P1

(a) $f(n) = n^2 \log n$, $n^{\log_b a} = n^{\log_2 4} = n^2$

$T(n) = \Theta(n^2 \log^2 n)$

(b) $f(n) = n \log n$ $n^{\log_b a} = n^{\log_8 8}$

$f(n) = \Theta(n^{\log_8 8 - \varepsilon})$. $\varepsilon > 0$

$T(n) = \Theta(n^{\log_8 8})$

(c) $f(n) = n^{\sqrt{\log n}}$ $n^{\log_b a} = n^{\log_2 \sqrt{\log n}} = n^{6.275}$

$f(n) = \Omega(n^{6.275 + \varepsilon})$ $\varepsilon > 0$

$\sqrt{\log n} \cdot (\frac{n}{2})^{\sqrt{\log n}} \leq c n^{\sqrt{\log n}}$, $c < 1$

$T(n) = \Theta(n^{\sqrt{\log n}})$

(d) $f(n) = 2^n$ $n^{\log_b a} = n^{\log_2 n}$

$f(n) = \Omega(n^{\log_2 n + \varepsilon})$ $\varepsilon > 0$

$T(n) = \Theta(2^n)$

(e) let $n = 2^m$, $T(2^m) = 2T(2^{\frac{m}{2}}) + m$

$\therefore U(m) = 2T(\frac{m}{2}) + m$

$f(m) = m \quad m^{\log_2^2} = m$

$\therefore U(m) = \Theta(m \log m)$

$\therefore T(n) = \Theta(\log_2 n \log\log_2 n)$

P2

Divide the n cards into 2 sets, both sets have $\frac{n}{2}$ cards, which named S_1, S_2. Recursively run the algorithm, if the size of set is 1, return the one card. if the size of set is 2, test whether 2 cards are equivalent. If they are equivalent, return either card. If one card is returned, test the card with other cards. Finally return a card from majority equivalence if one is found.

If there is a majority equivalence set for at least one of the two sides.

$T(n) = 2T(n/2) + n$

$T(n) = O(n \log n)$

P3

We sort lines in order of increasing slope. Divide n lines into 2 sets. Both sets have $\frac{n}{2}$ lines. Recursively run the algorithm. If $n \leq 3$, it is easy to find which line is visible. The minimum slope line and maximum slope line are always visible, just need to judge other lines Then merge sets and consider lines that are uppermost, return the uppermost lines to upper level until the algorithm complete.

$T(n) = O(n \log n)$

P4

Divide a into $\frac{a}{2}$, and recurisively run the algorithm. When a = 0, result is 1 let return subresult is y. Then if a is even, return result is y*y. If a is odd, return result is y*y*x. return the result to upper line until the algorithm complete.

$T(n) = 2T(n) + 1$

$T(n) = O(n)$

P5

If two strings have different length, they cannot be J-similar to each other, they cannot be divided into pieces are of equal size.

The algorithm is that if string a is equal to b, they are J-similar. If the length of string a and b is odd, they are not J-similar.

Divide string a into 2 pieces, named a_1, a_2.
Divide string b into 2 pieces, named b_1, b_2.
Recursively run the algorithm. If a_1 is J-similar to b_1, a_2 is J-similar to b_2 or a_1 is J-similar to b_2, a_2 is similar to a_1, return true.

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

$T(n) = O(n \log n)$

P6

a) First check whether middle element is fixed Point. If it is, return it. Otherwise if the index of middle +1 element is less than or equal to the value at the rightmost index, then Fixed Point might on the right side of the middle Point. Similarly, check if the index of middle -1 element is greater than or equal to the value at the leftmost index, then Fixed Point might on the left side of the middle point.

Repeat the operation until find fixed point or current leftmost index is larger than current rightmost index, it means there is no fixed point, return -1. Time complexity is $O(\log n)$

b) $T(n) = T(\frac{n}{2}) + 1$

$f(n) = 1 \quad n^{\log_b^a} = n^{\log_2^1} = 1$

$f(n) = \Theta(1)$

$\therefore T(n) = \Theta(\log n)$

c) If the index of P is i, then just check i-1 element and i+1 element. If they are not fixed point, there are no other fixed point in the array. Time complexity is $O(1)$