

CS 3500 Section A, Spring 2000

Final Exam Solutions

Problem 1 Arrange the following 6 functions in increasing O order: $4n + n^{\log n}$, $\log n + n^{0.02}$, n^3 , 2^n , $n!$, $20n^2$.

Solution :

$$\begin{array}{c} \log n + n^{0.02} \\ 20n^2 \\ n^3 \\ 4n + n^{\log n} \\ 2^n \\ n! \end{array}$$

Notice that $n^{\log n}$ is larger than any constant power of n , so $4n + n^{\log n} = \Theta(n^{\log n})$ should come after n^3 . To see that $4n + n^{\log n}$ comes before 2^n , use the identity $n = 2^{\log n}$ to get that

$$n^{\log n} = (2^{\log n})^{\log n} = 2^{(\log n)^2} < 2^n$$

for large enough n . □

Problem 2 Use the master theorem to find the Θ solution to the recurrence

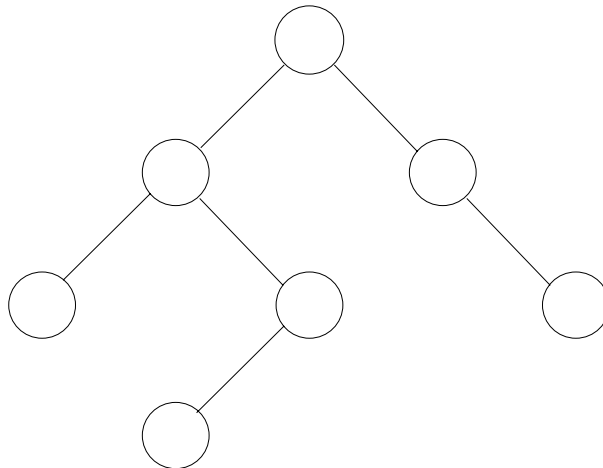
$$T(n) = 2T(n/4) + 3\sqrt{n} + \log^2 n$$

Solution : Using the notations in the master theorem, we have $a = 2$, $b = 4$, and $f(n) = 3\sqrt{n} + \log^2 n = \Theta(n^{1/2})$. Since $\log_4 2 = 1/2$, $f(n) = \Theta(n^{\log_4 2})$. Hence, by Case 2 of the theorem,

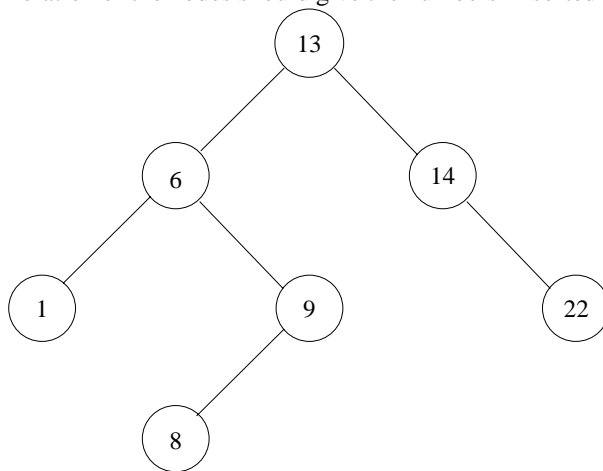
$$T(n) = \Theta(n^{1/2} \log n) = \Theta(\sqrt{n} \log n)$$

□

Problem 3 Label the following binary tree with numbers from the set $\{6, 22, 9, 14, 13, 1, 8\}$ so that it is a legal *binary search tree*.



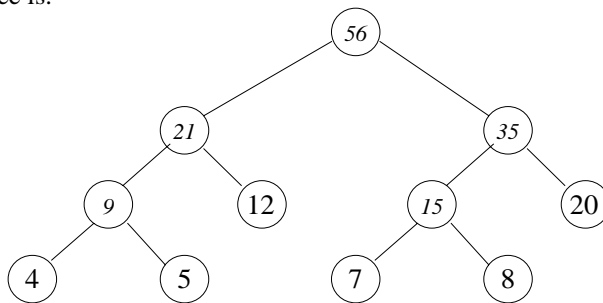
Solution : The inorder enumeration of the nodes should give the numbers in sorted order:



□

Problem 4 Construct the Huffman tree corresponding to letter frequencies of 4,5,7,8,12, and 20.

Solution : The Huffman tree is:



□

Problem 5 Describe an $O(n \log n)$ time algorithm to test whether a sequence a_1, \dots, a_n of integers contains any duplicates.

Solution : Sort the sequence using an $O(n \log n)$ algorithms, such as heapsort or merge sort. Then traverse the sorted sequence, checking whether any two consecutive elements are equal. Since this last step adds only $O(n)$ to the running time, the whole algorithm runs in $O(n \log n)$. □

Problem 6 Describe an $O(n \log n)$ time algorithm that, for given integers x and a_1, \dots, a_n , determines whether or not there exist two a_i 's whose difference is exactly x .

