Reference:
Discuss with b08902066 and other classmates, TAs.
Introduction to Algorithm
https://en.wikipedia.org/wiki/Logarithmic_integral_function#Asymptotic_expansion
http://oldwww.ma.man.ac.uk/~mdc/MATH41022/Background/Background%20big%20O%20notation.pdf
https://stackoverflow.com/questions/5628260/how-to-solve-tn-tn-2-tn-4-tn-8-n

Problem 5

(1) a. Claim : $\sqrt{n} \neq O(n^{\sin n})$ and prove by contradiction.

<Proof>

1° Suppose $\sqrt{n} = O(n^{\sin n})$, $\exists c, n_0 > 0$, s.t. $\forall n > n_0 \Rightarrow 0 \leq \sqrt{n} = n^{\frac{1}{2}} \leq c \cdot n^{\sin n}$

since $n > n_0$, there is some $n > 1$ and $n > n_0$ s.t. $\frac{1}{2} \leq \sin n$

2° We know that $-1 \leq \sin n \leq 1$, since $n > n_0$ (also when $n > 1$)

there is some n s.t. $\frac{1}{2} > \sin n$, Contradiction !

3° $\because$ 1°, 2° Contradiction $\therefore$ we know that the hypothesis is incorrect

$\Rightarrow$ we know $\sqrt{n} \neq O(\sin^n)$, QED.

b. Claim : if $f(n) = \theta(g(n))$ then $\log f(n) = \theta(\log g(n))$

<Proof>

1° $\because f, g$ is unbounded , $f(n) = \theta(g(n))$ $\exists n_0, c_1, c_2 > 0$ s.t. $\forall n > n_0$

$\Rightarrow c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \rightarrow \log c_1 + \log g(n) \leq \log f(n) \leq \log c_2 + \log g(n)$

2° $\because f, g$ is increasing and unbounded , $\exists n_1, c_1', c_2'$ s.t. $\forall n > n_1$ $\left[c_2' \geq \frac{\log c_2}{\log g(n_1)} + 1 \right]$

$\Rightarrow 0 \leq \log f(n) \leq \log c_2 + \log g(n) \leq c_2' \log g(n) \Rightarrow \log f(n) = O(\log g(n))$

3° $0 \leq c_1' \log g(n) \leq \log c_1 + \log g(n) \leq \log f(n)$ $\left[0 < c_1' \leq \frac{\log c_2}{\log g(n_1)} + 1 \right]$

$\Rightarrow \log f(n) = \Omega(\log g(n))$

4° From 2°, 3° $\Rightarrow$ we know $\log f(n) = \theta(\log g(n))$

c. Claim: if $f_1(n) = O(g_1(n))$, and $f_2(n) = O(g_2(n))$, then "not" always $(f_1 \circ f_2)(n) = O(g_1 \circ g_2(n))$

<Proof>

1° To prove the statement incorrect, give a counter-example.

Suppose $f_1(n) = g_1(n) = 2^n$, $f_1(n) = O(g_1(n))$ holds $(\frac{1}{2} \cdot g_1(n) \le f_1(n) \le 2 \cdot g_1(n))$

Suppose $f_2(n) = 2n$, $g_2(n) = n$, $f_2(n) = O(g_2(n))$ holds $(g_2(n) \le f_2(n) \le 3 \cdot g_2(n))$

2° So $f_1 \circ f_2(n) = 2^{2n} = 4^n$, $g_1 \circ g_2(n) = 2^n$ $\Rightarrow^{*} 4^n \neq O(2^n)$

$\Rightarrow$ so $f_1 \circ f_2(n) \neq O(g_1 \circ g_2(n))$ for this case.

$\therefore$ if $f_1(n) = O(g_1(n))$, & $f_2(n) = O(g_2(n))$, $(f_1 \circ f_2)(n) = O(g_1 \circ g_2(n))$ doesn't always hold.

