J233 - Mechanical Grid

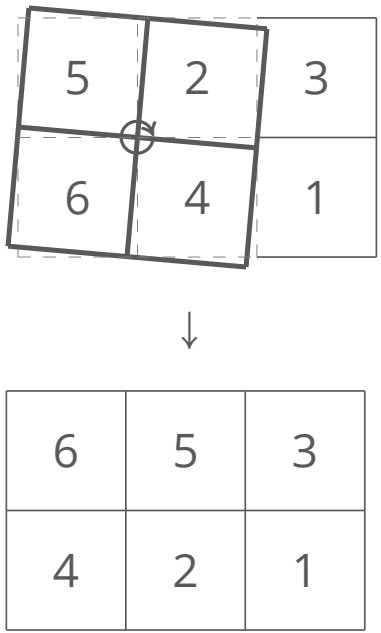
Benson Yeung {mtyeung1}
2023-02-04

Table of Contents

1 The Problem
2 Recap: Various Sorting Algorithms
3 Subtask 1: A small case
4 Subtask 2: Sorting a single row
5 Subtask 3: Sorting two rows

Background

Problem Idea by `mtyeung1`
Preparation by `yfng`, `kctung` and `cmli`
Figures by `microtony`
Presented by `mtyeung1`



(Extra) How to read a problem efficiently and correctly?

Skim through the statement to get a **brief idea of the problem**.

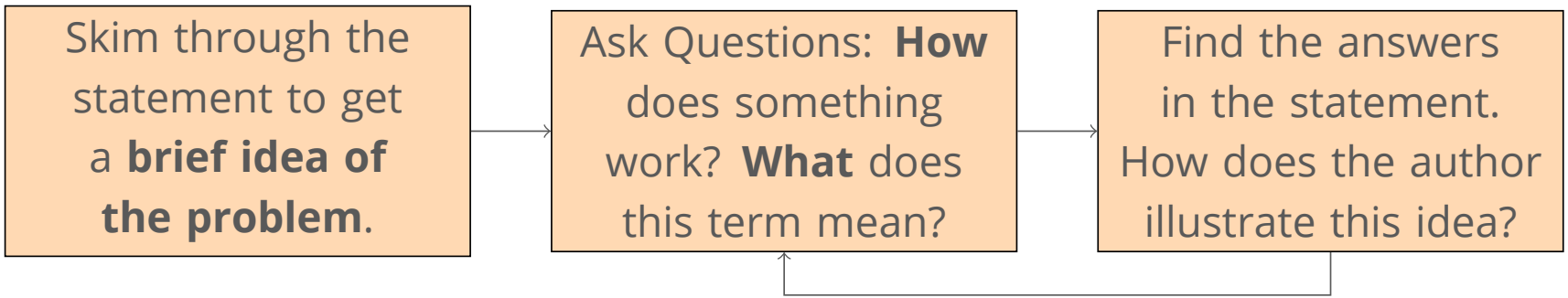
(Extra) How to read a problem efficiently and correctly?

Skim through the statement to get a **brief idea of the problem**.

→

Ask Questions: **How** does something work? **What** does this term mean?

(Extra) How to read a problem efficiently and correctly?

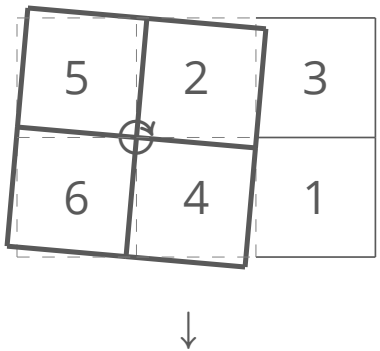


Problem Restatement

Given a $2 \times N$ grid with distinct numbers from 1 to $2N$, perform at most $\lfloor 2.5N \rfloor$ **rotation operations** to sort the grid.

A *rotation operation* is defined choosing a subgrid and rotate it clockwise until it perfectly fits back to the grid.

The grid is considered *sorted* if and only if each element is greater than the elements directly on the left or above it.



6	5	3
4	2	1

Statistics

0 points	33	+	16	+	2	+	0	= 51
9.1 points	2	+	0	+	0	+	0	= 2
13 points	9	+	4	+	0	+	0	= 13
30.55 points	0	+	2	+	0	+	0	= 2
52 points	0	+	1	+	2	+	0	= 3
52.15 points	0	+	0	+	1	+	0	= 1
76 points	0	+	0	+	5	+	0	= 5
85.6 points	0	+	1	+	6	+	2	= 9
100 points	0	+	0	+	0	+	6	= 6

First solved by **shayeung** at **73m 54s**
Second solved by **s20192** at **74m 27s**

Subtasks and Scoring

Partial Scoring:

Operations	$[0, \lfloor 2.5N \rfloor]$	$(\lfloor 2.5N \rfloor, 3N]$	$(3N, 6N]$	$(6N, 10^5]$
Score (%)	100	70	50	45

Subtasks:

	Points	**Constraints**
1	13	$N = 2$
2	39	Second row contains $N+1, N+2, \ldots, 2N$
3	48	No additional constraints

Table of Contents

1 The Problem

2 Recap: Various Sorting Algorithms

3 Subtask 1: A small case

4 Subtask 2: Sorting a single row

5 Subtask 3: Sorting two rows

Various Sorting Algorithms

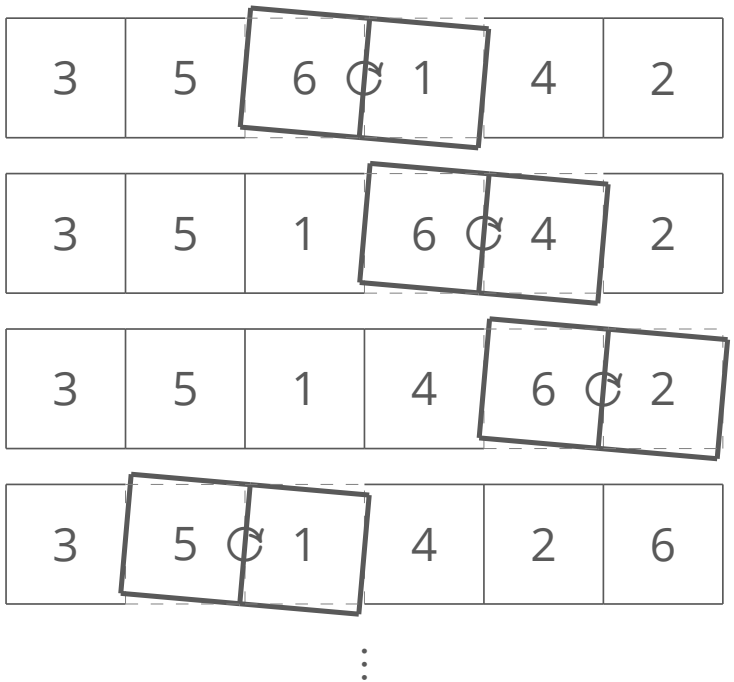
- There are many (standard) ways to sort an array.
- Although it is possible to "invent" a sorting algorithm by yourself during contest, it is always useful to learn the idea of these algorithms beforehand.
- They will be covered in HKOI Intermediate Training - **Sorting and Searching** (in online video format this year).