Solution : One possibility is to sort the array in $O(n \log n)$ time, then binary search for each $a_i + x$ in the sorted array (this takes $O(n \log n)$ more time, since each binary search can be performed in $O(\log n)$ steps).

A more elegant solution is to reduce this problem to the previous one. Construct the sequence

$$a_1, a_2, \dots, a_n, a_1 + x, a_2 + x, \dots, a_n + x$$

after eliminating duplicates in a_1, \dots, a_n , if any, using the $O(n \log n)$ time algorithm given for the previous problem. Now test again for duplicates in the newly constructed sequence. If any duplicate is found, it can only be produced by a pair of indices for which $a_i = a_j + x$, which means that $a_i - a_j = x$. The running time of this last step is $O(2n \log(2n)) = O(n \log n)$, so this algorithm still takes $O(n \log n)$ time overall. □

Problem 7 You are given a sequence of positive integers a_1, a_2, \dots, a_n , and a positive integer B . Your goal is to determine if some subsequence of a_1, a_2, \dots, a_n sums to exactly B . In other words, determine if there is a subset S of (not necessarily consecutive) indices so that $\sum_{i \in S} a_i = B$. Give an $O(nB)$ dynamic programming algorithm for this problem.

Solution : For every $i \in \{1, \dots, n\}$ and $b \in \{1, \dots, B\}$, let

$$exists(i, b) = \begin{cases} TRUE, & \text{there exists a subsequence of } a_1, \dots, a_i \text{ summing up to exactly } b \\ FALSE, & \text{otherwise} \end{cases}$$

Clearly, $exists(1, b) = TRUE$ if $b = 0$ or $b = a_1$, and $exists(1, b) = FALSE$ for all other b 's. Also,

$$exists(i + 1, b) = exists(i, b) \text{ OR } exists(i, b - a_{i+1})$$

Using this recurrence, we can compute all values $exists(i, b)$ in $O(nB)$. The answer to the original question is given by $exists(n, B)$. \square

Problem 8 For a graph with n vertices and m edges, Prim's MST algorithm runs in $O(m \log n)$ if Q is implemented as a binary heap, and in $O(n^2)$ time if Q is implemented as an unsorted array. What relation must exist between m and n (i.e., how dense must the graph be) for the array implementation to be asymptotically faster than the binary heap implementation?

Solution : The array implementation is at least as fast as the heap implementation if $m \log n = \Omega(n^2)$, or $m = \Omega\left(\frac{n^2}{\log n}\right)$. \square

Problem 9 Describe an $O(m)$ algorithm for computing single-source shortest paths in a connected undirected graph with edges of weight 1 or 2.

Solution : It is possible to give a modification of Dijkstra's algorithm that runs in $O(m)$ by using a "bucket" implementation of the priority queue. It is also possible to modify breadth-first search to handle graphs with edge-weights of 1 or 2. The following solution uses an unmodified breadth-first algorithm, by converting the shortest-paths problem in a graph with edge-weights 1 or 2 to a shortest-paths problem in a graph with unit edge-weights.

Suppose G is a graph with n vertices and m edges of weight 1 or 2. Construct a new graph G' obtained from G by *subdividing* each edge of weight 2, i.e., by adding a new vertex v_e for each such edge e , deleting e , then adding two edges of weight 1 connecting v_e to the two ends of e . Clearly, this transformation does not change the weight of the shortest paths. Since G' has only edges of weight 1, the shortest paths can now be computed by using BFS. The running time of this BFS is $O(n' + m')$, where n' and m' are the number of vertices and edges in G' . Since $n' = O(n + m)$, $m' = O(m)$, and $n = O(m)$, it follows that the overall running time is $O(m)$. \square

Problem 10 Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* if there exists a bijection $f : V_1 \rightarrow V_2$ such that $(u, v) \in E_1$ iff $(f(u), f(v)) \in E_2$. A graph $G_2 = (V_2, E_2)$ is a *subgraph* of a graph $G_3 = (V_3, E_3)$ if $V_2 \subseteq V_3$ and $E_2 \subseteq E_3$. Let SUBGRAPH_ISO be the problem of deciding whether a graph G_1 is isomorphic to a subgraph of a graph G_3 , i.e.

$$\text{SUBGRAPH_ISO} = \{ \langle G_1, G_3 \rangle \mid G_1 \text{ is isomorphic to a subgraph of } G_3 \}$$

Notice that SUBGRAPH_ISO is in the class NP, since it is easy to verify that G_1 is isomorphic to a subgraph of G_3 if the subgraph and the isomorphism mapping are given. Complete the proof that SUBGRAPH_ISO is NP-complete.

Solution : We will prove that CLIQUE reduces to SUBGRAPH_ISO. Let $\langle G, k \rangle$ be an instance of CLIQUE. Let G_1 be a complete graph on k vertices, and $G_3 = G$; clearly the pair $\langle G_1, G_3 \rangle$ can be constructed in polynomial time for given G and k . Since a clique in G is nothing but a complete subgraph, it follows that G has a clique of size k if and only if G_1 is isomorphic to a subgraph of G_3 . \square