* [prove $4^n \neq O(2^n)$ → Suppose $4^n = O(2^n) \Rightarrow \exists c, n_0 > 0$ s.t. $4^n < c \cdot 2^n$
$\Rightarrow$ devide 2^n : $2^n < c$, but theres no c, n_0 holds → $4^n \neq O(2^n)$]

d. Claim: $(n+a)^b = \theta(n^b)$, where a,b are real constant with $b > 0$

<Proof>

1° $(n+a)^b = C_b^b \cdot n^b + C_{b-1}^b n^{b-1} \cdot a + \cdots + C_0^b \cdot a^b = \sum_{k=0}^{b} n^k \cdot a^{(b-k)}$

Suppose $C > 0$ and $C = c_0 + c_1 + \cdots + c_b$, where for $0 \le i \le b$, $a^i \le c_i$

2° So when $n > n_0 = 1$ and $0 \le i \le b$, $c_i \ge a^i \Rightarrow c_i \cdot n^b \ge c_i \cdot n^{(b-i)} \ge a^i \cdot n^{(b-i)}$

$\Rightarrow (n+a)^b = \sum_{k=0}^{b} n^k \cdot a^{(b-k)} \le \sum_{k=0}^{b} c_k \cdot n^b = c \cdot n^b$ for $n > n_0 = 1$

$\Rightarrow$ thus $(n+a)^b \le c \cdot n^b$ — ①

3° Same as above, there exist $c' = c_0' + c_1' + \cdots + c_b'$ s.t. for $0 \le i \le b$, $a^i \ge c_i'$

$\Rightarrow (n+a)^b = \sum_{k=0}^{b} n^k \cdot a^{(b-k)} \ge \sum_{k=0}^{b} c_k' \cdot n^b = c \cdot n^b$ for $n > n_0 = 1$

$\Rightarrow$ thus $(n+a)^b \ge c' \cdot n^b$ — ②

4° from ①、② $\Rightarrow$ $c' \cdot n^b \le (n+a)^b \le c \cdot n^b$ for some c、c'、n_0

thus, $(n+a)^b = \theta(n^b)$. QED

(2) a. Claim: $T(n) = \theta(\frac{n}{\log n})$

<Proof>

1° By Recursion-Tree Method. Set $n - 127 \cdot k \le 2 \Rightarrow k \ge \frac{n-2}{127}$

$$\Rightarrow T(n) = T(n-127) + \frac{127}{\log(n)} = T(n-127\times 2) + \frac{127}{\log(n-127)} + \frac{127}{\log(n)}$$

$$= T(n-127\cdot k \le 2) + \frac{127}{\log(n-127k+127)} + \cdots + \frac{127}{\log(n-127)} + \frac{127}{\log(n)}$$

$$\Rightarrow T(n) = 1 + \sum_{i=0}^{k-1} \frac{127}{\log(n-127i)} = 1 + 127 \cdot \sum_{i=0}^{k-1} \frac{1}{\log(n-127i)} \ge 1 + 127 \cdot k \cdot \frac{1}{\log(n)}$$

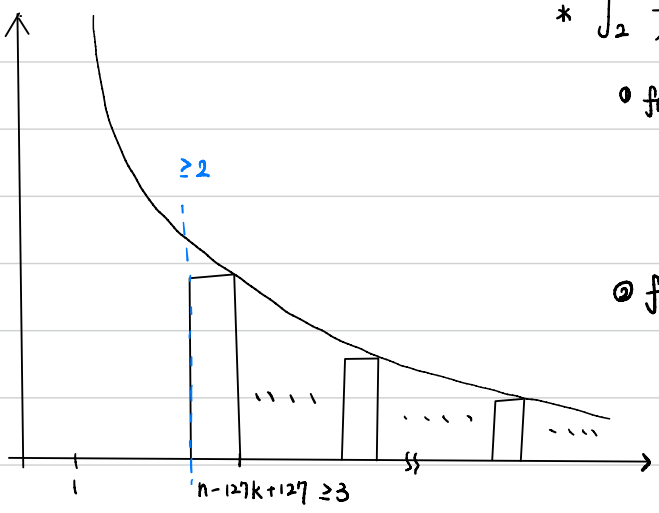
$$\ge 1 + 127 \cdot (\frac{n-2}{127}) \cdot \frac{1}{\log(n)} = 1 + \frac{n-2}{\log(n)} = \Omega(1) + \Omega(\frac{n}{\log(n)}) = \Omega(\frac{n}{\log(n)})$$