Comparison Based Sorting (Slow)	Comparison Based Sorting (Fast)	Non-comparison Based Sorting
• Selection sort • Bubble sort • Insertion sort	• Shell Sort • Quick Sort • Merge Sort	• Counting Sort • Radix Sort

Bubble Sort

- The algorithm processes the array in $N - 1$ passes (rounds).
- For each pass,
  - the algorithm compares every two adjacent elements.
  - if the left one is greater than the right one, then it swaps the elements.

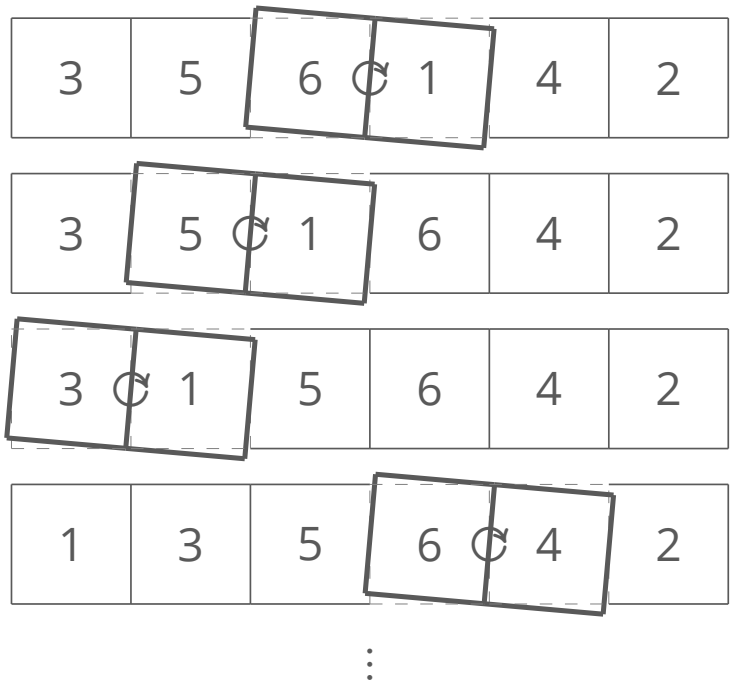
Time Complexity: $O(N^2)$



Insertion Sort

- The algorithm processes the elements from left to right.
- For each element x,
  - move all elements greater than x to the right by 1 position.
  - insert x into the vacated position.
- After processing the i-th element, the first i elements will be sorted.

Time Complexity: $O(N^2)$



Selection Sort

- The algorithm keeps selecting the smallest unprocessed element, and move it to the front.

Time Complexity: $O(N^2)$

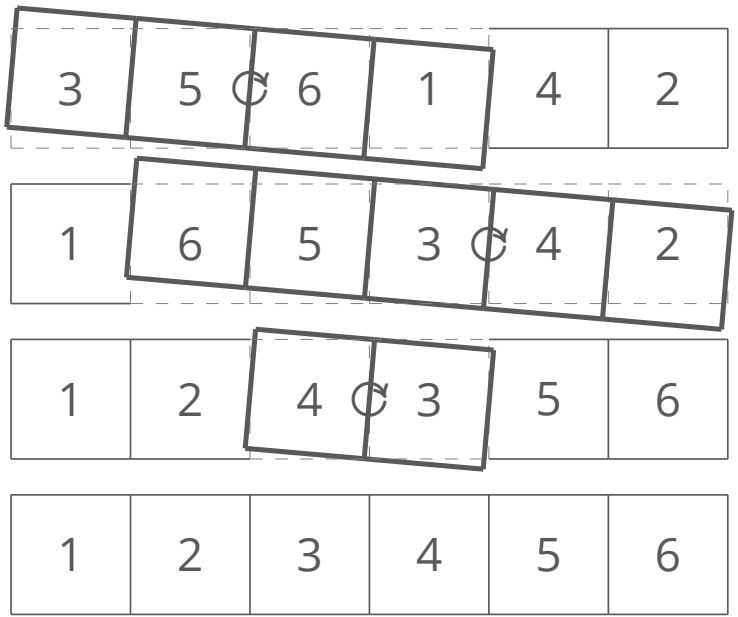


Table of Contents

1 The Problem

2 Recap: Various Sorting Algorithms

3 Subtask 1: A small case

4 Subtask 2: Sorting a single row

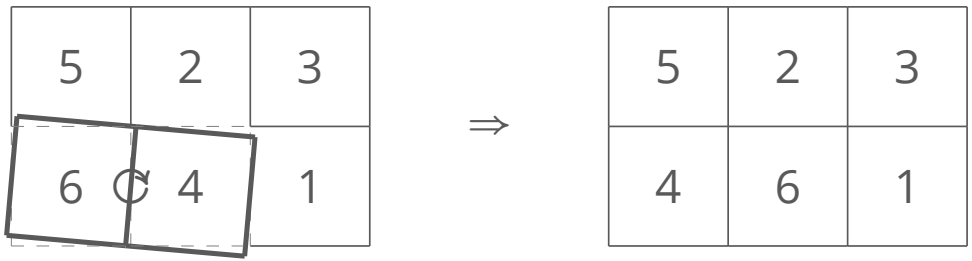
5 Subtask 3: Sorting two rows

The First Observation

Observation 1.1

If we apply the rotation operation on two adjacent cells, it is equivalent to **swapping** the values in the two cells.

Example:

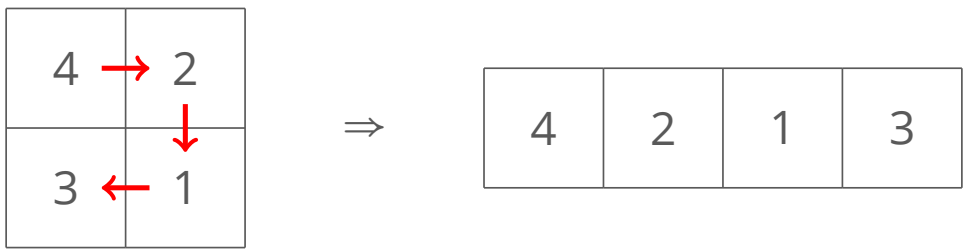


Subtask 1 Solution 1 (9.1 pts)

- The standard sorting algorithms could only be applied on an array, but now we have a 2×2 grid!

