. Suppose there is a solution inductive with the first $i(\geq 1)$ jobs being greedy chooses on the finishing times. Then from the above reasoning we cab pick the earliest finishing time without decreasing the number of jobs completed.

8. We are given a set of events (with starting and finishing times) that have to be scheduled in a number of halls without conflicts. Design an algorithm to find the minimum number of halls needed for this purpose. Note that the timings are fixed and no two events can happen at the same time in the same hall.
   You can think about the events as intervals on the real line such that that we have to assign a color to each interval in a way that no two overlapping intervals are assigned the same color. What is the minimum number of colors required ?

   Sol

   Consider the set of n intervals corresponding to start and finish times. If we scan from left to right, at any instant t, the number of intervals intersection the vertical line at t is the maximum number of halls necessary. Let us assign colours greedily to the intervals. If there are n_t intervals at time t we will be using t colours. If an interval ends, a clour becomes available. If an interval begins, we will need a different colour from the one being currently active. We can argue easily that we will never need more than n_{max} colors where n_{max} is the maximum number of active intervals over any time instant.

   Show how to implement the algorithm in $O(n \log n)$ steps.

9. The second minimal spanning tree is one that is distinct from the minimal spanning tree (has to differ by at least one edge) and is an MST if the original tree is ignored (they may even have the same weight). Design an efficient algorithm to determine the second MST.

   Sol Consider a second MST T' which differs from some MST T by the least number of edges, i.e. $T \oplus T'$ is minimum. Suppose the edges in increasing order of weights in T are

   $e_1, e_2, \ldots e_i \ldots e_{n-1}$ and for T' these are

   $e'_1, e'_2 \ldots e'_i \ldots e'_{n-1}$. Let i be the first index where they differ. Clearly $w(e'_i) \geq w(e_i)$ otherwise we can improve the MST by introducing e'_i in T that induces a cycle with edges that have larger weights in T.

   By introducing e_i in T',it induces a cycle C in T' , in which some edge, say $e" \notin T$. By removing $e"$ either T' strictly improves (to MST if $w(e") > w(e_i)$) or if $w(e") = w(e_i)$, then the resulting $T" = T' \oplus e_i - e"$ is *closer* to T which is a contradiction.