2° $T(n) = 1 + \sum_{i=0}^{k-1} \frac{127}{\log(n-127i)} \le 1 + 127\int_2^n \frac{dx}{\log(x)} = 1 + 127\left[\int_2^{\sqrt{n}} \frac{dx}{\log(x)} + \int_{\sqrt{n}}^n \frac{dx}{\log(x)}\right]$

$$\overset{*}{\le} 1 + 127 \cdot \left[\frac{(\sqrt{n}-2)}{\log 2} + \frac{n-\sqrt{n}}{\log\sqrt{n}}\right] = O(1) + O(\sqrt{n}) + O(\frac{n}{\log n}) = O(\frac{n}{\log n})$$

3° from 1°, 2°, We known $T(n) = O(\frac{n}{\log n})$ & $T(n) = \Omega(\frac{n}{\log n})$

$\therefore T(n) = \theta(\frac{n}{\log n})$, QED.



$$* \int_2^n \frac{dx}{\log x} = \int_2^{\sqrt{n}} \frac{dx}{\log x} + \int_{\sqrt{n}}^n \frac{dx}{\log x} = O(\frac{n}{\log n})$$

① for $x \in [2, \sqrt{n}]$, $\log x \ge \log 2 \Rightarrow \frac{1}{\log x} \le \frac{1}{\log 2}$

$$\rightarrow \int_2^{\sqrt{n}} \frac{dx}{\log x} \le \frac{\sqrt{n}-2}{\log 2} = O(\sqrt{n})$$

② for $x \in [\sqrt{n}, n]$, $\log x \ge \log\sqrt{n} \Rightarrow \frac{1}{\log x} \le \frac{1}{\log\sqrt{n}} = 2\frac{1}{\log n}$

$$\rightarrow \int_{\sqrt{n}}^n \frac{dx}{\log x} \le 2 \cdot \frac{n-\sqrt{n}}{\log n} = O(\frac{n}{\log n})$$

b. Claim : $T(n) = \Theta(n\log n)$

<Proof>

1° We can easily know that $T(\frac{n}{2}) > T(\frac{n}{4})$ & $T(\frac{n}{3}) > T(\frac{n}{8})$

$\Rightarrow T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + T(\frac{n}{8}) + n\log n > 3\cdot T(\frac{n}{8}) + n\log n$

∴ From Master Theorem, we know for $3\cdot T(\frac{n}{8}) + n\log n$

① $f(n) = n\log n = \Omega(n^{\log_8 3 + \epsilon}) = \Omega(n^{0.528\sim + \epsilon})$ for some $\epsilon > 0$

② $3\cdot f(\frac{n}{8}) = 3\cdot(\frac{n}{8})\cdot\log(\frac{n}{8}) = \frac{3}{8}\cdot n\cdot\log(\frac{n}{8}) \le \frac{3}{8}\cdot n\cdot\log(n)$ for $c = \frac{3}{8} < 1$

$\Rightarrow 3\cdot T(\frac{n}{8}) + n\log n = \Theta(n\log n) \Rightarrow T(n) = \Omega(n\log n)$

2° Suppose there's a, b s.t. $\forall n > 2$, $T(n) \le a\cdot n\log n + bn$

By Induction, find a, b

① $n \le 2$, trivial

② $n > 2$, $T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + T(\frac{n}{8}) + n\log n$ (Inductive hypothesis)

$\le a\cdot(\frac{n}{2})\log(\frac{n}{2}) + b(\frac{n}{2}) + a\cdot(\frac{n}{4})\log(\frac{n}{4}) + b(\frac{n}{4})$