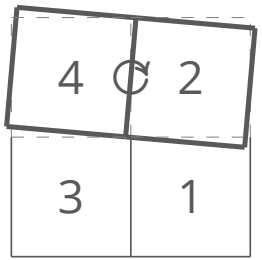
Subtask 1 Solution 1 (9.1 pts)

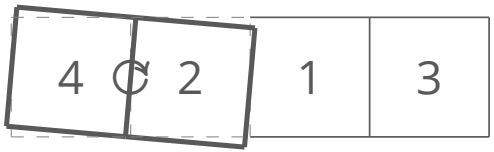
- The standard sorting algorithms could only be applied on an array, but now we have a 2×2 grid!
- Critical idea: Treat the grid as an array with turns.



Subtask 1 Solution 1 (9.1 pts)

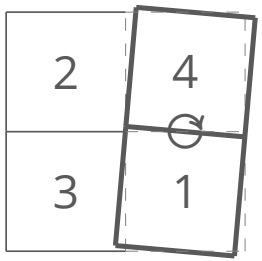
- Then, apply bubble sort (or insertion sort) to the array. (Note: Our goal is to sort the array to `1 2 4 3`, not `1 2 3 4`!)

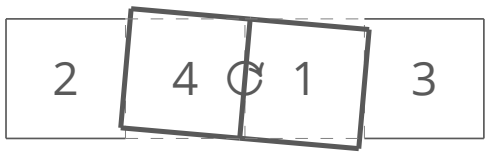




Subtask 1 Solution 1 (9.1 pts)

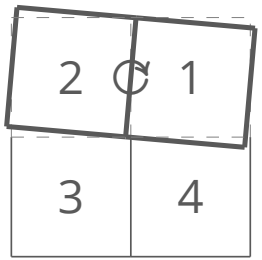
- Then, apply bubble sort (or insertion sort) to the array. (Note: Our goal is to sort the array to `1 2 4 3`, not `1 2 3 4`!)





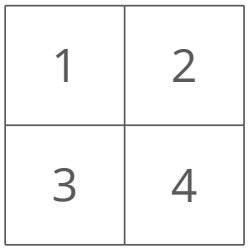
Subtask 1 Solution 1 (9.1 pts)

- Then, apply bubble sort (or insertion sort) to the array. (Note: Our goal is to sort the array to `1 2 4 3`, not `1 2 3 4`!)



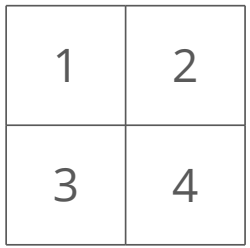
Subtask 1 Solution 1 (9.1 pts)

- Then, apply bubble sort (or insertion sort) to the array. (Note: Our goal is to sort the array to `1 2 4 3`, not `1 2 3 4`!)



Subtask 1 Solution 1 (9.1 pts)

- Then, apply bubble sort (or insertion sort) to the array. (Note: Our goal is to sort the array to `1 2 4 3`, not `1 2 3 4`!)



Number of rotations: 6
Expected score: 9.1

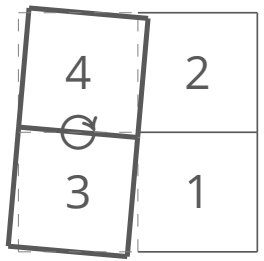
Subtask 1 Solution 2 (13 pts)

There exists an easier solution without the use of sorting algorithms.

Subtask 1 Solution 2 (13 pts)

There exists an easier solution without the use of sorting algorithms.

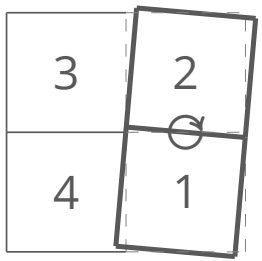
1. We first compare the values for each column. If the upper value is greater than the lower value, we swap the two values.



Subtask 1 Solution 2 (13 pts)

There exists an easier solution without the use of sorting algorithms.

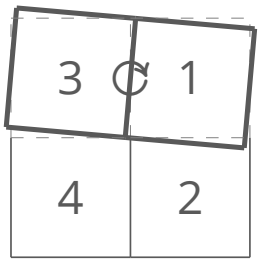
1. We first compare the values for each column. If the upper value is greater than the lower value, we swap the two values.



Subtask 1 Solution 2 (13 pts)

There exists an easier solution without the use of sorting algorithms.

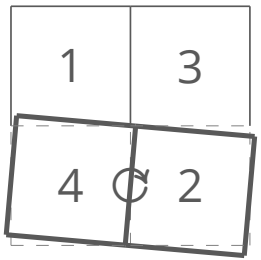
1. We first compare the values for each column. If the upper value is greater than the lower value, we swap the two values.
2. Then, we compare the values for each row. If the value on the left is greater than that on the right, we swap the two values.



Subtask 1 Solution 2 (13 pts)

There exists an easier solution without the use of sorting algorithms.

1. We first compare the values for each column. If the upper value is greater than the lower value, we swap the two values.
2. Then, we compare the values for each row. If the value on the left is greater than that on the right, we swap the two values.



Subtask 1 Solution 2 (13 pts)

There exists an easier solution without the use of sorting algorithms.

1. We first compare the values for each column. If the upper value is greater than the lower value, we swap the two values.
2. Then, we compare the values for each row. If the value on the left is greater than that on the right, we swap the two values.

1	3
2	4

Subtask 1 Solution 2 (13 pts)

There exists an easier solution without the use of sorting algorithms.

1. We first compare the values for each column. If the upper value is greater than the lower value, we swap the two values.
2. Then, we compare the values for each row. If the value on the left is greater than that on the right, we swap the two values.

1	3
2	4

Number of rotations: 4
Expected score: 13

Table of Contents

1 The Problem
2 Recap: Various Sorting Algorithms
3 Subtask 1: A small case
4 Subtask 2: Sorting a single row
5 Subtask 3: Sorting two rows

Subtask 2 (39 pts): $a_{2,i} = N + i$ for all $1 \le i \le N$

- Observe that in this subtask, the numbers on the second row are sorted, and they must be greater than the corresponding values on the first row.
- Therefore, our only objective is to sort the numbers $1, 2, \ldots, N$ on the first row.
- It will be better if we use rotation operations on the first row only. (otherwise, we would create a mess!)

Subtask 2 Solution 1 (17.55 pts)

- Apply bubble sort or insertion sort (introduced earlier) on the first row.
- Both sorting algorithms use at most $\frac{N(N-1)}{2} \leq 19900 < 10^5$ swaps.

