Prob 1

String Matching

1. input: - a string to match of size m
   - a string ~~t~~ to search of size n
   - an alphabet of characters Σ e.g. $\{\Sigma = \{1,0\}, \Sigma = \{1,2,a,b..\}\}$

2. Output:
   - true ~~of~~ false, string found
   - if true, index of string

3. Methods:
   - Naive approach, going through every index of the searchable text and checking for pattern match
   - All other approaches use pre-processing to make more efficient search
     - Finite state automaton
     - Rabin-Karp-
     - Knuth-Morris-Pratt
     - Boyer-Moore

4. Importance
   - closely related to theory of computing (finite state automata)
   - used in all search applications
   - important in DNA sequencing

5. I like the problem for its simplicity and its multitude of complex solutions

6. Measure the size
   m for number of characters in pattern
   n for size of searchable text
   $|\Sigma|$ for number of characters in alphabet

Here is the ordering, where functions on the same line are in the same equivalence class, and those higher on the page are Ω of those below them:

$2^{2^{n+1}}$	
2^{2^n}	
$(n+1)!$	
$n!$	see justification 7
e^n	see justification 1
$n \cdot 2^n$	
2^n	
$(3/2)^n$	
$(\lg n)^{\lg n} = n^{\lg\lg n}$	see identity 1
$(\lg n)!$	see justifications 2, 8
n^3	
$n^2 = 4^{\lg n}$	see identity 2
$n \lg n$ and $\lg(n!)$	see justification 6
$n = 2^{\lg n}$	see identity 3
$(\sqrt{2})^{\lg n} (= \sqrt{n})$	see identity 6, justification 3
$2^{\sqrt{2 \lg n}}$	see identity 5, justification 4
$\lg^2 n$	
$\ln n$	
$\sqrt{\lg n}$	
$\ln \ln n$	see justification 5
$2^{\lg^* n}$	
$\lg^* n$ and $\lg^*(\lg n)$	see identity 7
$\lg(\lg^*)n$	
$n^{1/\lg n} (= 2)$ and 1	see identity 4

Much of the ranking is based on the following properties:

- Exponential functions grow faster than polynomial functions, which grow faster than polylogarithmic functions.
- The base of a logarithm doesn't matter asymptotically, but the base of an exponential and the degree of a polynomial do matter.

We have the following *identities*:

1. $(\lg n)^{\lg n} = n^{\lg \lg n}$ because $a^{\log_b c} = c^{\log_b a}$.
2. $4^{\lg n} = n^2$ because $a^{\log_b c} = c^{\log_b a}$.
3. $2^{\lg n} = n$.
4. $2 = n^{1/\lg n}$ by raising identity 3 to the power $1/\lg n$.
5. $2^{\sqrt{2 \lg n}} = n^{\sqrt{2/\lg n}}$ by raising identity 4 to the power $\sqrt{2 \lg n}$.
6. $\left(\sqrt{2}\right)^{\lg n} = \sqrt{n}$ because $\left(\sqrt{2}\right)^{\lg n} = 2^{(1/2)\lg n} = 2^{\lg \sqrt{n}} = \sqrt{n}$.
7. $\lg^*(\lg n) = (\lg^* n) - 1$.

The following *justifications* explain some of the rankings:

1. $e^n = 2^n (e/2)^n = \omega(n 2^n)$, since $(e/2)^n = \omega(n)$.
2. $(\lg n)! = \omega(n^3)$ by taking logs: $\lg(\lg n)! = \Theta(\lg n \lg \lg n)$ by Stirling's approximation, $\lg(n^3) = 3 \lg n$. $\lg \lg n = \omega(3)$.
3. $(\sqrt{2})^{\lg n} = \omega\left(2^{\sqrt{2 \lg n}}\right)$ by taking logs: $\lg(\sqrt{2})^{\lg n} = (1/2) \lg n$, $\lg 2^{\sqrt{2 \lg n}} = \sqrt{2 \lg n}$. $(1/2) \lg n = \omega(\sqrt{2 \lg n})$.
4. $2^{\sqrt{2 \lg n}} = \omega(\lg^2 n)$ by taking logs: $\lg 2^{\sqrt{2 \lg n}} = \sqrt{2 \lg n}$, $\lg \lg^2 n = 2 \lg \lg n$. $\sqrt{2 \lg n} = \omega(2 \lg \lg n)$.
5. $\ln \ln n = \omega(2^{\lg^* n})$ by taking logs: $\lg 2^{\lg^* n} = \lg^* n$. $\lg \ln \ln n = \omega(\lg^* n)$.
6. $\lg(n!) = \Theta(n \lg n)$ (equation (3.19)).
7. $n! = \Theta(n^{n+1/2} e^{-n})$ by dropping constants and low-order terms in equation (3.18).
8. $(\lg n)! = \Theta((\lg n)^{\lg n + 1/2} e^{-\lg n})$ by substituting $\lg n$ for n in the previous justification. $(\lg n)! = \Theta((\lg n)^{\lg n + 1/2} n^{-\lg e})$ because $a^{\log_b c} = c^{\log_b a}$