$+ a(\frac{n}{8})\log(\frac{n}{8}) + b(\frac{n}{8}) + n\log n$

$= (\frac{7}{8}a + 1)\, n\log n + (\frac{7}{8}b - \frac{11}{8}a\log 2)\, n$

$\le a\cdot n\log n + bn$ $\quad \frac{1}{8}b = -11\log 2$

∴ $T(n) \le a\cdot n\log n + bn$ holds, when $a = 8$, $b = -88\log 2$

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

3° From 1°, 2°, We know $T(n) = \Omega(n\log n)$ & $T(n) = O(n\log n)$

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

c. Claim: $T(n) = \Theta(n^2)$

<Proof>

1° From Master Theorem, $f(n) = n\log(n) = O(n^{\log_2 4 - \epsilon}) = O(n^{2-\epsilon})$, $\epsilon = 0.5 > 0$

$$\left[\lim_{n\to\infty} \frac{n\log n}{n^{2-\epsilon}} \xrightarrow{\text{take } \epsilon = 0.5} \lim_{n\to\infty} \frac{\log n}{\sqrt{n}} \xrightarrow{\text{L'Hospital}} \lim_{n\to\infty} \frac{\frac{1}{x}}{\frac{1}{2}\cdot\frac{1}{\sqrt{x}}} = 0 \text{, } n^{1.5} \text{ grows faster} \right]$$

$$\Rightarrow T(n) = \Theta(n^{\log_2 4}) = \Theta(n^2)$$

d. Claim: $T(n) = \Theta(n\log\log n)$

<Proof>

1° Set $n = e^k \Rightarrow T(e^k) = e^{\frac{k}{2}} T(e^{\frac{k}{2}}) + e^k \xrightarrow{\text{divide by } e^k} \frac{T(e^k)}{e^k} = \frac{T(e^{\frac{k}{2}})}{e^{\frac{k}{2}}} + 1$

Set $\frac{T(e^k)}{e^k} = S(k) \Rightarrow \frac{T(e^k)}{e^k} = \frac{T(e^{\frac{k}{2}})}{e^{\frac{k}{2}}} + 1 = S(k) = S(\frac{k}{2}) + 1$

2° From Master Theorem, $f(k) = 1 = \Theta(k^{\log_2 1}) = \Theta(k^0) = \Theta(1)$

$$\Rightarrow S(k) = \Theta(n^{\log_2 1}\cdot\log k) \Rightarrow T(n) = e^k \cdot \Theta(\log k) = \Theta(e^k \cdot \log k) = \Theta(n\log\log n)$$

Problem 6.

(1) 1° Construct a $n \times n$ dp map to calculate the sum of weight of a rectangle whose upper-left point is (0, 0) and lower-right is current (x, y)

$$\Rightarrow \text{accumulate}(x,y) = \begin{cases} \text{weight of } (0,0) & , x,y = 0 \\ \text{weight of } (x,y) + \text{accumulate}(x-1,y) & , x \neq 0 \ , y = 0 \\ \text{weight of } (x,y) + \text{accumulate}(x,y-1) & , x = 0 \ , y \neq 0 \\ \text{weight of } (x,y) + \text{accumulate}(x-1,y) + \text{accumulate}(x,y-1) - \text{accumulate}(x-1,y-1) \end{cases}$$

⇒ the addition will be O(1) but the construct of the dp map is $O(n^2)$

2° To obtain a rectangle with upper-left (x_1, y_1), lower-right (x_2, y_2), its weight equals

⇒ accumulate (x_2, y_2) - accumulate (x_1-1, y_2) - accumulate (x_2, y_1-1) + accumulate (x_1-1, y_1-1)

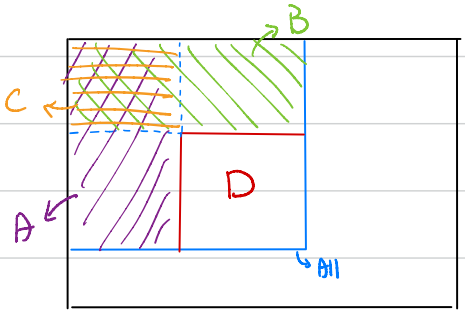
= rectangle (x_1, y_1, x_2, y_2)

3° At last, get the weight of perimeter by the rectangle (x_1, y_1, x_2, y_2) function

