No
DATE

Q1 (a) $a = 4$, $b = 2$, $c = \log_2 4 = 2$, $f(n) = n^2 \log(n)$

$\therefore f(n) = \Theta(n^2 \log n)$ $\therefore k = 1$, then $T(n) = \Theta(n^2 \log^2 n)$

(b) $a = 8$, $b = 6$, $c = \log_b a = \log_6 8$, $f(n) = n \log n$

$\because O(n^{\log_6 8}) > O(n \log n)$ $\therefore f(n) = O(n^{\log_6 8 - \varepsilon})$

then $T(n) = \Theta(n^{\log_6 8})$ ; $0 < \varepsilon < \log_6 8 - 1$

(c) $a = \sqrt{6006}$, $b = 2$, $c = \log_b a = \log_2 \sqrt{6006} > 1$, $f(n) = n^{\sqrt{6006}}$

$\therefore f(n) = \Omega(n^{\log_2 \sqrt{6006} + \varepsilon})$ then $T(n) = \Theta(n^{\sqrt{6006}})$, $\varepsilon > 0$

(d) $a = 10$, $b = 2$, $c = \log_b a = \log_2 10 > 1$, $f(n) = 2^n$ $\therefore f(n) = \Omega(n^{\log_2 10 + \varepsilon})$

the $T(n) = \Theta(2^n)$, $\varepsilon > 0$

(e) Assume $n = 2^m$, then $T(2^m) = 2 \cdot T(2^{m/2}) + m$.

Assume $T(2^m) = F(m)$, then $F(m) = 2F(m/2) + m$

$a = 2$, $b = 2$, $\log_b a = 1$, $f(m) = m$.

then $f(m) = \Theta(m)$, $k = 0$. $\Rightarrow F(m) = \Theta(m \log_2 m)$

$\Rightarrow T(2^m) = \Theta(2^m \log_2 2^m) = \Theta(m \cdot 2^m)$

$\Rightarrow T(n) = \Theta(n \log_2 n)$

- $T(n) = T(n/2) - n + 10$

  $a = 1$, $b = 2$, $c = \log_b a = \log_2 1 = 0$, $f(n) = 10 - n$, $0 < n \le 10$

  then $f(n) = O(1)$, $c = 0$, $k = 0$

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

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

  $a = 2^n$, $b = 2$, $c = \log_b a = \log_2 2^n = n$. , $f(n) = n$

  $f(n) = O(n^{n-\varepsilon})$, then $T(n) = \Theta(n^n)$ for some $\varepsilon > 0$

- $T(n) = 2T(n/4) + n^{0.51}$

  $a = 2$, $b = 4$, $c = \log_b a = \log_4 2 < 1$, $f(n) = n^{0.51}$

  $\log_4 2 = \log_4 4^{\frac{1}{2}} = 0.5 < 0.51$ $\therefore f(n) = \Omega(n^{\log_4 2 - \varepsilon})$

  then $T(n) = \Theta(n^{\log_4 2}) = \Theta(\sqrt{n})$ for some $\varepsilon > 0$

No.
DATE / /

- $T(n) = 0.5\,T(n/2) + 1/n$

$a = 0.5$, $b = 2$. $c = \log_b a = \log_2 0.5 = \log_2 \frac{1}{2} = \log_2 2^{-1} = -1$

$f(n) = \frac{1}{n} = n^{-1}$, then $f(n) = \Theta(n^{-1})$, $k = 0$, then $T(n) = \Theta(n^{-1} \log n)$

- $T(n) = 16\,T(n/4) + n!$

$a = 16$, $b = 4$, $c = \log_b a = \log_4 16 = 2$, $f(n) = n!$

$\therefore f(n) = \Omega(n^{2+\varepsilon})$, $T(n) = \Theta(n!)$

Q2. We use induction to prove there always exist a local minimum.

For $n=3$, we have $A_1 \geq A_2$ and $A_3 \geq A_2$, thus $A_2$ is a local minimum.

Let $A_1 A_2 \dots A_{n-1} A_n$ exist a local minimum

Then we have to prove that $A_1 A_2 \dots A_{n-1} A_n A_{n+1}$ also exist a minimum.