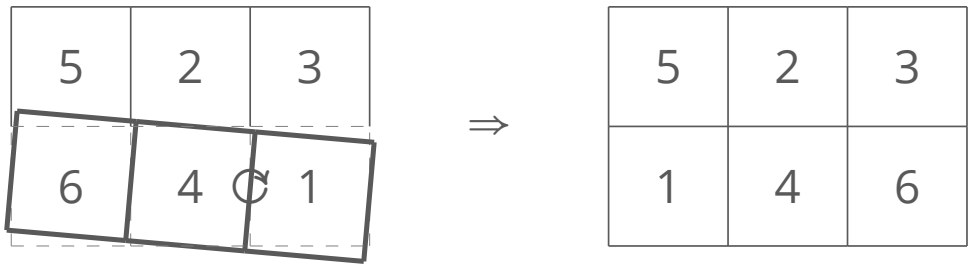
Nuumber of rotations: 19900
Expected score: 17.55

A Stronger Observation

Observation 1.2

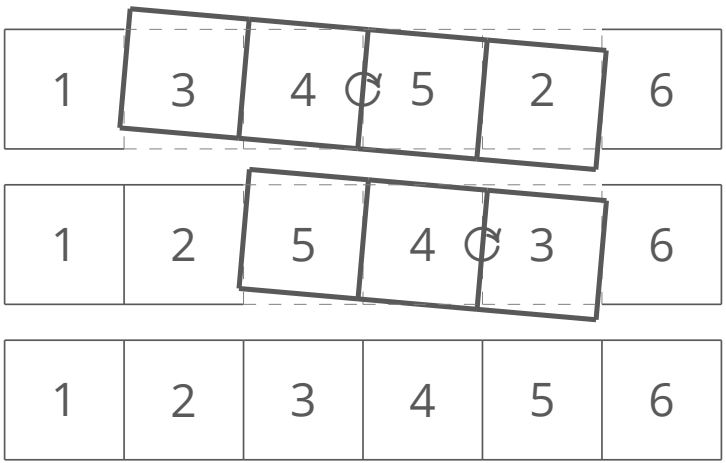
If we apply the rotation operation on a $1 \times k$ subgrid, it is equivalent to reversing the $1 \times k$ array.

Example:



Subtask 2 Solution 2 (38 pts)

Observe that in insertion sort, using N swaps to insert an element is not necessary. Instead, we can insert an element using 2 reverse operations.



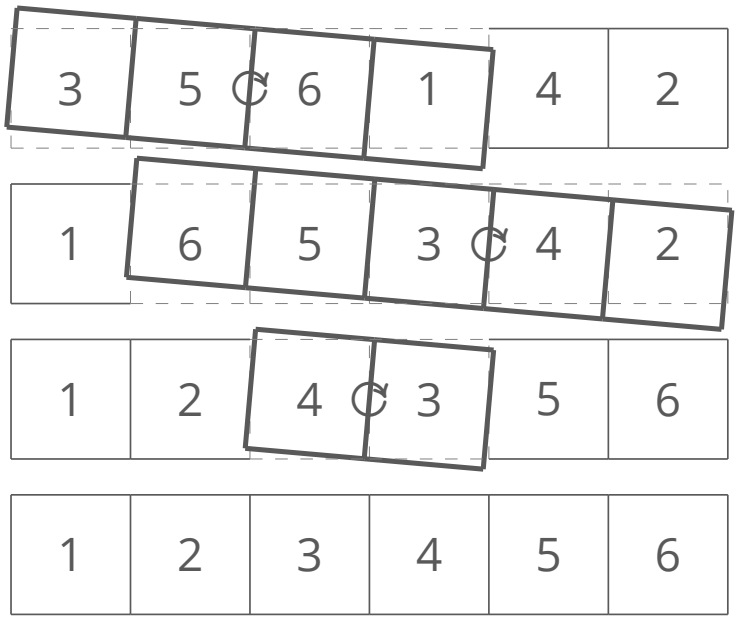
This reduces the total number of operations to $2N$.

Subtask 2 Solution 3 (38 pts)

- Insertion sort is still inefficient after the optimization.
- Why not try selection sort instead?

Subtask 2 Solution 3 (38 pts)

- Insertion sort is still inefficient after the optimization.
- Why not try selection sort instead?
- We keep finding the smallest element, and use 1 rotation operation to bring the element to its correct position. Note that we do not need to preserve the order of the remaining elements as they are still unprocessed.



Subtask 2 Solution 3 (38 pts)

- Insertion sort is still inefficient after the optimization.
- Why not try selection sort instead?
- We keep finding the smallest element, and use 1 rotation operation to bring the element to its correct position. Note that we do not need to preserve the order of the remaining elements as they are still unprocessed.

