Homework 1-6 & 1-7 solution

Solution

Problem 6 (20 Points)

- 6-(1) 求逆序數對的演算法 (7 points)
  - 在merge sort的merge後多補上code:

```
// Count inversion pair between A[l~m] and A[m+1~r]
// A[l~m] and A[m+1~r] are sorted due to it's merged before by merge sort.
void counting(int l, int m, int r){
    int r_ptr = m+1;
    for(int i=l; i<=m; i++){
        while(r_ptr <= r && A[r_ptr] < A[i])
            r_ptr ++;
        ans += r_ptr - m - 1;
    }
}
```

- 6-(2) 證明逆序數對是O(NlogN) (5 points)
  - By the algorithm/merge sort above, the time complexity is $T(n) = 2T(\frac{n}{2}) + c \cdot n$, and $T(x) = 1, \forall x \leq 1$。
  - let $n' = 2^k \geq n, k \in N$, and the recursion formula with n' will be $T(2^k) = 2T(2^{k-1}) + c \cdot 2^k = 2(2T(2^{k-2}) + c \cdot 2^{k-1}) + c \cdot 2^k = 4T(2^{k-2}) + 2 \cdot c \cdot 2^k$
  - Expand recursion till k=0, $T(2^k) = 2^k T(1) + k \cdot c \cdot 2^k$. Therefore $T(n') = O(n' \log n')$。
  - Because T is non-decreasing function trivially, $T(n') \geq T(n)$; thus, $T(n) = n \log n, \forall n \in N$
  - Or you can just say by Master Theorem.
- 6-(3) 證明逆序數對=泡沫排序交換次數 (2 points)
  - We try to prove $I'(B) = I(B) - 1$ after swapping the adjacent inversion pair.
    - If $b_i > b_{i+1}$ and we want to swap b_i and b_{i+1}, we can part down the original $I(B)$ into small parts:
      - (A): Nothing with b_i, b_j part :$(x, y) \forall x < y, b_x > b_y, x, y \in [1, i-1] \cup [i+2, n]$
      - (B): Remaining part without $(i, i+1)$:
        - $(x, y) \forall b_x > b_y, x \in [1, i-1], y \in [i, i+1]$
        - $(x, y) \forall b_x > b_y, x \in [i, i+1], y \in [i+2, n]$
      - (C): $(i, i+1)$
    - After swapping, nothing change in (A) part and (B) part trivially. But (C) part will decrease 1.
      - This imply $I'(B) = I(B) - 1$ after swapping the adjacent inversion pair.
  - Bubble sort will swap all the adjacent inversion pair till it can't; thus the $I(B) = \text{the number of exchanges when performing bubble sort.}$
- 6-(4) 證明條件全滿的controllable ghost leg (2 points)

- First, we sort the array $X = [x_1, x_2 \ldots, x_n]$ such that constraints are [$(1, x_1)$,$(2, x_2)$...(n, x_n)]. (x, y) means that x should go to y after performed ghost leg and both x and y indicates which line you lie in.
- And the answer is $\text{Inversion-pair}(X)$. Next, we prove the correctness of this algorithm.

Problem	Supported operation	Target
Bubble Sort	If meets the adjacent inversion pair, swap it.	Repeat the swapping till no more move.
Full-constrained Ghost Leg	If there's any constraint that $x_i > x_{i+1}$, which means that i should go to x_i, $i+1$ should go to x_{i+1}, the horizontal line between $i, i+1$ must be added. And the new constraints about $i, i+1 \to (i, x_{i+1}), (i+1, x_i)$. (2) If it's not an inversion pair, it can't be swapped. Because it'll increase the inversion pair. The reason is mentioned in 6-3.	Repeat the operation left-mentioned till meets the constraints. (1)

> (1): If there's no adjacent inversion pair, that means there's nothing to be changed.
>
> - Note that it's impossible that having inversion pair without adjacent inversion pair, because only non-decreasing series can have no inversion pair.

> (2): Which is equivalent to bubble sort's array.

