Practice questions

Problems provided as practice questions for the midterm.

1. Sort the following terms from *slowest* growing to *fastest* growing. If two terms are equivalent asymptotically, draw a circle around them in your ordering.

$$(\log n)^2 + n \quad 3^{2n} \quad n^{1/4} \quad n^{\log_3 7} \quad 2^{3n} \quad 1000(\log n)^2 \quad 2^{\log_2 n} \quad n \log_5 n \quad 5^{\log_3 n}$$

$1000(\log n)^2 \quad n^{1/4}, \quad \{2^{\log_2 n}, (\log n)^2 + n\}, \quad n \log_5 n, \quad 5^{\log_3 n} = n^{\log_3 5}, \quad n^{\log_3 7}, \quad 2^{3n} = 8^n, \quad 3^{2n} = 9^n$

Useful facts: $a^{log_b n} = n^{log_b a}$

2. (2pts each) In each of the following cases, indicate if $f = O(g)$, $f = \Omega(g)$, $f = \Theta(g)$.

	$f(n)$	$g(n)$	O	Ω	θ
(a.)	$n^2 \log n$	$1500n^2 + 10n$			
(b.)	$1.2 + 1.2^2 + \cdots + 1.2^n$	1.2^{n+2}			
(c.)	$\log_2 n$	$\log_8 n$			
(d.)	3^n	n^{10}			
(e.)	$n^{\log_4 5}$	$n^{\log_2 5}$			

	$f(n)$	$g(n)$	O	Ω	θ
(a.)	$n^2 \log n$	$1500n^2 + 10n$	*N*	*Y*	*N*
(b.)	$1.2 + 1.2^2 + \cdots + 1.2^n$	1.2^{n+2}	*Y*	*Y*	*Y*
(c.)	$\log_2 n$	$\log_8 n$	*Y*	*Y*	*Y*
(d.)	3^n	n^{10}	*N*	*Y*	*N*
(e.)	$n^{\log_4 5}$	$n^{\log_2 5}$	*Y*	*N*	*N*

3. Maximum independent set: for a sequence of n numbers $x_1, ..., x_n$, find the set that has the largest sum and does not include consecutive pairs in $O(n)$ time. For example, the answer to the input 6, 4, 8, 2, 3, 5 is 6+8+5. Try to solve this problem without any hint. But if you are indeed stuck, look at the hint[1].

Let $L(i)$ be the maximum sum achievable considering sequence $x_1, ..., x_i$. The optimal solution to this subproblem can either use x_i or not use x_i. If it uses x_i, then the maximum achievable will be $x_i + L(i-2)$, since x_{i-1} is not usable anymore. If it does not use x_i, then the maximum achievable will be $L(i-1)$. So the recursion is simply $L(i) = \max\{L(i-1), x_i + L(i-2)\}$. To compute, we start with $L(1) = x_1$ and fill in the value for $i = 2, ..., n$, and return $L(n)$ as the final output. To fill in each value, we only need to do a constant time comparison, so overall run time is $O(n)$.

[1]Let $L(i)$ be the maximum sum achievable considering sequence $x_1, ..., x_i$

4. You are given a string of n characters $s[1, ..., n]$, which is a corrupted text in which punctuation has vanished (e.g., "itwasthebestoftime..."). You need to reconstruct the document using a dictionary, which is available in the form of a boolean function $dict(\cdot)$: for any string w, $dict(w)$ returns *true* if w is a valid word, otherwise it returns $false$. Give a dynamic programming algorithm that determines whether the string $s[\cdot]$ can be reconstructed as a sequence of valid words. The run time should be at most $O(n^2)$, assuming a *dict* call takes unit time. Try to solve this problem without any hint. If you are truly stuck, consider the hint[2].

   Based on the definition of the subproblem provided in the footnote, the recursion is as follows: $L(i) =$ true if $L(j)$ is true and $dict(s[j+1, ..., i])$ is true for any $j = 1, ..., i-1$. We initialize $L(0) =$ true and fill in the value for L from $i = 1$ to n. To fill in each value $L(i)$, we need to iterate through $j = 1, ..., i-1$, thus leading to $O(n^2)$ run time.

5. Given two strings $x = x_1x_2\cdots x_n$ and $y = y_1y_2\cdots y_m$, we wish to find the length of their longest common substring, that is, the largest k for which there are indices i and j with $x_ix_{i+1}\cdots x_{i+k-1} = y_jy_{j+1}\cdots y_{j+k-1}$. Show how to do this in time $O(mn)$. Try to solve this problem without any hint. If you are truly stuck, consider the hint[3]

   Based on the definition of the subproblem provided in the footnote, the recursion is as follows: if $x_i = y_j$, $L(i,j) = 1 + L(i-1, j-1)$. Otherwise $L(i,j) = 0$.

