Q2 a) For the purpose of contradiction,

assume $\exists\, x_i$ that $x_i > f_i$ for $i \geq 2$.

Since $f_i = \frac{d_{i-1}}{d_i}$,

$$x_i > \frac{d_{i-1}}{d_i} \Rightarrow x_i \cdot d_i > d_{i-1}$$

Then, we can always optimal S_g by moving the part $(x_i \cdot d_i - d_{i-1})$ to x_{i-1}:

$$T_0 = x_1 \cdot d_1 + \cdots + x_{i-1} \cdot d_{i-1} + x_i \cdot d_i + \cdots + x_n \cdot d_n$$

$$= x_1 \cdot d_1 + \cdots + x_{i-1}' \cdot d_{i-1} + x_i' \cdot d_i + \cdots + x_n \cdot d_n$$

where $x_{i-1}' = x_{i-1} + \lfloor \frac{x_i \cdot d_i}{d_{i-1}} \rfloor$

$x_i' = x_i - \lfloor \frac{x_i \cdot d_i}{d_{i-1}} \rfloor \cdot \frac{d_{i-1}}{d_i} = x_i - \lfloor \frac{x_i \cdot d_i}{d_{i-1}} \rfloor \cdot f_i$

Now, $x_1 + \cdots + x_{i-1}' + x_i' + \cdots + x_n < x_1 + \cdots + x_{i-1} + x_i + \cdots + x_n$, since $f_i > 1$

Thus, if S_g is optimal, $0 \leq x_i < f_i$ for all $i \geq 2$

Q2 b) Assume $\exists\, y_i$ that $y_i > f_i$ for $i \geq 2$.

Since $f_i = \frac{d_{i-1}}{d_i}$,

$$y_i > \frac{d_{i-1}}{d_i} \Rightarrow y_i \cdot d_i > d_{i-1}$$

Then, we can always optimal S by moving the part $(y_i \cdot d_i - d_{i-1})$ to y_{i-1}:

$$T_0 = y_1 \cdot d_1 + \cdots + y_{i-1} \cdot d_{i-1} + y_i \cdot d_i + \cdots + y_n \cdot d_n$$

$$= y_1 \cdot d_1 + \cdots + y_{i-1}' \cdot d_{i-1} + y_i' \cdot d_i + \cdots + y_n \cdot d_n$$

where $y_{i-1}' = y_{i-1} + \lfloor \frac{y_i \cdot d_i}{d_{i-1}} \rfloor$

$y_i' = y_i - \lfloor \frac{y_i \cdot d_i}{d_{i-1}} \rfloor \cdot \frac{d_{i-1}}{d_i} = y_i - \lfloor \frac{y_i \cdot d_i}{d_{i-1}} \rfloor \cdot f_i$

Now, $y_1 + \cdots + y_{i-1}' + y_i' + \cdots + y_n < y_1 + \cdots + y_{i-1} + y_i + \cdots + y_n$, since $f_i > 1$

Therefore, S is not optimal.

Q2c). Suppose S is a hypothetical optimal solution.

Let $y_{i^*} \neq x_{i^*}$ that i^* is the highest index where S_g and S differ.

- Suppose $i^* \geq 2$, then

$$x_1 = y_1, \ x_2 = y_2, \ \dots, \ x_{i^*-1} = y_{i^*-1}, \ x_{i^*} \neq y_{i^*}$$

and so $T_{i^*-1} = T_0 - x_1 \cdot d_1 - \dots - x_{i^*-1} \cdot d_{i^*-1} < d_{i^*-1}$

$T_{i^*-1} > x_{i^*} \cdot d_{i^*} > y_{i^*} \cdot d_{i^*}$ since $x_{i^*} \in S_g$ (y_{i^*} cannot be greater than x_{i^*})

Then, $(x_{i^*} - y_{i^*}) d_{i^*} \geq d_{i^*}$

$T_{i^*} = T_{i^*-1} - y_{i^*} \cdot d_{i^*} \geq d_{i^*}$

By the algorithm, $y_{i^*+1} = \left\lfloor \frac{T_{i^*}}{d_{i^*+1}} \right\rfloor \geq \frac{d_{i^*}}{d_{i^*+1}} = f_{i^*}$ which contradicts the hypothetical assumption.

Thus, from (b), we know S is not optimal.

- Suppose $i^* = 1$, then $y_1 < x_1$ since $x_1 \in S_g$.

Now, $y_2 > x_2$. moreover, $y_2 = x_2 + \frac{(x_1 - y_1) d_1}{d_2}$

Then $y_1 + y_2 + y_3 + \dots + y_n$

$= [x_1 - (x_1 - y_1)] + [x_2 + (x_1 - y_1) \cdot f_2] + y_3 + \dots + y_n$

$= x_1 + x_2 + (f_2 - 1)(x_1 - y_1) + y_3 + \dots + y_n$

$> x_1 + x_2 + x_3 + \dots + x_n$ since $f_2 > 1$

This contradicts that S is optimal.

Therefore, if i^* exists, S is not optimal.

Q3.

We will define our greedy algorithm as follow.

① sort the requests by $f(r_i)$. Let $r_{(i)}$ be i-th request in order.