Number of rotations: N

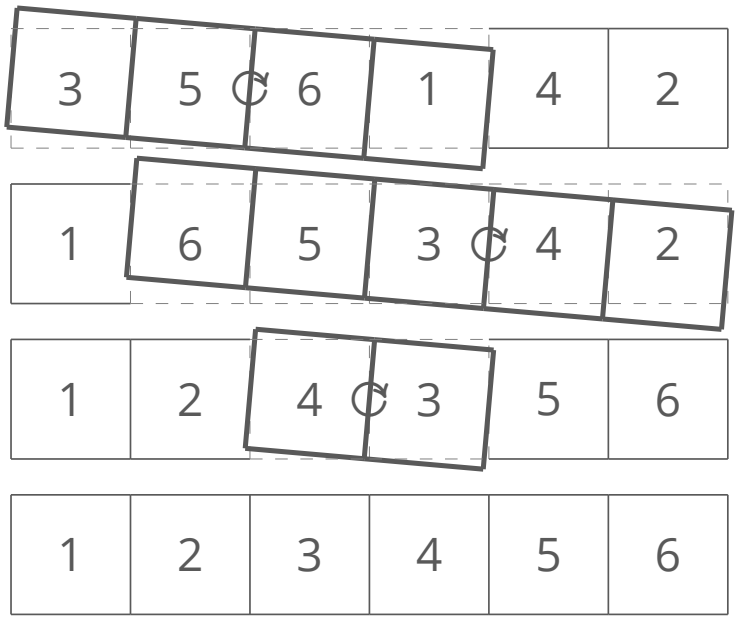


Table of Contents

1 The Problem

2 Recap: Various Sorting Algorithms

3 Subtask 1: A small case

4 Subtask 2: Sorting a single row

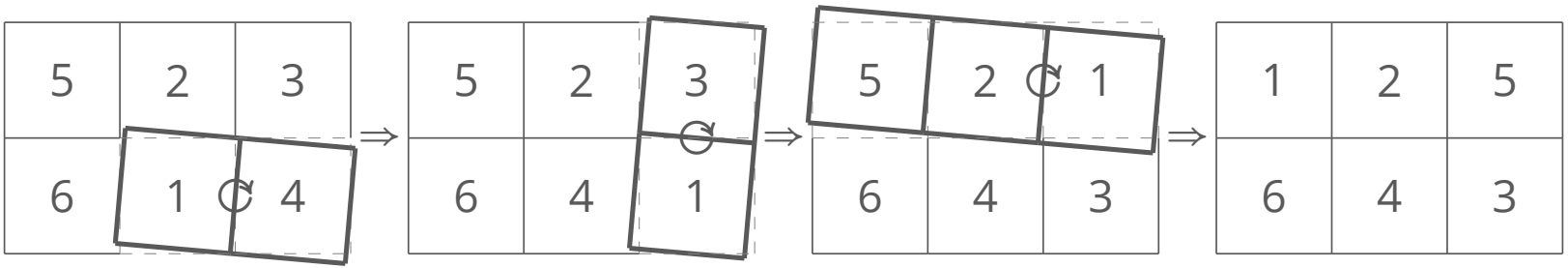
5 Subtask 3: Sorting two rows

Subtask 3 Solution 1 (69.5 pts)

- We combine two ideas from subtask 1 and subtask 2 respectively.
  1. In subtask 1, we use a "snake" pattern to transform a 2×2 grid into an array.
  2. In subtask 2, we use selection sort to fix the elements to the correct positions.

Subtask 3 Solution 1 (69.5 pts)

- We process the elements in the order $(1,1),\ldots,(1,N),(2,1),\ldots,(2,N)$.
- If the element is on another row, we use 3 operations to bring it to the correct position (do not move the processed elements!).



Number of rotations: $4N$

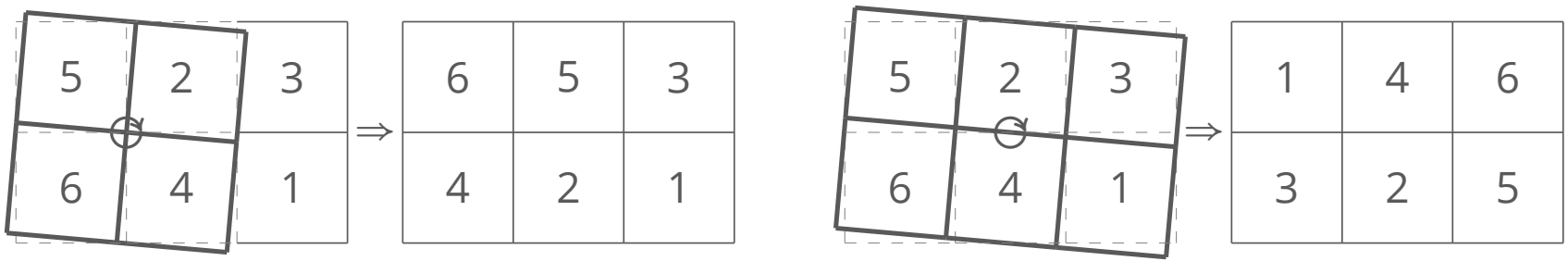
Expected score: 69.5

More Observations

Observation 2.1

If we apply the rotation operation on a square (1×1 or 2×2 subgrid), the subgrid will be rotated by $90°$. Otherwise, it will be rotated by $180°$.

Examples:

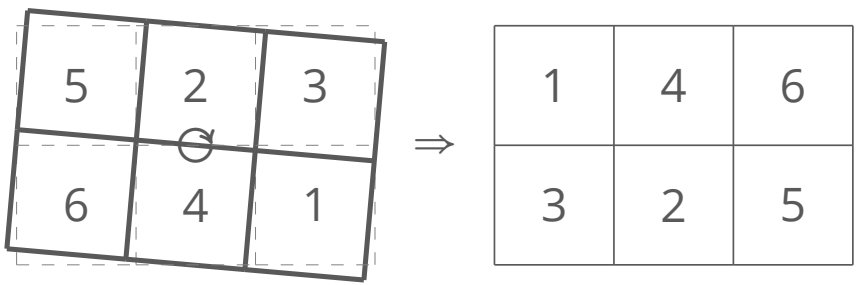


More Observations

Observation 2.2

If we apply the rotation operation on a $2 \times k$ subgrid, where $k > 2$, it is equivalent to reversing the arrays in both rows, and then swapping the values in each column.

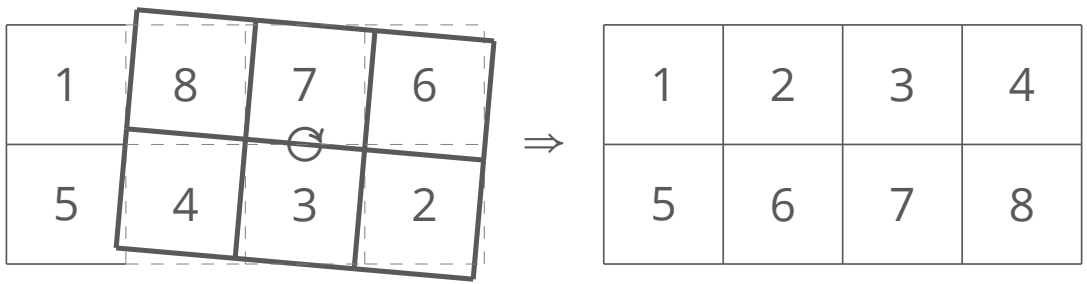
Example:



Subtask 3 Solution 2 (85.6 pts)

Note that in Subtask 3 Solution 1, we can reduce the number of rotations needed to fix a number from 3 to 1 or 2 using $2 \times k$ subgrids.

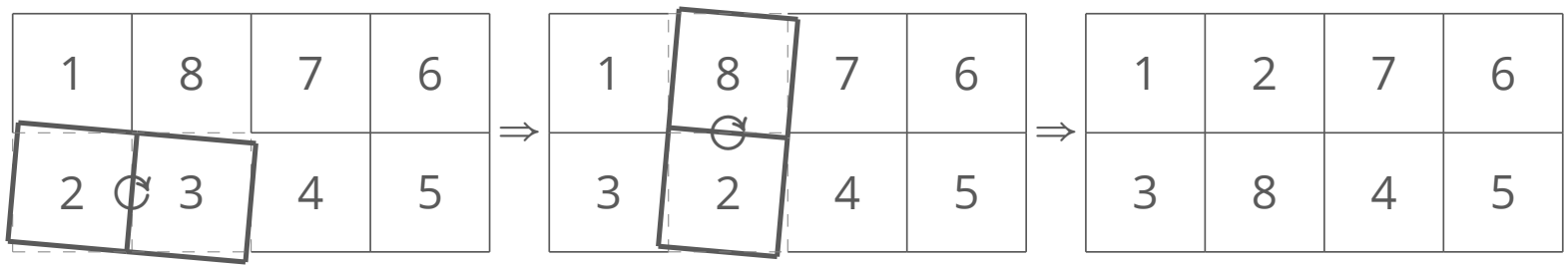
- If the position of the number is on the right of its correct position, we can use a single rotation operation (if the subgrid is not 2×2) or two rotation operations (if the subgrid is 2×2).
  An example that brings the number 2 to the correct position:



Subtask 3 Solution 2 (85.6 pts)

Note that in Subtask 3 Solution 1, we can reduce the number of rotations needed to fix a number from 3 to 1 or 2 using $2 \times k$ subgrids.

