Project 1 — Nightpass: A Survival Card Game

CMPE250 - Fall 2025

SAs: Ebubekir Erden, Emre Osman
TA: Beyza İrem Urhan

Due: 30.10.2025, 23:55

1 Introduction

Your ship has crashed. When you awaken, there are no other survivors — only you, stranded on a desolate island. With scraps from the wreckage and tools from the wild, you build a barrack and light a campfire each night, both for warmth and in the desperate hope that someone might see your signal.

But your fire draws attention of another kind. From the jungle's darkness emerges a shadowy figure, shaped like a man yet more than human. He calls himself The Stranger. You quickly notice he never crosses into the glow of your fire — whether because he cannot, or because he chooses not to, you cannot tell.

The Stranger offers a bargain: every night he will return, and you will duel him using cards you gather during the day. If he wins, he retreats into the darkness. If you win, he steps closer to your fire. The rules of this game are strict, and you realize your survival depends on mastering them.

What The Stranger does not say is that your fire has its own power. Each night, its warmth spreads like a healing aura, slowly restoring even your fallen warriors and calling them back to your deck. With every duel, every scar, every revival, your cards carry the weight of survival.

You are The Survivor. Will you endure The Stranger's challenge, night after night — or will the fire fade?

2 General Description

You are expected to read commands from the "input.txt" and write expected outputs into "output.txt" file. In this input file there will be cards to be added to the deck and commands to be executed. You need to generate correct outputs for each line in input file. The number of lines in the input and output files should be same.

The game is a card dueling game. You need to manage your cards efficiently. These cards will be stored in the deck and the deck will be consisting of cards with names, attack and health points. There will be cards with same health and/or same attack. You are expected to be filter the cards correctly to find optimal match by some priority rules for The Stranger's card and get points by killing or damaging the card.

For the remaining of the text, some parameters' meanings are as following:

- A_{init} : initial attack value that the card had when it was first drawn.
- A_{base} : through the game card's baseline attack value can be changed by revival processes. This value means current base attack value without any battle damage.
- A_{cur} : through the game card's attack score can change by battle damages or revival penalties. This value shows this and means current attack score.
- H_{init} : initial health value that the card had when it was first drawn.
- H_{base} : through the game card's baseline health value can be changed. This value means current base health value without any battle damage.
- H_{cur} : through the game card's health score can change by battle damages. This value shows this and means current health score.

3 Card Piles

There are two types of card piles of The Survivor: the Deck and the Discard Pile. The Stranger does not have a discard pile nor a deck.

3.1 The Deck

The deck contains all active cards that have not been discarded. Formally, a card belongs to the deck if

$$0 < health \leq H_{base}.$$

This includes both fresh (undamaged) and damaged cards. Damaged cards remain playable in future battles with their reduced stats, and they continue to be selected according to the battle priority rules.

A card can be entered into deck through `draw_card` command. In this command there will be initial health H_{init}, attack A_{init} points and card's `unique alphanumerical` name. They are removed from the deck through `steal_card` command or if they are dead (in that case they will go to the discard pile). For more detailed information about commands look at the Commands Section.

3.2 The Discard Pile (for Type-2)

The discard pile stores cards that have been defeated and reduced to zero health. A card is considered discarded if

$$H_{cur} = 0 \quad \wedge \quad 0 < H_{missing} \leq H_{base}.$$

Discarded cards are not immediately lost; they can re-enter the game through revival mechanics such as the Healing Phase or special actions.

4 Commands

There are various commands that you should implement independently:

- Battle Commands:
  - `draw_card <card_name> <attack_init> <health_init>`
  - `battle <stranger_card_attack> <stranger_card_health> <heal_pool_amount>`
  - `steal_card <attack_limit> <health_limit>`
- Queries:
  - `deck_count`
  - `discard_pile_count`
  - `find_winning`

4.1 Draw Card

You ready your deck by drawing cards. This is the basic way of a card to enter your deck. You will be given in input file: a **unique alphanumerical** name, a initial attack point **attack_init** to hurt The Stranger's cards and a initial health point **health_init** to survive encounters. Both the attack and the health points are integers and **not unique**. With this values you add your card to the deck.

There can be multiple cards with same health and/or same attack point. You need to order them correctly.

