Exercise Sheet #1, Computer Science, 308-251A
M. Hallett, Kaleigh Smith, X Fall 2002
Due: Sept. 26, 2002, midnight
Drop Box: 308-251 drop box (McConnell, 1st Floor, North Wing)

Hand in solutions to questions that gave you difficulty, if you are unsure if you have the correct answer or, if you would like advice from the TAs.

Question 1: [Asymptotic Growth] (from 2001)

1. Prove $(n+a)^b = \Theta(n^b)$, for any real constants a and b, where $b > 0$.
2. Prove or disprove: $2^{n+1} = O(2^n)$
3. Prove or disprove: $2^{2n} = O(2^n)$
4. Prove or disprove: $2^{log_b n} \neq \Theta(2^{log_a n})$, if $a \neq b$.
5. Is $\lceil lg\ n \rceil !$ polynomially bounded?
6. Is $\lceil lg\ lg\ n \rceil !$ polynomially bounded?
7. Show $log(n!) = \Theta(nlogn)$.
8. Show $\Sigma_{i=1}^{n} i^{-1} = \Theta(logn)$.

Question 2: [Pure Induction] (from 2001) Prove by induction that the i^{th} Fibonacci number satisfies the equality $F_i = (\phi^i - \widehat{\phi}^i)/\sqrt{5}$, where ϕ is the golden ratio and $\widehat{\phi}$ is its conjugate.

Question 3: [Bounding] (from 2001) Give bounds on the following summations:

1. $\Sigma_{k=1}^{n} k^r$, $r \geq 0$.
2. $\Sigma_{k=1}^{n} lg^s k$, $s \geq 0$.
3. $\Sigma_{k=1}^{n} k^r \cdot lg^s k$, $r, s \geq 0$.

Question 4: [Recurrences] (from 2001) Let α be a constant, $0 < \alpha < 1$. Solve the following recurrence:

$$T(n) = T(\alpha \cdot n) + T((1-\alpha) \cdot n) + n.$$

Question 5: [Recurrences] IMPORTANT QUESTION (4.7 in CLR ed. 1, page 75; 4.6 in CLRS ed. 2, page 88)

Question 6: (midterm 2001)
Let $A =< a_1, a_2, \ldots, a_n >$ be a list of integers. Also, let $X = \{x_1, x_2, \ldots, x_k\}$ be a set of integers, where k is a constant. Suppose we knew that each a_i, $1 \leq i \leq n$, is restricted to a value $x_j \in X$ for some j, $1 \leq j \leq k$.

(a) Give an $O(n)$ algorithm to sort A. **This must be a comparison sort.**

(b) Suppose k was not a constant but a function of n (call it $f(n)$). Given a tight upper bound on $f(n)$ for which your algorithm from part (a) is asymptotically no worse than the lower bound

for ComparisonSort.

(c) Consider the following variant of the problem from part (a) where we know that each a_i takes one of k values from the set $X_i = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$. That is, each a_i has its own private set of k values.

Does the $\Omega(n \cdot log\ n)$ lower bound for Comparison Sort hold for this problem? If "yes", give a concrete example of why. If "no", give an algorithm that is $o(n \cdot log\ n)$.

Question 7: (midterm 2001) The version of MergeSort we saw in class splits the input array A into two (almost equal) halves. An obvious question is "it is advantegeous to split A into more than 2 pieces at each level of the recursion?". Suppose we split A into $\lfloor\sqrt{n}\rfloor$ pieces of roughly $\lfloor\sqrt{n}\rfloor$ elements.

(a) Give a recurrence describing the running time of this algorithm.

(b) Draw a recursion tree for this recurrence. Make sure you label it with the size of the subarray, the degree of branching, and the total work done at each level of the tree.

(b) Give a tight upper bound on this recurrence. Try to make it as far as you can in the time permitted. Bound what you can.

Question 8: (2002) Practice Proof Prove (carefully and very clearly) that $o(g(n)) \cap \omega(g(n)) = \emptyset$.