- If the position of the number is on the left of its correct position, we use one operation to bring it to the correct column and another operation to bring it to its correct position.
  An example that brings the number 2 to the correct position:



Subtask 3 Solution 2 (85.6 pts)

- The total number of operations used will be between $2N$ and $3N$. Theoretically, it should score 85.6 points.
- However, you might pass all test cases by enumerating the possible final configurations, and/or the order of processing the elements. The solution could use very few operations in practice.

A Two-Stage Algorithm

Here, we borrow the idea of subtask 1 solution 2.

A Two-Stage Algorithm

Here, we borrow the idea of subtask 1 solution 2.

1. We first compare the values for each column. If the upper value is greater than the lower value, we swap the two values.
   - Our ultimate goal here is to make $a_{1,i} < a_{2,i}$ for all i.

A Two-Stage Algorithm

Here, we borrow the idea of subtask 1 solution 2.

1. We first compare the values for each column. If the upper value is greater than the lower value, we swap the two values.
   - Our ultimate goal here is to make $a_{1,i} < a_{2,i}$ for all i.
2. Then, we sort the values for each row.
   - This is exactly what we have done in subtask 2!

Why is this correct?

A Two-Stage Algorithm

Theorem 3

Given a grid that satisfies $a_{1,i} < a_{2,i}$ for all $1 \le i \le N$. If we sort the arrays a_1 and a_2, the condition $a_{1,i} < a_{2,i}$ for all $1 \le i \le N$ still holds.

A Two-Stage Algorithm

Theorem 3

Given a grid that satisfies $a_{1,i} < a_{2,i}$ for all $1 \le i \le N$. If we sort the arrays a_1 and a_2, the condition $a_{1,i} < a_{2,i}$ for all $1 \le i \le N$ still holds.

Proof

Let a be the sorted grid. Consider the element $a_{1,i}$. Note that $a_{2,i} \ge a_{2,j}$ for all $1 \le j \le i$ (since a is sorted). At the same time, for each value $a_{2,j}$, there exists an element $a_{1,k}$ such that $a_{1,k} < a_{2,j}$ (take the corresponding element on the first row in the initial grid). Therefore, there are at least i values $a_{1,k}$ such that $a_{1,k} < a_{2,i}$. The result follows.

Subtask 3 Solution 3 (48.25 - 85.6 pts)

We follow the two-stage algorithm.

Subtask 3 Solution 3 (48.25 - 85.6 pts)

We follow the two-stage algorithm.

1. In the first stage, we compare the values column by column, and swap the two values using a single rotation operation if needed.
   - N operations are needed in the worst case.

Subtask 3 Solution 3 (48.25 - 85.6 pts)

We follow the two-stage algorithm.

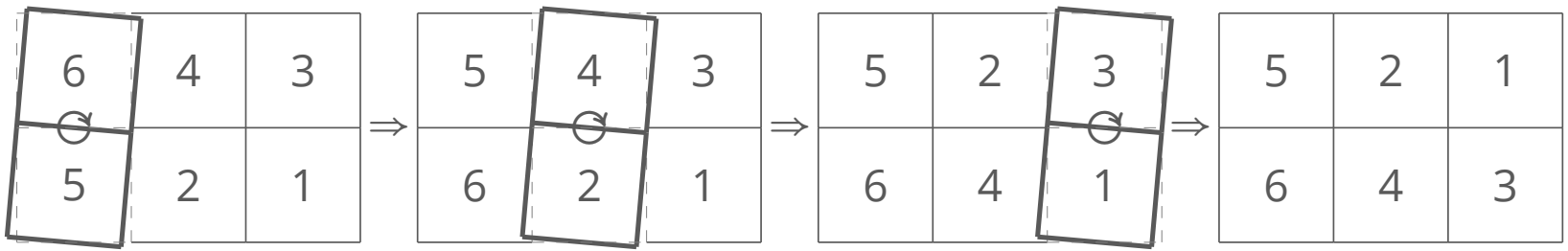
1. In the first stage, we compare the values column by column, and swap the two values using a single rotation operation if needed.
   - N operations are needed in the worst case.
2. In the second stage, we sort the values in each row using a solution from subtask 2.
   - If bubble sort is used ($N(N-1)$ operations), you get 48.25 points.
   - If insertion sort is used ($4N$ operations), you get 72.1 points.
   - If selection sort is used ($2N$ operations), you get 85.6 points.

Full Solution

Observe that it is hard to optimize the number of rotation operations used in the second stage. Hence we should focus on the first stage.

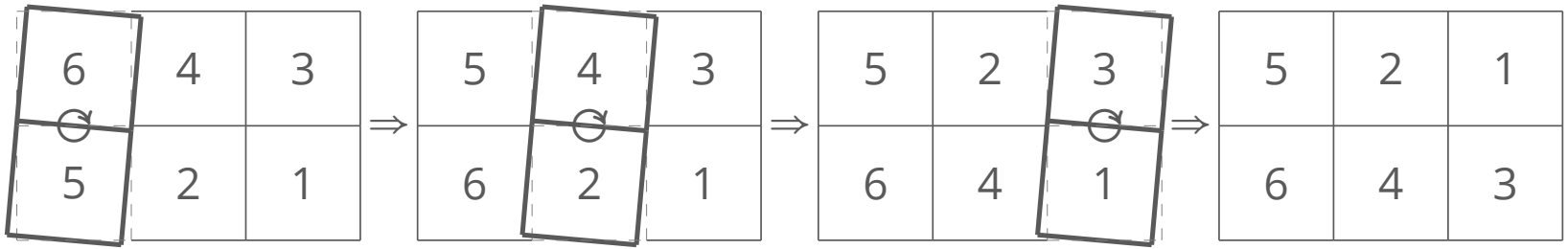
Full Solution

Consider this case with the worst case performance (N operations):

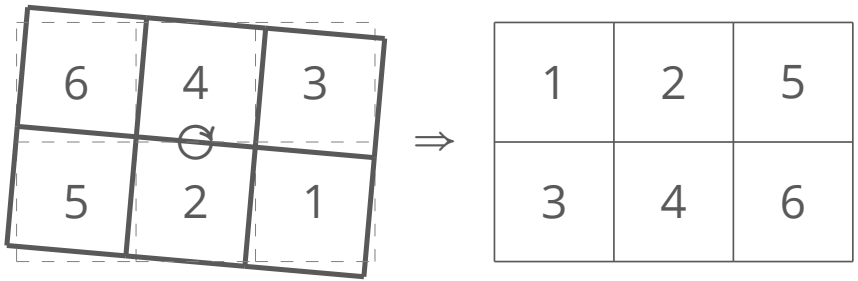


Full Solution

Consider this case with the worst case performance (N operations):



It is smarter to do this instead (1 operation only):



Full Solution

- What does the constraint $\lfloor 0.5N \rfloor$ suggest?