   To compute, the base cases are $L(0,0) = 0$ and $L(i,0) = L(0,j) = 0$ for all $i = 1,\ldots,n$ and $j = 1,\ldots,m$. To fill in the 2-d table, we go from left to right, top to down, and output the largest value we encounter, i.e., $max_{i,j}L(i,j)$. Computing each entry requires constant time. Thus the overall running time is $O(mn)$.

6. We use Huffman's algorithm to obtain an encoding of alphabet $\{a, b, c\}$, with frequencies f_a, f_b, f_c. In each of the following cases, either give an example of frequencies $\{f_a, f_b, f_c\}$ that would yield the specified code, or explain why the code cannot possibly be obtained (no matter what the frequencies are).

   a Code: $\{0, 10, 11\}$

   b Code: $\{0, 1, 00\}$

   c Code: $\{10, 01, 00\}$

   a $\{2/3, 1/6, 1/6\}$;

   b Not possible. Not prefix free.

   c Not possible. Code is not optimal since it is not a full binary tree. $\{1, 01, 00\}$ *gives a shorter encoding.*

7. Under a Huffman encoding of n symbols with frequencies $f_1, f_2, \ldots, f_n$, what is the longest codeword could possibly be? Give an example set of frequencies that would produce this case.

   The longest would be $n-1$. An encoding of n symbols with $n-2$ of them having probabilities $\frac{1}{2}, \frac{1}{4}, ..., (\frac{1}{2})^{n-2}$ and two of them having probability $(\frac{1}{2})^{n-1}$ achieves this value.

8. A server has n customers waiting to be served. The service time required by each customer is t_i minutes for customer i. So if, for example, the customers are served in the order of increasing t_i, then the i-th customer has to wait for $\sum_{j=1}^{i} t_j$ minutes. We wish to minimize the total wait time

$$T = \sum_{i=1}^{n}(\text{time spent waiting by customer } i).$$

[2] Let $L(i)$ be the answer to the question, can $s[1, .., i]$ be reconstructed as a sequence of valid words? Now if we know $L(1), L(2), ..., L(i-1)$, how can we use them to figure out $L(i)$?

[3] Let $L(i,j)$ be the longest common substring between $x_1x_2\cdots x_i$ and $y_1y_2\cdots y_j$ terminating at i and j.

Prove the greedy algorithm that serves the customer in increasing order of t_i gives the optimal solution.
Suppose the greedy algo is not optimal, then there is an optimal order such that $t_i < t_j$ but customer j is served before customer i ($j < i$). Now if we just exchange i and j, we can show that the change of the total time is negative. One convenient fact to note is that the total run time can be expressed as

$$\sum_{i=1}^{n} t_i \times (n-i) \tag{1}$$

. To see this, we can see that everyone except for the first customer need to wait for the first customer to be served, so the first customer's contribution to the total wait time is $(n-1) \times t_1$. Similarly, the i-th customer's contribution to the total wait time is $(n-i) \times t_i$. With this, we can easily see that the change of wait time after exchanging i and j is:

$$T_{new} - T_{old} = (n-j) * t_i + (n-i) * t_j - ((n-i) * t_i + (n-j) * t_j) = (t_i - t_j) * (i-j) < 0$$

The total waiting time is shorter - a contradiction to the fact that the ordering before exchange is optimal.

Note that you don't have to use equation 1 to compute the change of total wait time. You can also do a simple enumeration of the wait time for the two ordering:

Old order:	t_1	t_2	t_3	$\dots$	t_j	t_{j+1}	$\dots$	t_i	$\dots$	t_n
Old wait time:	0	t_1	$t_1 + t_2$	$\dots$	$\sum_{k=1}^{j-1} t_k$	$\sum_{k=1}^{j} t_k$	$\dots$	$\sum_{k=1}^{i-1} t_k$	$\dots$	$\sum_{k=1}^{n-1} t_k$
New order:	t_1	t_2	t_3	$\dots$	t_i	t_{j+1}	$\dots$	t_j	$\dots$	t_n
Net change (new-old):	*0*	*0*	*0*	*0*	*0*	$t_i - t_j$	$t_i - t_j$	$t_i - t_j$	*0*	*0*

It is easy to see that the wait time only change for the position $j+1$ to i, each changing by $t_i - t_j$, leading to a total change of $(t_i - t_j) \times (i - j)$.