Question 9: (2002) Suppose that we can show that algorithm Φ is $O(n^d)$ where n is the size of the input and d is a constant. This means that Φ can be computed in polynomial time (as compared to super-polynomial or exponential time). However, the algorithm might not be *practical*. Although the running time of the algorithm is bounded by a polynomial there may be certain things the $big - O$ notation hides that make it effectively impractical. Give two examples of how $big - O$ notation can be abused.

Question 10: (2002) Let S be a binary sequence of length n. That is, $S = s_0 s_2 \ldots s_{n-1}$ where $s_i \in \{0, 1\}$. We say that σ is a *substring* of S iff there exists i, j, $0 \leq i \leq j \leq n-1$, such that $s_i s_{i+1} \ldots s_j = \sigma$.

If S is a *binary cyclic sequence*, then consider s_{n-1} to be adjacent to s_0. In other words, $s_n = s_0$.

A *de Bruijn sequence* B_k for $k \in \mathbf{N}, k > 0$, is a binary cyclic sequence of length 2^k such that every possible binary string with k bits is a subtring of B_k exactly once.
For example, $B_3 = 00011101$. Verify that every 3 bit sequence is present in B_3 exactly once.

(1) Construct B_4.

(2) Try to find an algorithm to find B_k for any k that runs in time $2^{2k} \cdot k$.

(3) **(hard)** Try to prove your algorithm correct. Use your intuition but do not use more than 1/2 a page.

Question 11: (2002) You are given an ordered sequence $A =< a1, \ldots, a_n >$ where A is a permutation of $\{1, \ldots, n\}$. You now pick an element e_1 from A, and remove e_1 from A. This is your first pick. You pick a second element e_2 from A and remove e_2 from A. This is your second pick. You do this a total of k times. What is the probability that $e_1 < e_2 < \ldots < e_k$? What is the probability that $e_1 \leq e_2 \leq \ldots \leq e_k$ if you don't remove e_i, $1 \leq i \leq k$, from A when you pick it.

Question 12: (midterm 2002)
You are given 7 natural numbers $a_1, \ldots, a_7$ and you are asked to find the median (the element a_i that has rank 4, $1 \leq i \leq 7$). Give a lower bound for the number of comparison operations needed and show your work.

Question 13: (midterm 2002)
Consider the following variant of `MergeSort`:

```
LazyMergeSort(A, p, r, α)
    if p < r then
        Choose a random number z between 0 and 1.
        If z ≤ α, then
            q ← ⌊(p + r)/2⌋
            LazyMergeSort(A, p, q, α)
            LazyMergeSort(A, q + 1, r, α)
            Merge(A, p, q, r)
        else
            for i from p to r do
                A[i] ← −i
                od
```

The input A is a set of n distinct positive natural numbers and α is a real number, $0 < \alpha < 1$. The output is a set A of n numbers (not necessarily positive). What is the probability that the array A is sorted into ascending order after calling `LazyMergeSort`$(A, 1, n, \alpha)$?

Question 14: (midterm 2002)
Intel releases a new chip that has the following features:

- There are two registers r_1 and r_2 that each hold an integer;
- Only three operations can be performed on r_1 and r_2:
  - (Equality operation) We can test for equality between r_1 and r_2. This test returns `true` iff the contents of r_1 equal the contents of r_2. `false` otherwise.
  - (Add operation) We can add 1 to register r_1 or r_2. That is, we can perform $r_i \leftarrow r_i + 1$, for $i = 1$ or 2.
  - (Subtract operation) We can subtract 1 from register r_1 or r_2. That is, we can perform $r_i \leftarrow r_i - 1$, $i = 1$ or 2.
  - We can not do anything else to register r_1 or r_2.
- All other typical programming constructs are available (`for` or `while` loops, multiplication, divisions, variables, etc.).

Part (a): Suppose there is an integer in r_1 and an integer in r_2. Give an algorithm to find the maximum of the value in r_1 and the value in r_2 (thereby implementing a $<$ operator). Try to minimize the number of Equality, Add, and Subtract operations.

Part (b): Give a tight upper bound on the number of Equality, Add and Subtract operations performed by your algorithm (use "big-Oh" notation).

Part (c): Give an upper bound on how many Equality operations are performed by your algorithms. Do not use "big-Oh" notation (You do not need to show a proof.)