Full Solution

- What does the constraint $\lfloor 0.5N \rfloor$ suggest?
- Let x be the number of sorted columns and y be the number of unsorted columns. So $\min(x, y) \leq \lfloor 0.5N \rfloor$.

Full Solution

- What does the constraint $\lfloor 0.5N \rfloor$ suggest?
- Let x be the number of sorted columns and y be the number of unsorted columns. So $\min(x, y) \leq \lfloor 0.5N \rfloor$.
- If $y \leq x$, then $y \leq \lfloor 0.5N \rfloor$. We could swap the columns one by one without exceeding $\lfloor 0.5N \rfloor$ operations.

Full Solution

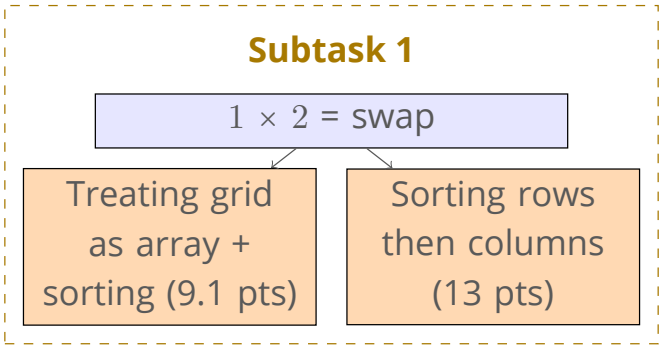
- What does the constraint $\lfloor 0.5N \rfloor$ suggest?
- Let x be the number of sorted columns and y be the number of unsorted columns. So $\min(x, y) \leq \lfloor 0.5N \rfloor$.
- If $y \leq x$, then $y \leq \lfloor 0.5N \rfloor$. We could swap the columns one by one without exceeding $\lfloor 0.5N \rfloor$ operations.
- If $y > x$, it is wiser to rotate the **entire grid** first, which swaps the values of x and y. After that, $x \leq \lfloor 0.5N \rfloor$. We could swap the columns one by one.

Full Solution

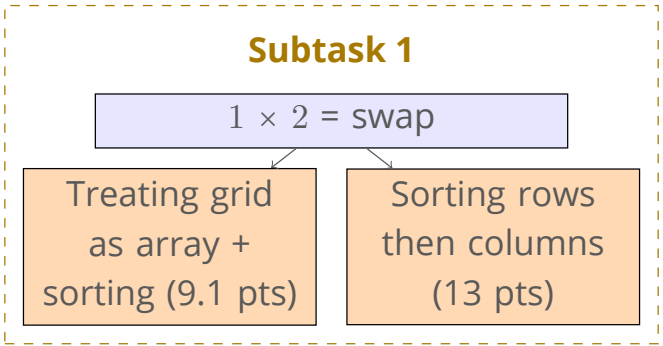
- What does the constraint $\lfloor 0.5N \rfloor$ suggest?
- Let x be the number of sorted columns and y be the number of unsorted columns. So $\min(x, y) \leq \lfloor 0.5N \rfloor$.
- If $y \leq x$, then $y \leq \lfloor 0.5N \rfloor$. We could swap the columns one by one without exceeding $\lfloor 0.5N \rfloor$ operations.
- If $y > x$, it is wiser to rotate the **entire grid** first, which swaps the values of x and y. After that, $x \leq \lfloor 0.5N \rfloor$. We could swap the columns one by one.
- You might notice that this stage takes $\lfloor 0.5N \rfloor + 1$ operations in the worst case, which is greater than $\lfloor 0.5N \rfloor$. However, since the selection sort stage only takes $2N - 2$ operations, the total number of operations would still be always less than $\lfloor 2.5N \rfloor$.