Problem 3

1. $O(n^b) = (\frac{n}{a}+c)^b$ for constants $a>0, b>0$ and c

$f(n) = (\frac{n}{a}+c)^b = (\frac{1}{a})^b (n+ac)^b = k(n+ac)^b$ where k is constant

$g(n) = n^b$

without loss of generality, suppose there is a c_0 and $n=n_0$ where $f(n_0) \leq c_0 g(n_0) \Rightarrow k(n+ac)^b \leq c_0 n^b$ for $n=n_0$

Then there exists no $n_1 = n$ that is sufficiently large ~~that~~ makes the inequality false. ~~This contradicts the definition of big-O so the statement of~~ Therefore, $O(n^b) = (\frac{n}{a}+c)^b$ is ~~false~~ true.

2. $T(n) = 4T(\frac{n}{2}) + n^2$

Using the master theorem on p. 94, using case 2 with $a=4$ & $b=2$ for $f(n) = \Theta(n^{\log_2 4})$ we find $T(n) = \Theta(n^2 \log n)$.

[✓ C]

3. $T(n) = T(n-1) + \frac{1}{2^n}$

$$T(n) = \underbrace{\frac{1}{2^n} + \frac{1}{2^{n-1}} + \frac{1}{2^{n-2}} + \cdots \frac{1}{2^2} + \frac{1}{2^1}}_{\text{This value approaches } 1}$$

$$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} \cdots\cdots \Rightarrow 1$$

so, $T(n) = \Theta(1)$

[✓ d]

Problem 4

We have n students and m hospitals. Each hospital m_i has k_i open positions to fill. The total # of positions is $p = \sum_{i=1}^{m} k_i$ $p \geq n$. We propose the following algorithm to fill each p_j position.

ALGORITHM

```
While there exists an unfilled position $p_j$ {
    The hospital $m_i$ that contains $p_j$ selects the highest
        ranked student s on $m_i$'s list that has
        not already been proposed to

    If s is free {
        add student s to position $p_j$
    }
    else // student already engaged {
        if student s prefers $p_j$ to its already engaged
            job $p'$, add student s to $p_j$ and remove from $p_i$
    }
}
```

PROOF OF STABILITY

Since there are m hospitals and each hospital has a list of n students, ~~and~~ at worst case, each hospital will go through all of them, the complexity is $O(mn)$.

Instability 1 cannot occur because if h prefers s' to s, h would have proposed to s' first. Since once a student has a position it will never be free again, h must never have proposed to s' so this is a contradiction.

Instability 2 cannot occur because if h preferred s' to s, h would have proposed to s' first. s' also would have withdrawn from ~~h~~ to go to h', so s' would prefer h' to h. Therefore, the instability cannot happen.

Problem 5

For the algorithm, we use the greedy approach and always merge the two smallest files.

ALGORITHM

```
Given a sorted list of n files
while there is more than one file
    merge the two shortest files $f_i$ and $f_j$ and put the result $f'$
        back into the list sorted, removing $f_i$ and $f_j$
```

RUNNING TIME

The time to initially sort the files is $4n \log n$. The time to find the two smallest files is 2. The time to insert a new file into the sorted list is $O(\log n)$. For a list of n files there will be exactly $n-1$ merge actions or $O(n)$. ~~The sum of the file sizes is $\sum_{i=1}^{n} L_i = N$. The time for a merge action in the worst case is $O(N)$.~~

Neglecting the running time of a merge action, the running time of the algorithm is $O(n \log n)$, dominated by the pre-processing of the sorted list. If the merge action is not constant, then the algorithm is likely to be $O(nO(p))$ where $O(p)$ is some complexity as a function of the merge size.

PROOF OF CORRECTNESS

We notice that the cost function only has a constant modifier and does not affect the decision ~~of~~ finding the optimal merge order. We also recognize this problem maps to the problem of finding the optimal Huffman codes where each file f_i represents a character c_i and the character's frequency corresponds to the file's size. Using the proof from the book, Theorem 16.4, we see that the greedy approach is optimal.