- In conclusion, the full-constrained controllable ghost legs problem (next we add horizontal line in where) is equivalent to the bubble sort's problem (next we swap what adjacent inversion pair). **So the minimum number of horizontal lines are equivalent to the number of exchanges when performing bubble sort, which is also equivalent to Inversion-pair of specified array.**

- 6-(5) 證明條件非全滿的controllable ghost leg (4 points)
  - First, we sort the array $X = [x_1, x_2 \ldots, x_n]$ such that constraints are [$(1, x_1)$,$(2, x_2)$...(n, x_n)]. If there's no constrained with x, we use $(x, ?_x)$ instead.
  - Fill the $?$ in the constraints from start to the end with the lowest number that not exist now in the constraints.
    - For example, if the constraints are $[(1, 3), (2, ?_2), (3, ?_3)]$, $?_2$ should be filled with 1 and $?_3$ should be filled with 2.
  - **After we filled up all the constraints, the problem is equivalent to 6-(4).** Next, we prove the correctness of this algorithm.
    - After substitution with $?$, the problem now is **how to fill the $?$ to archive the minimum inversion pair.** And now we prove why fill-lowest-number method is the only way to archive.
    - If there's an inversion pair in $?$ array, which means $\exists(j, k), j < k, ?_j > ?_k$. That's see the influence by these two $?$ and what will happen if these two swapped.
      - For all position i that $i < j < k$ or $j < k < i$ are ignored. Because it's trivially the same. (Same reason in 6-3).
      - For $j < i < k$,

Condition	inv pair if $?_j > ?_k$ (Original)	inv pair if $?_j < ?_k$ (Swapping)
x_i > both $?_j$ and $?_k$	|(i,k), (j,k)| = 2	|(i,k)| = 1
x_i < both $?_j$ and $?_k$	|(j,i), (j,k)| = 2	|(j,i)| = 1
x_i is in the middle	|(j,i), (i,k), (j,k)| = 3	|(j,i), (i,k)| = 0

- Thus, no matter what situation, $?$ array without any inversion pair is the best solution to minimize total array inversion pair. So $?$ must be **Non-decreasing array**, and fill-lowest-number method guarantee it's monotonic property.

Problem 7 (15 Points)

- 7-(1) 證明單方塊可以$O(1)$解 (2 points)
  - If there's k-length block in i position & the length of board is N,
    - if $N = k \cdot 2^j, j \in \text{non-negitive integers}$ and $\frac{(i-1)}{K}$ is non-negative integer , it can be solved.
    - otherwise, it can't be solved.
  - So if both $\log(\frac{N}{k})$ and $\frac{(i-1)}{K}$ are non-negative integer , the problem can be solved. and because we assume the log-operation is $O(1)$, so the time complexity of this solution to the problem is $O(1)$.
  - The proof of correctness is in 7-2.
- 7-(2) 證明$O(\log N)$的單方塊步驟 (1 point)
  - we define the problem $DC(k, i, N)$ = the solution step of k-length block in i position & the length of board is N.
    - if $k = N$, which means the problem is solved. return []
    - if $k \neq N$, which means it should be "unfold":
      - The length before it unfold to N should be $\frac{N}{2}$ trivially. (Every unfold will doubly increase the block), **which means if N and $N \neq K$, there's no solution. ... (1)**
      - The last-filled $\frac{N}{2}$ must in the left-most or right-most trivially. (Only unfold-to-left and unfold-to-right operations), **which means if the left space or right space are not the multiple of K or 0, there's no solution. Because we divide the problem into small problem, the property of multiple of K or 0 will not changed. ...(2)**
      - The last-filled boards cannot contain any piece of block. (no-overlapping)
      - So the problem
        $$DC(k,i,N) = \begin{cases} DC(k,i,\frac{N}{2}) + (["\ right\ "])\text{if } \frac{N}{2} \text{ right-most board is empty} \\ DC(k,i-\frac{N}{2},\frac{N}{2}) + (["\ left\ "])\text{if } \frac{N}{2} \text{ left-most board is empty} \end{cases}$$
  - Due to **(1), (2)**, we know both $\log(\frac{N}{k})$ and $\frac{(i-1)}{k}$ are non-negative integer , the problem can be solved.
- 7-(3) 證明unfold完狀態是$O(d_1 + d_2)$ (3 point)
  - if there's a k-length block, then it can only perform at most $\log_2(d_1 + d_2)$ times because unfold will doubly increase the block length. For simplicity, we let $d_1 + d_2 = N$.