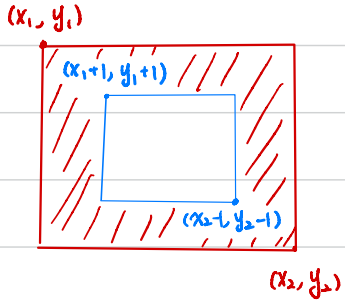
$$\Rightarrow \text{weight}(x_1,y_1,x_2,y_2) = \begin{cases} \text{rectangle}(x_1,y_1,x_2,y_2) & , x_2 - x_1 \leq 1 \ \vee \ y_2 - y_1 \leq 1 \\ \text{rectangle}(x_1,y_1,x_2,y_2) - \text{rectangle}(x_1+1, y_1+1, x_2-1, y_2-1) \end{cases}$$

⇒ the addition will be O(1) but calculate k points is O(k)

4° From 1°, 3° we know, it takes $O(n^2 + k)$



D = All - A - B + C



(2) 1° Construct a $n \times n$ dp map to calculate the sum of weight of a rectangle whose upper-left point is $(0, 0)$ and lower-right is current (x, y)

$$\Rightarrow \text{accumulate}(x,y) = \begin{cases} \text{weight of } (0,0), & x, y = 0 \\ \text{weight of } (x,y) + \text{accumulate}(x-1, y), & x \neq 0,\ y = 0 \\ \text{weight of } (x,y) + \text{accumulate}(x, y-1), & x = 0,\ y \neq 0 \\ \text{weight of } (x,y) + \text{accumulate}(x-1, y) + \text{accumulate}(x, y-1) - \text{accumulate}(x-1, y-1) \end{cases}$$

⇒ the addition will be $O(1)$ but the construct of the dp map is $O(n^2)$

2° At the same time, get all weight of rectangle with upper-left $(x-\Delta x, y-\Delta y)$ and lower-right (x, y) where $2(\Delta x + \Delta y) \leq L$ and rectangle $(x-\Delta x, y-\Delta y, x, y)$

= accumulate (x, y) - accumulate $(x-\Delta x-1, y)$ - accumulate $(x, y-\Delta y-1)$ + accumulate $(x-\Delta x-1, y-\Delta y-1)$

3° At last, get the weight of perimeter by the rectangle $(x-\Delta x, y-\Delta y, x y)$ function

$$\Rightarrow \text{weight}(x-\Delta x, y-\Delta y, x, y) = \begin{cases} \text{rectangle}(x-\Delta x, y-\Delta y, x, y), & \Delta x \leq 1,\ \Delta y \leq 1 \\ \text{rectangle}(x-\Delta x, y-\Delta y, x, y) - \text{rectangle}(x-\Delta x+1, y-\Delta y+1, x-1, y-1) \end{cases}$$

for every (x, y), we have to run through all $(\Delta x + \Delta y) \times 2 \leq L \leq 2n$ possible non-negative Δx, Δy ⇒ at most $H_{2n}^3 = C_n^{2n+2} = \frac{1}{3}(2n+2)(2n+1)$ ⇒ $O(n^2)$

4° For every operation in 1°, we have to perform 3° simultaneously $(n^2 \times \frac{1}{3}(2n+2)(2n+1))$

⇒ it takes $O(n^2 \times n^2) = O(n^4)$

(3) 1° Construct two $n \times n$ array, one row_sum and another col_sum recording the accumulate value of weight.

$$\text{row\_sum}[i][j] = \begin{cases} \text{weight of } (i,0), & \text{if } j = 0 \\ \text{row\_sum}[i][j-1] + \text{weight of } (i,j), & \text{otherwise} \end{cases}$$

$$\text{col\_sum}[i][j] = \begin{cases} \text{weight of } (0,j), & \text{if } i = 0 \\ \text{col\_sum}[i-1][j] + \text{weight of } (i,j), & \text{otherwise} \end{cases}$$

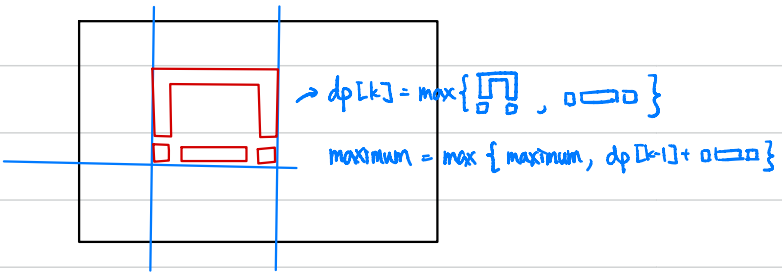
2° Use two pointers to go through all possible "column" combination with left column pointer from $0 \to n-2$ and right from $i+1 \to n-1$.