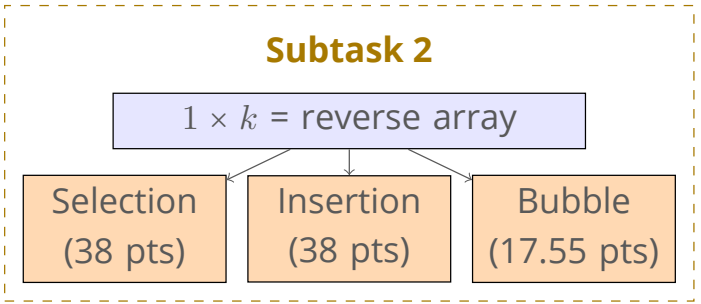
Summary

Summary

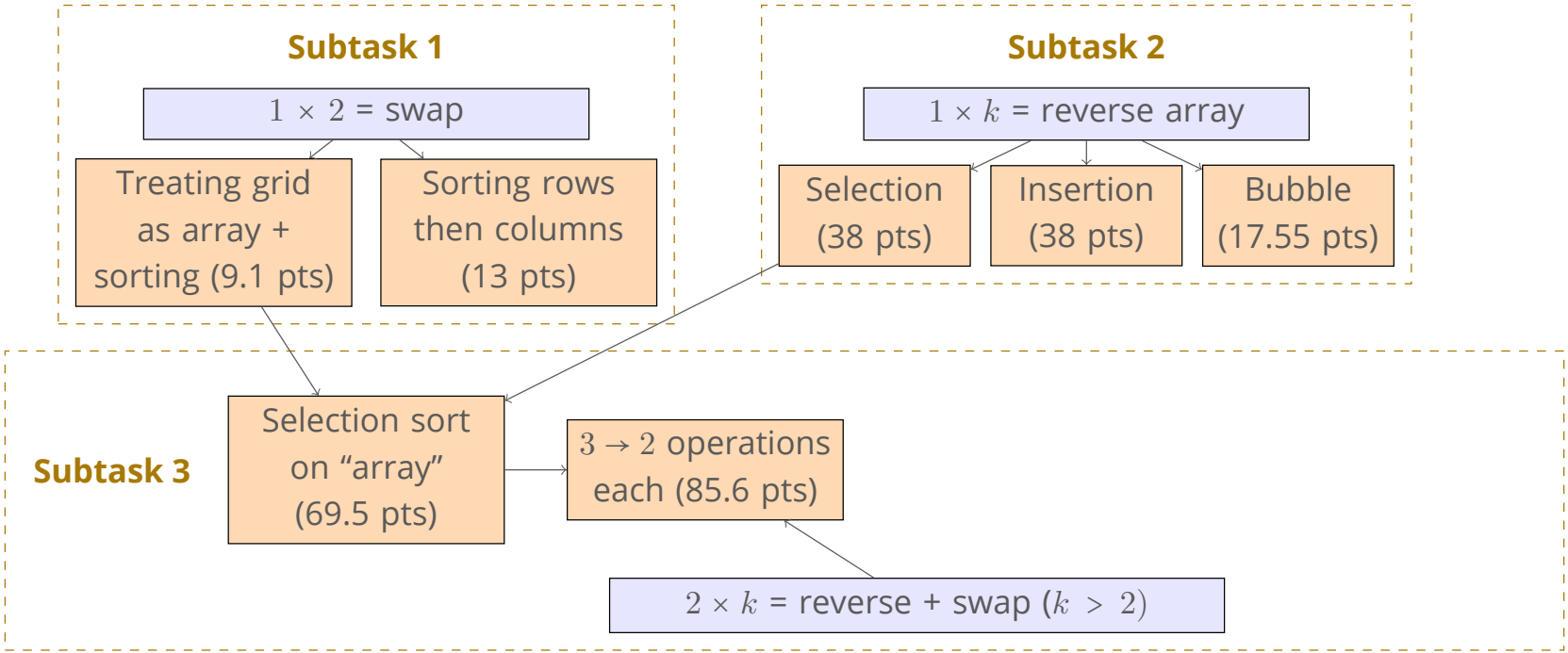

Subtask 1
1 × 2 = swap
Treating grid as array + sorting (9.1 pts)
Sorting rows then columns (13 pts)

Summary

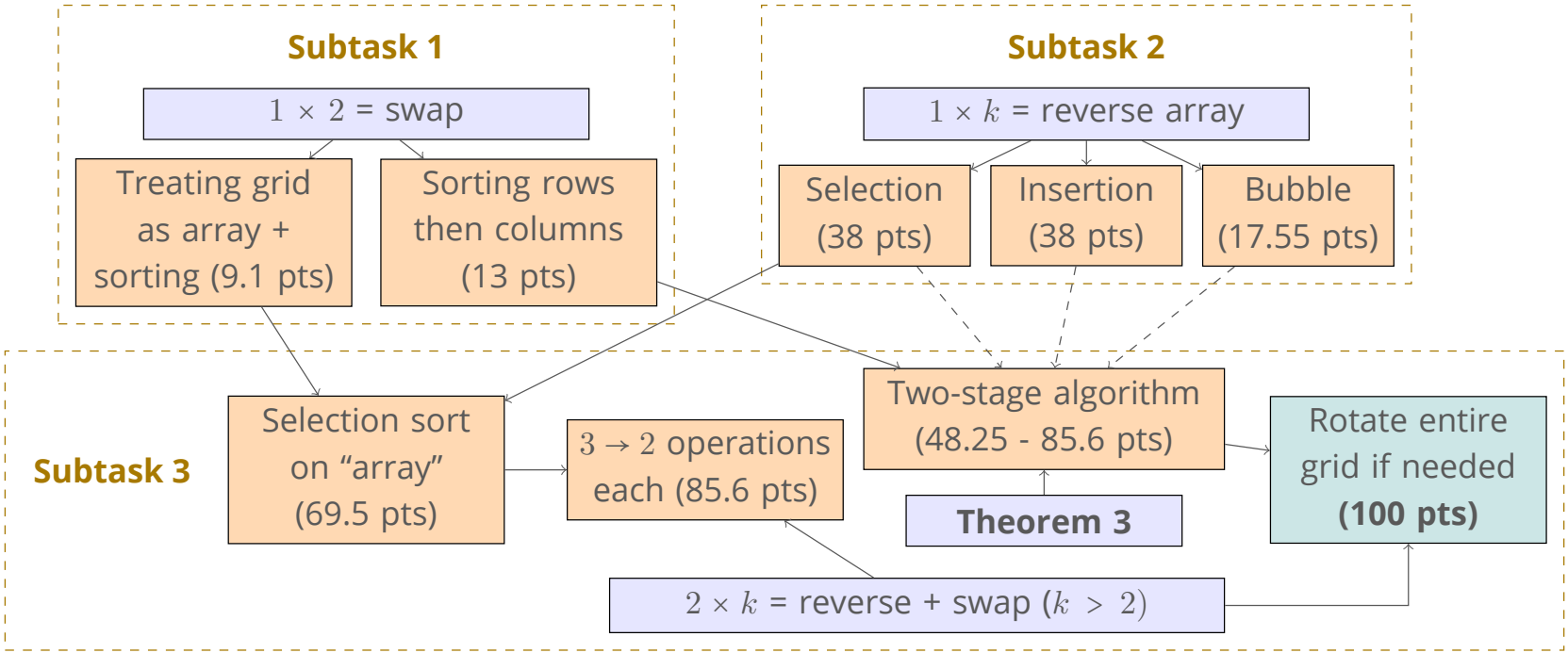

Subtask 1
1×2 = swap
Treating grid as array + sorting (9.1 pts)
Sorting rows then columns (13 pts)


Subtask 2
$1 \times k$ = reverse array
Selection (38 pts)
Insertion (38 pts)
Bubble (17.55 pts)

Summary


Subtask 1
1×2 = swap
Treating grid as array + sorting (9.1 pts)
Sorting rows then columns (13 pts)
Subtask 2
$1 \times k$ = reverse array
Selection (38 pts)
Insertion (38 pts)
Bubble (17.55 pts)
Subtask 3
Selection sort on "array" (69.5 pts)
$3 \to 2$ operations each (85.6 pts)
$2 \times k$ = reverse + swap ($k > 2$)

Summary


Subtask 1
1×2 = swap
Treating grid as array + sorting (9.1 pts)
Sorting rows then columns (13 pts)
Subtask 2
$1 \times k$ = reverse array
Selection (38 pts)
Insertion (38 pts)
Bubble (17.55 pts)
Subtask 3
Selection sort on "array" (69.5 pts)
$3 \to 2$ operations each (85.6 pts)
Two-stage algorithm (48.25 - 85.6 pts)
Theorem 3
Rotate entire grid if needed (100 pts)
$2 \times k$ = reverse + swap ($k > 2$)