Question 1 (Fibonacci Computation, 30 points). Consider the following algorithm for computing Fibonacci numbers (you do not need to show that this is correct):

Fib(n):
   If n = 0, Return 0
   If n = 1 or 2, Return 1
   Let m = Floor(n/2)
   Return Fib(m)*Fib(n-m-1)+Fib(m+1)*Fib(n-m)

(a) Give a recurrence relation for the runtime of Fib(n). Assume that multiplying Fib(a) by Fib(b) takes \(O(a \cdot b)\) time. [15 points]

The first three lines of the program take constant time. The program then makes four recursive calls to problems of approximately half the size, and has to perform multiplications taking \(O(n^2)\) time. Thus, the runtime is given by the recurrence

\[ T(n) = 4T(n/2 + O(1)) + O(n^2). \]

(b) What is the asymptotic runtime of Fib(n)? [15 points]

We apply the Master Theorem with \(a = 4, b = 2\) and \(d = 2\). Note that \(a = b^d\). Thus, the runtime is \(O(n^d \log(n)) = O(n^2 \log(n))\).
Question 2 (Tour Scheduling, 35 points). Let $G$ be a DAG with a source $s$ and a sink $t$. Each edge of $G$ has an associated cost $c(e)$, and each vertex and associated payout $p(v)$. The profit of a path in $G$ is the sum of the payouts of the vertices in the path minus the sum of the costs of the edges in the path. Give an algorithm that given $G, s, t, c$ and $p$ computes the highest possible profit on an $s-t$ path in $G$. For full credit, your algorithm should run in linear time or better.

Construct a weighted graph $G'$ in the following way. The graph structure of $G'$ is identical to that of $G$, but the weight $\ell(u, w)$ of an edge $(u, w)$ is given by $\ell(u, w) = c((u, w)) - p(w)$. Our algorithm uses the shortest paths in DAGs algorithm to compute the length $L$ of the shortest $s-t$ path in $G'$ and returns $p(s) - L$.

The construction of $G'$ is clearly linear time, as is the shortest paths in DAGs algorithm. This completes our runtime analysis.

As for correctness, note that given a path $P$ from $s$ to $t$ that $p(s)$ minus the length of $P$ in $G'$ is exactly the sum of the payouts of the vertices of $P$ minus the sum of the costs of the edges, or equivalently the profit in $G$ of the path. Thus, maximizing the profit of $P$ is equivalent to minimizing the length of $P$ (which is done by the shortest path algorithm), and the profit is $p(s)$ minus the length.
**Question 3** (Target Summation, 35 points). Let $L = \{v_1, v_2, \ldots, v_n\}$ be a list of distinct positive numbers given in random order. Let $T$ be a positive real number. Call a subset $S$ of $L$ good if the sum of its elements is at least $T$ (note that sets cannot have repeated elements, so $S$ uses each element of $L$ at most once). Give an algorithm that given $L$ and $T$ computes the smallest possible size of such a set. For full credit, your algorithm should run in time $O(n)$ expected time or better.

We model our solution off of the order statistics algorithm. Note that the optimal $S$ will always contain the $k$ largest elements of $L$ for some value of $k$. Given an element $x \in L$ we can divide $L$ into $L_-, L_+, L_=$, the subsets consisting of elements less than, more than, and equal to $x$. We note that if the total sum of the elements of $L_+$ is at least $T$, then we can reach our goal using only heavier elements, and the optimal solution will require only some subset of $L_+$. If the sum of $L_+$ is less than $T$ but $L_+$ and $x$ is enough, then the best set is $L_+ \cup L_=$. If, this sum is still not enough, the best set must include $L_+ \cup L_-$ plus the best subset of $L_-$ adding to at least $T$ minus the sum of the elements in $L_+ \cup L_=$. This gives us the following algorithm:

```
NumElts(L, T)
Let x be a random element of L
Split L into L_+, L_- and L_= consisting of the elements
    greater than, less then, and equal to x, respectively.
Let S be the sum of the elements in L_+
If S >= T
    Return NumElts(L_+, T)
Else If S+x >= T
    Return 1+|L_+|
Else
    Return 1+|L_+|+NumElts(L_-, T-S-x)
```

To analyze the runtime, note that the split operation takes $O(n)$ time, as does the computation of $S$. We then have a recursive call on either $L_+$ or $L_-$ (unless we got lucky and hit the return statement). With at least 50% probability, $x$ was between the first and third quartiles of the elements of $L$, in which case this recursive call has size at most $3n/4$. Therefore, we get a recurrence for the expected runtime of:

$$T(n) = O(n) + T(3n/4).$$

By the Master Theorem, this implies that $T(n) = O(n)$.