Assume $A_2 > A_3$, in this case, assume $A_2, A_3 \dots A_n, A_{n+1}$ equal to $A'_1, A'_2, \dots, A'_{n-1}, A'_n$. also satisfy the induction hypothesis

$$\begin{cases} A_2 = A'_1 \\ A_3 = A'_2 \\ A'_1 \geq A'_2 \end{cases} \Rightarrow A_2 \geq A_3 \qquad \begin{cases} A_n = A'_{n-1} \\ A_{n+1} = A'_n \\ A'_{n-1} \geq A'_n \end{cases} \Rightarrow A_n \geq A_{n+1}$$

Thus, $A_2, A_3 \dots A_n, A_{n+1}$ satisfied, and $A_1, A_2 \dots A_{n-1}, A_n$ satisfied by induction hypothesis, Therefore $A_1, A_2 \dots A_n, A_{n+1}$ always exists a local minimum

Let's take length $n$ of $A$ as input, the minimum element as output

Persudo Code:

If $n=3$:
    return $A_2$
If $n>3$:
    $i = n/2$
    If $A_{i-1} \geq A_i$ & $A_{i+1} \geq A_i$: ------ ①
        return $A_i$
    If $A_i > A_{i-1}$ ------ ②
        do recursive algorithm on $A_1$ to $A_i$
    else if $A_i > A_{i+1}$
        do recursive algorithm on $A_i$ to $A_n$

Runtime Complexity: Assume runtime based on length $n$.

when $n=3$, runtime is $O(1)$

when $n>3$, we have to divide the array become 2 subarrays

In case ①, runtime is $O(1)$. In case ②, we have to recursive $A_1$ to $A_i$ or $A_i$ to $A_n$, then runtime is $T(n/2)$

No.

DATE / /

Therefore, $T(n) = T(n/2) + O(1) + O(1)$

According to the Master Theorem, $a=1$, $b=2$, $c = \log_b a = \log_2 1 = 0$

$f(n) = O(1) = $ constant, then $T(n) = \Theta(n^c \log^{k+1} n)$, $c=0, k=0$

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

Q5. we can use dynamic programming to solve this problem

1. Build a dp[] array whose length same as length of array a. and set all elements in dp[] equals to -1.
2. Assume $0 \le i < n$, set $i = 0$ at beginning, then we have to go through this array a and dp.

   $dp[i] = \max(a[i] + dp[i + a[i]], dp[i+1])$

   Compare the sum of current value plus the next value and the value of dp[i+1], then choose a bigger one

   then recursive this algorithm

Runtime complexity: $O(n)$

We go through Array a and Array dp at same time, runtime of them are both $O(n)$, $O(n) + O(n)$ is still $O(n)$

No
DATE

Q6. I want to use contradiction to prove this
Assume that length of a $\neq$ length of b
Hypothesis: string a and b are J-similar to each other in one of these 2 cases.
In Case 1: a is equal to b, which is contradict to our hypothesis.
In Case 2: In (a). $a_1$ J-similar to $b_1$ and $a_2$ J-similar $b_2$
$a_1 = a_2$, $b_1 = b_2$, $\because a \neq b$ $\therefore a_1 \neq b_1$, $a_1 \neq b_2$, $a_2 \neq b_1$, $a_2 \neq b_2$
$\because$ $a_1$ is J-similar to $b_1$ and $a_1 \neq b_1$, so it's not suits case 1
then in case 2: cut $a_1$ into $a'_1$, $a'_2$, cut $b_1$ into $b'_1$, $b'_2$
$a'_1$ J-similar $b'_1$; $a'_2$ J-similar $b'_2$, then do resursion.
finally, $a^f_1 = a^f_2 = 1$, $b^f_1 = b^f_2 = 1$.
$a^f_1$ J-similar $b^f_1$ $a^f_2$ J-similar $b^f_2$
$a^f_1 = b^f_1 = 1$, it's contradiction.
Thus, only strings having the same length can be J-similar to each other.

Algorithm:
1. if length of a is not equal to length of b, return False
2. else if $|a| = |b| = 1$, return True
3. Divide a into $a_1$ and $a_2$ and length of $a_1$ = length of $a_2$
4. Divide b into $b_1$ and $b_2$ and length of $b_1$ = length of $b_2$
5. do recursive from step 1.

Time complexity: we have to divide string into 2 substring.
and then we have to handle both 2 substring.
then $T(n) = 2T(n/2) + O(n)$.
Thus $T(n) = \Theta(n \log n)$ by Master Theorem

Q7.

Assume the inverted array of array p is array i.

The infinite long super-array:

piip ippi ippi piip ......

We'll see that there are 2 array p and 2 array i out of every 4 subarrays. So we can think of the 4 subarrays as a whole, each subarray consists of 0's and 1's.

Let's calculate the sum of elements between a and b.

Algorithm:

① set start point at index a, then go through this array, find next meaningful index c when index % (4 * n) becomes 0.

② Do recursion, where set start point be c, and we can get next meaningful index d when index % (4 * n) becomes 0.

③ Go through index d to index b,

Runtime Complexity:

We do recursion in this algorithm, so runtime is $O(n)$

As we all know from question stem, $n << a << b$

thus $O(n) << O(b)$, the solved.