② Let A be the set of optimal r_i's, let $r_a = r_{(1)}$

③ Then, we will pick r_b such that r_b intersects r_a and $f(r_b)$ is the Latest.

④ Insert r_b into A. Remove all requests that intersects r_b. Let $r_a = r_{(\text{next smallest})}$ and repeat ③. ④ until no requests left.

⑤ Return A.

Key claim: (Greedy stays ahead)

Suppose S_g is the optimal greedy solution and S be an arbitrary solution.

$S_g = \{r_{g_1}, \dots, r_{g_k}\}$ $S = \{r_{h_1}, \dots, r_{h_{k-1}}\}$

We claim that for $r_{g_i} \in S_g$, $r_{h_i} \in S$, r_{g_i} can "cover" equal or wider than r_{h_i}

("cover" means the total time period of all requests that it overlaps)

Proof of optimality:

Base on the key claim, we know that the total "coverage" of $\{r_{g_1}, \dots, r_{g_{k-1}}\}$ is "wider" than that of $\{r_{h_1}, \dots, r_{h_{k-1}}\}$. However, S_g has one more r_{g_k} to cover all the requests. Thus, S cannot cover all the requests. Therefore, S_g is optimal.

Proof of key claim:

By induction:

Let $P(n)$ be the statements that r_{g_i} covers equal or wider than r_{h_i} for $1 \le i \le n$.

- Base case: $n=1$: $P(1)$ holds since r_{g_1} overlaps all requests, the coverage is $f(r_{(1)}) - s(r_{(n)})$
- Induction Hypothesis: $P(k)$ holds
- Induction Conclusion:

To prove $P(k+1)$: By IH. $r_{g_1}, \dots, r_{g_k}$ covers equal or wider than $r_{h_1}, \dots, r_{h_k}$.

If we add one more $r_{g_{k+1}}$ and $r_{h_{k+1}}$, suppose $r_{h_{k+1}}$ makes the coverage wider. Then, $r_{h_{k+1}}$ covers wider than $r_{g_{k+1}}$, which means my algorithm should pick $r_{h_{k+1}}$ over $r_{g_{k+1}}$. Thus, this is a contradiction. That is, $P(k+1)$ holds.

Thus, by POMI, key claim is proved.

Complexity Analysis:

The sort will use merge sort which is $O(n\log n)$

The process that iterates through all request is $O(n^2)$.

Thus, $T(n) \in O(n^2)$, which satisfies the requirement.

Q4.

We will design a dynamic programming algorithm and here's the set up:
Let $P(n,k)$ be the probability that A still need to win k states out of n states to win.
Let a_i be the probability that i-th state wins.
Assume for the i-th state, there are two possible results as ① win ② lose:

① i is a winning state, then we will calculate $a_i \cdot P(n-i, k-1)$ given $P(n-i, k)$ (at i-th state, for the rest of $n-i+1$ states, A needs k winning states)

② i is a losing state, then we will calculate $(1-a_i) \cdot P(n-i, k)$ given $P(n-i, k)$

If $n-i=k$, we know that A has to win all the rest of the states; if $k=0$, we know that A has get more than half winning states and so the probability of winning will be 1.

The algorith will be as follow:

```
WR(i, k, A, H):
  // i: i-th state in order;   k: the number of wins still need.
  // A: the set of probabilites such that A[i] is a_i (start from 1)
  // H: to store the value of P(n,k) that is already calculated.
  n ← length(A)
  if k = 0 then
     return 1
  if H[i,k] DNE:
     if (n−i) = k then
        H[i,k] ← A[i] * WR(i+1, k-1, A, H)
     else
        H[i,k] ← A[i] * WR(i+1, k-1, A, H) + (1−A[i]) * WR(i+1, k, A, H)
  return H[i,k]

WinRate(A).
  // A: A[i] means a_i (start from 1)
  n ← length(A)
  k ← ⌊n/2⌋ + 1
  H ← hash table (<int, int>, int)
  return WR(1, k, A, H)
```

Complexity analysis:

$$T(n, \lfloor \tfrac{n}{2} \rfloor + 1) = T(n-1, \lfloor \tfrac{n}{2} \rfloor) + T(n-1, \lfloor \tfrac{n}{2} \rfloor + 1)$$

$$= T(n-2, \lfloor \tfrac{n}{2} \rfloor - 1) + T(n-2, \lfloor \tfrac{n}{2} \rfloor) + T(n-2, \lfloor \tfrac{n}{2} \rfloor) + T(n-2, \lfloor \tfrac{n}{2} \rfloor + 1)$$

$$= T(n-2, \lfloor \tfrac{n}{2} \rfloor - 1) + T(n-2, \lfloor \tfrac{n}{2} \rfloor) + \quad 1 \quad + T(n-2, \lfloor \tfrac{n}{2} \rfloor + 1)$$

$$\vdots$$

As we can see, since we store the probabity of $P(i,k)$, we only require $O(1)$ when the same resursive called was resolved to. Thus, we need to calculate all the values once in the hash table, and that would be $O(n(\lfloor \frac{n}{2} \rfloor + 1)) = O(n^2)$. For the rest, we will have $O(1)$ to get the value in hash table. Therefore $T(n) \in O(n^2)$.