1. For the functions satisfying the following three recurrences, determine which is the fastest growing and which is the slowest growing

(a) \( T(n) = 2T(n - 1), T(1) = 5. \)

(b) \( S(n) = S(n - 1) + \lceil \log(n) \rceil, S(1) = 17. \)

(c) \( R(n) = 3R(\lfloor n/2 \rfloor) + 4, R(1) = 11. \)

We have \( T(n) = 5 \cdot 2^{n-1}. \) We have \( S(n) = \left\lfloor \log(n) \right\rfloor + \left\lfloor \log(n - 1) \right\rfloor + \cdots + \left\lfloor \log(2) \right\rfloor + 17 \leq \log(n!) + 17 \in \Theta(n \log(n)) \) and similarly \( S(n) \geq \log(n!) - n + 17. \) By the master theorem for divide and conquer recurrences, \( R(n) \in \Theta(n \log^2(3)). \) Thus \( S(n) \in o(R(n)) \) and \( R(n) \in o(T(n)). \)

2. Prove that \( \log^2(n) \in O(2^{\sqrt{n}}). \)

It suffices to show that \( \log(n) \in O(2^{\sqrt{n}/2}). \) By L’Hospital’s rule,

\[
\lim_{n \to \infty} \frac{\log(n)}{2^{\sqrt{n}/2}} = \lim_{n \to \infty} \frac{(1/n) \log(e)}{2^{\sqrt{n}/2}(n^{-1/2}/4) \ln(2)} = \lim_{n \to \infty} \frac{\log(e)}{2^{\sqrt{n}/2}(n^{1/2}/4) \ln(2)} = 0.
\]

3. Give a proof by invariants that the following algorithm correctly searches a sorted list \( X[0], \ldots, X[n - 1] \) for element \( z. \)

```
Search(X,n,z) {
    i = 0
    j = n - 1
    while (i < j) {
        k = (i+j)/2
        if (X[k] = z)
            return(k)
        if (X[k] < z)
            i = k + 1
    }
    return(-1)
```

else j = k-1
}
if (X[i] = z)
    return(i)
else return(Failure)
}

Assumption: $X[1] \leq X[2] \leq \cdots \leq X[n-1]$

Correctness: P: If there is a $t$ so that $X[t] = z$, then the algorithm returns a $t$ with $X[t] = z$.

Invariant: R: If there is a $t \in \{0, 1, \cdots, n-1\}$ so that $X[t] = z$, then there is a $t \in \{i, \cdots, j\}$ so that $X[t] = z$.

R is true initially since $i = 0$ and $j = n-1$. Assume R at the start of an iteration. If $X[k] = z$, then the loop never repeats. Suppose there is a $t \in \{0, 1, \cdots, n-1\}$. By R we may assume $i \leq t \leq j$. If $X[k] < z$, then any $t$ with $X[t] = z$ satisfies $k < t$, so there is a $t \in \{k+1, \cdots, j\}$ with $X[t] = z$. If $X[k] > z$, then any $t$ with $X[t] = z$ satisfies $k > j$, so there is a $t \in \{i, \cdots, k-1\}$ with $X[t] = z$. Thus R holds at the start of the next iteration.

If the algorithm returns at either return statement, it is returning a correct value. Suppose the loop terminates without having returned, R holds, and there is a $t$ so that $X[t] = z$. Then there is a $t$ so that $X[t] = z$ and $i \leq t \leq j \leq i$, so $t = i$ and $i$ will be returned.

The loop eventually returns or terminates since $j - i + 1$ decreases at each iteration.

4. Let $H$ be an array containing a min-heap with $n$ elements (the smallest element is at the top, and the children of $H[i]$ are $H[2i]$ and $H[2i+1]$). Give pseudocode for an efficient algorithm for deleting the smallest element from $H$. What is the worst case time complexity of your algorithm?

Move the last element to $H[1]$ and then bubble it down until it’s smaller than both its children. Always swap with the smaller child.

```plaintext
x = H[1]
H[1] = H[n]
n = n-1
k = 1
while (2k+1 <= n and H[k] > min(H[2k],H[2k+1])) {
    if (H[2k] < H[2k+1])
        j=2k
else j=k-1
    if (X[i] = z)
        return(i)
    else return(Failure)
}
```

$x = H[1]$
$H[1] = H[n]$
n = n-1
k = 1
while (2k+1 <= n and H[k] > min(H[2k],H[2k+1])) {
    if (H[2k] < H[2k+1])
        j=2k
else j=k-1
    if (X[i] = z)
        return(i)
    else return(Failure)
}
else \( j = 2k+1 \)
swap(H[k],H[j])
k = j
}
if (2k \leq n \text{ and } H[k] > H[2k])
swap(H[k],H[2k])
return(x)

Time: the number of iterations is at most the height of the heap, which is at most \( \lceil \log(n) \rceil \).
So time is \( O(\log(n)) \).

5. (a) Find a sharp upper bound on the height of a 2-3 tree with \( n \) nodes.

Every internal node has at least 2 children, and every leaf is at the same depth, so if the height is \( h \) there are at least \( 1 + 2 + 2^2 + \cdots + 2^h = 2^{h+1} - 1 \) nodes. That is, \( 2^{h+1} - 1 \leq n \), so \( h \leq \log(n+1) - 1 \).