        - After 0-time unfold, the possibilities of block status: $|\{[i, i+k-1]\}| = 2^0$
        - After 1-time unfold, the possibilities of block status: $|\{[i-k, i+k-1], [i, i+2k-1]\}| = 2^1$
        - After t-time unfold,the possibilities of block status: $|\{[i-(2^t-1)k, i+k-1], [i-(2^t-2)k, i+2k-1], \ldots, [i, i+2^t k-1]\}| = 2^t$. (Which Proved by 7-(1), if the size and position is good, then it can be solved/unfolded.)
    - And $\sum_x |x\text{-times unfold}| = 2^0 + 2^1 + 2^2 \ldots 2^t = 2 \cdot 2^t - 1$, thus the answer to the problem **at most** $2 \cdot 2^{log_2 N} - 1 = 2 \cdot N - 1 = 2 \cdot (d_1 + d_2) - 1 = O(d_1 + d_2)$.
- 7-(4) 寫出DP (5 points)
    - **DP state definition: DP[i] means is board[1,i] can be filled.**
    - **Initial state: DP[0] = true**
    - Transition:
        - Without loss of generality, we let B as the total blocks in the board. (sorted by position)

```
for now_B in B:
    for (x,y) in possibilities_of_unfold(now_B):
        if DP[x-1] is True:
            DP[y] = Ture
```

        - **And the answer is DP[N]**.
- 7-(5) 最佳子結構(1 point)，重複子問題(1 point)
    - Optimal substructure: It's trivial that we will choose the optimal solution (because we only use DP[x] if it is true (which means **the best answer of DP[x]**)).
    - Overlapping sub-problems: DP[x] may solved many times in brute force, but we use bottom-up method to let the sub-problems only be calculated once.
- 7-(6) 複雜度為$\Theta(N)$ (2 points)
    - $\Omega(N)$: Trivial. In worst case, we need to process whole the DP table. For example, 1-length block in $\frac{N}{2}$ position.
    - $O(N)$:
        - If there's n blocks, and we let the left space that Block_i can unfold is d_i, and right space that Block_i can unfold is d_{i+1}, the time complexity of DP algorithm will be $(d_1 + d_2) + (d_2 + d_3) + (d_3 + d_4) \ldots + (d_n + d_{n+1}) = 2\sum_{i=1}^{n+1} d_i - d_{n+1} - d_1$ due to the proof of 7-(3). **... (1)**
        - $\sum_{i=1}^{n+1} d_i \leq N$ trivially. **... (2)**
        - Because **(1)** and **(2)**, $2\sum_{i=1}^{n+1} d_i - d_{n+1} - d_1 \leq 2N$; thus, this DP algorithm is $O(N)$.
    - All in all, this DP algorithm is $\Theta(N)$.

Criterion (正在更新。)

基本上我看得懂你想表達什麼，並且沒有漏掉東西就會給對。

Problem 6 (20 points)

- 6-(1) 求逆序數隊的演算法 (7 points)
    - 沒寫算的過程(3 points)。
    - 錯的counting(3 points)。

- 二分搜(4 points)，因為複雜度是$O(n\log^2 n)$

- 6-(2) 證明逆序數隊是O(NlogN) (5 points)
  - 只有說mergesort是NlogN，所以inversion是NlogN (0.5 points)
  - 只寫T(n) = 2T(n/2) + n (2.5 points)
- 6-(3) 證明逆序數隊=泡沫排序交換次數 (2 points)
  - 只有說exchange是逆隊 (0.5 points)，
    - 因為必須考慮其他數字對交換的影響，以及交換後是剛好-1。否則證明不完全。
- 6-(4) 證明條件全滿的controllable ghost leg (2 points)
  - No Correctness (0 point)
- 6-(5) 證明條件非全滿的controllable ghost leg (4 points)
  - 直接把沒條件的拔掉當6-(4)算是錯的。反例：{(1->3), (3->1)}這樣算是1，但是因為中間多一個2要交換，所以導致整個逆對會變成2。如果是這樣的情況則0分。

Problem 7 (15 points)

- 7-(1) 證明單方塊可以O(1)解 (2 points)
  - 只寫1格 (-1 point)
  - 只寫$k \times 2^n$ (-0.5 point)
- 7-(2) 證明O(logN)的單方塊步驟 (1 point)
  - 複雜度寫歪 ($O(\log n)$寫成$O(n)$) (-0.5 point)
- 7-(3) 證明unfold完狀態是$O(d_1 + d_2)$ (3 point)
- 7-(4) 寫出DP (5 points)
  - Initial Statue (1 points)
  - Only DP definition (2 points)
  - Transition Function (2 points)
- 7-(5) 最佳子結構(1 point)，重複子問題(1 point)
- 7-(6) 複雜度為theta N (2 points)
  - Ω 0.5 points, O 1.5 points