3° Also, create a $1 \times n$ dp array to record the largest possible if current left column pointer $= i$ & right $= j$. Also, maintain a row pointer k running from $0 \to n$, choose between maximum of ⊓ & ▭. and find value.

$$dp[k] = \begin{cases} \text{col\_sum}[0][j] - \text{col\_sum}[0][i-1], & \text{if } k = 0 \\ \text{MAX}\{\, dp[k-1] + \text{weight of } (k,i) + \text{weight of } (k,j), \\ \qquad \text{col\_sum}[k][j] - \text{col\_sum}[k][i-1] \,\} \end{cases}$$

$$\text{maximum} = \text{MAX}\{\text{maximum},\ dp[k-1] + \text{col\_sum}[k][j] - \text{col\_sum}[k][i-1]\}$$

4° So by traversing all i, j, k, and record the maximum simultaneously, we can find the answer in $O(n^3)$



(4) 1° Construct two $n \times n$ array, one row_sum and another col_sum recording the accumulate value of weight.

$$\text{row\_sum}[i][j] = \begin{cases} \text{weight of } (i,0), & \text{if } j = 0 \\ \text{row\_sum}[i][j-1] + \text{weight of } (i,j), & \text{otherwise} \end{cases}$$

$$\text{col\_sum}[i][j] = \begin{cases} \text{weight of } (0,j), & \text{if } i = 0 \\ \text{col\_sum}[i-1][j] + \text{weight of } (i,j), & \text{otherwise} \end{cases}$$

2° Use two pointers to go through all possible "column" combination with left column pointer I from $0 \to n-2$ and right J from $I+1 \to \frac{1}{2}L + I$

∴ for a rectangle whose top is U and bottom is D

$\Rightarrow$ weight of $\square IDJU$ = row_sum[U][I] − row_sum[U][J−1] +

row_sum[D][J] − row_sum[D][I−1] + col_sum[D][J] − col_sum[U−1][J]

+ col_sum[D][I] − col_sum[U−1][I].

= (row_sum[U][I] − row_sum[D][I−1] + col_sum[D][I] − col_sum[U−1][I])

+ (row_sum[D][J] − row_sum[U][J−1] + col_sum[D][J] − col_sum[U−1][J])

3° So we design a Monotone deque storing (value, index) =

(row_sum[U][I] − row_sum[D][I−1] + col_sum[D][I] − col_sum[U−1][I], U)

When we traverse k from $0 \to n-1$, we check and pop head from the monotone deque until k − deque.head.index $\leq \lfloor \frac{1}{2}L - (j-1) \rfloor$

4° Record maximum = MAX{maximum, deque.head.value + (row_sum[D][J]

− row_sum[U][J−1] + col_sum[D][J] − col_sum[U−1][J])}

5° And pop front until deque.head.value > (row_sum[U][I] - row_sum[D][I-1] + col_sum[D][I] - col_sum[U-1][I])

And push (row_sum[U][I] - row_sum[D][I-1] + col_sum[D][I] - col_sum[U-1][I]) into the deque. So after running all k's, we can find the maximum.

<Correctness>

Since we traverse all possible column (under restriction of perimeter ≤ L) and change the problem to find maximum in every sub-region and do it by preserving a monotone deque, we can make sure we get the answer since we go through all maximum combination possible.

<Time Complexity>

For the column pointer, we use n^2 to traverse all. During each traverse, we have a row pointer with n and deque with n. The time of addition, comparison and deque operation is $O(1)$

⇒ Total time complexity : $O(n^2 \cdot 2n) = O(n^3)$.