(b) Compare the performance of search trees, red-black trees, B-trees, or other such structures under various mixes of the basic operations (search, insert, delete).

There are lots of possible variants of this and lots of different answers. The main idea is to discuss time complexity of the various operations, how much extra memory is needed, ease of programming, situations where one or another structure is preferred, and so on.

6. Describe an efficient algorithm to count the connected components in an undirected graph.

Analyze the complexity of your algorithm. It should run in linear time.

Use a counter \( c \) to count the components, initialized to 0. While possible, pick an unvisited node \( u \) and do a DFS from \( u \). After each DFS, increment \( c \). The final value of \( c \) is the number of connected components. Time is \( O(n + e) \) since this is the time of DFS.

7. Let \( G = (V, E) \) be a directed graph with edge capacities \( c(u, v) \geq 0 \), a sink \( s \), and a source \( t \). Prove that if \( f \) is a flow on \( G \) and \( f' \) is a flow on the residual graph \( G_f \), then the function \( g \) defined by \( g(u, v) = f(u, v) + f'(u, v) \) is a flow on \( G \).

The symmetry and conservation laws are linear equations so are preserved by addition of the flows.

To prove the capacity bound, for every edge \( e \) we have \( f'(e) \leq c_f(e) = c(e) - f(e) \). Thus \( f(e) + f'(e) \leq f(e) + c(e) - f(e) = c(e) \).

8. Consider the following problem:
**Instance:** An integer $K \geq 0$ and $n$ distinct positive integers $x_1, \ldots, x_n$.

**Output:** A subset of $\{x_1, \ldots, x_n\}$ whose sum is precisely $K$ (if there is such a subset).

Design an $O(nK)$ algorithm that solves this problem. Give pseudocode, explain your pseudocode, and analyze the algorithm. (HINT: Use dynamic programming).

We need to compute extra information. For each pair $(z, j)$ with $1 \leq K$ and $1 \leq n$ we find a subset $V[z, j]$ of $X_j = \{x_1, \ldots, x_j\}$ whose sum is $z$, if there is one. We have

$$V[z, j] = \begin{cases} V[z - x_j, j - 1] \cup \{x_j\} & \text{if } V[z - x_j, j - 1] \text{ is defined} \\ V[z, j - 1] & \text{if } V[z, j - 1] \text{ is defined.} \end{cases}$$

**Pseudocode:**

```plaintext
time = O(nK) since it’s just a pair of nested loops of size $K$ and $n - 1$.

9. (a) Give pseudocode for and explain the Euclidean algorithm (which finds the GCD of two integers).

```````````
At each stage \( a = qb + (a \pmod{b}) \) for some \( q \), so \( \gcd(a, b) = \gcd(b, a \pmod{b}) \). At the end, \( \gcd(a, 0) = a \).

(b) Analyze the time complexity of the Euclidean algorithm.

Simple analysis: in two iterations we have \((a, b)\) replaced by \((a \pmod{b}, c)\) for some \(c < a \pmod{b}\). If \( a \geq 2b \), then \( a \pmod{b} < b \leq a/2 \). If \( b \leq a < 2b \), the \( a \pmod{b} = a - b < a - a/2 = a/2 \). Thus after 2 iterations the larger term is at least divided by 2. But since the first term is the second term of the next stage, the smaller of \( a \) and \( b \) is also divided by 2 after 2 iterations. Thus the algorithm takes at most time \( O(\lg(\min(a, b))M(\max(a, b))) \), where \( M(n) \) is the time needed to divide two numbers that are less than or equal to \( n \).

(c) Show that the problem of testing whether an integer is not a prime number is in the complexity class NP.

Here is a nondeterministic polynomial time algorithm for this problem: Given \( n \), guess integers \( a, b \). If \( a < n \) and \( b < n \) and \( ab = n \) then return “yes” else return “no”.

10. (a) Define the term “polynomial time mapping reduction.”

A polynomial time mapping reduction from language \( L \) to language \( K \) is a polynomial time computable function \( f \) from strings to strings satisfying \( x \in L \) if and only if \( f(x) \in K \).

(b) HITTING SET is the problem:

**Instance:** A collection \( C = \{C_1, \cdots, C_m\} \) of subsets of a set \( S \) (i.e., each \( C_i \subseteq S \)), and a positive integer \( k \).

**Question:** Does \( S \) contain a subset \( T \) such that \( |T| \leq k \) and \( T \) contains at least one element of each \( C_i \)?

Prove that HITTING SET is NP-complete. (Hint: use the fact that VERTEX COVER is NP-complete).

First, HITTING SET is in \( NP \) because we can guess the set \( T \) of size at most \( k \) and look for each element of each \( C_i \) in \( T \). This takes \( O(k|\cup_i C_i|) \) which is polynomially bounded.

Next we show a reduction from VC to HITTING SET. Let \( G = (V, E) \) be a graph and \( b \) an integer. Let \( S = V \), and for each edge \( e = (u, v) \in E \), let \( C_e = \{u, v\} \). Then a hitting set for \( S, C \) is a vertex cover of \( G \). Thus \( f(G, b) = (S, C, b) \) is a reduction from VC to HITTING SET.