Input: `draw_card <card_name> <attack_init> <health_init>`

Output: `Added <card_name> to the deck`

Example:

```
Input: draw_card gloomst4lker 17 25
Output: Added gloomst4lker to the deck
```

4.2 Battle

Each night you battle with The Stranger and get scores to survive. In this battles you need to select optimal match for The Stranger's card and try to heal as much of your cards as possible. Battles have two phases and will be conducted as stated in the order below:

1. Battle Phase
2. Healing Phase (for Type-2)

4.2.1 Battle Phase

At this stage, The Stranger reveals his card. To counter and get most points, you must play your cards in the order dictated by the battle priority system. If the card search for first priority fails you go to next priority and this follows until last priority. There will be **only one** correct card to put against The Stranger's card.

1. **First Priority: Survive and Kill:**
   First, look for a card that can both survive The Stranger's attack and discard his card in return. When you have several options, choose the one with the smallest health and attack values that still get the job done, so you don't waste a stronger card too early.
2. **Second Priority: Survive and Not Kill:**
   If no card can both survive and kill, then focus on survival and damage. Play a card that won't die but can deal as much damage as possible. If two or more cards deal the same maximum damage, go with the one that has the lowest health.
3. **Third Priority: Kill and Don't Survive:**
   Sometimes you'll find that no card can survive at all. In that case, forget about defense and aim for destruction. Choose the card with the lowest attack higher than opponent's card's health, and if multiple cards tie, again prioritize the one with the smallest health.
4. **Fourth Priority: Maximum Damage:**
   Since you cannot kill the opponent's card's, focus on maximum damage and choose the card with maximum attack. If equal ones exist, choose the one with minimal health again.
5. **Tie-Breaker:**
   In any of these conditions, you might still have multiple candidates. When that happens, play the card that entered your deck first. Remember, though, that any change in attack, health, or revival process makes a card count as newly entered for this purpose.

Battle Resolution:

- If there is no card that can counter The Stranger's card, print `"No cards to play"`. In this case, The Stranger automatically gains +2 points.
- If you do have a candidate, apply simultaneous damage: subtract your card's attack score from The Stranger's card's health, and subtract The Stranger's card's attack score from your card's health.

- If your card's health is less than or equal to 0, your card dies and The Stranger gains +2 points.
- If The Stranger's card's health is less than or equal to 0, his card dies and you gain +2 points.
- If The Stranger reduces your card's health but does not kill it ($0 < \text{health} \leq H_{base}$), The Stranger gains +1 point.
- If you reduce The Stranger's card's health but do not kill it ($0 < \text{health} \leq H_{base}$), you gain +1 point.
- Both sides can score in same turn
- If your card is only damaged and not dead, its attack is reduced proportionally to its new health: **Resolution:**

$$A'_{cur} = \max\left(1,\ \left\lfloor \frac{A_{base} \cdot H_{cur}}{H_{base}} \right\rfloor\right)$$

This means, if the health of the card reduced by 5%, then the attack score will be reduced by 5% too. Attack reduction always calculated from the base attack score A_{base}. However, in revival process this value can change.

Mathematically

Priority Rules:

1. **First Priority: Survive and Kill:**
   Choose a card such that

$$H_{\text{cur}} > A_{\text{stranger}} \ \wedge \ A_{\text{cur}} \geq H_{\text{stranger}}.$$

   Among such cards, select the one with minimal A_{cur}, and if tied, minimal H_{cur}.

2. **Second Priority: Survive and Not Kill:**
   If no card satisfies the first priority, choose a card such that

$$H_{\text{cur}} > A_{\text{stranger}} \ \wedge \ A_{\text{cur}} < H_{\text{stranger}}.$$

   Among these, pick the one with maximal A_{cur}, and if tied, minimal H_{cur}.

3. **Third Priority: Kill and Don't Survive:**
   If no card survives, select a card such that

$$H_{\text{cur}} \leq A_{\text{stranger}} \wedge A_{\text{cur}} \geq H_{\text{stranger}}$$

   Among these, choose the one with minimal A_{cur}, and if tied, minimal H_{cur}.

4. **Fourth Priority: Maximum Damage:**
   Choose the card with maximum A_{cur} regardless of its H_{cur}. If tied minimal H_{cur}.

5. **Tie-Breaker:**
   If multiple cards remain after applying the above rules, pick the card that entered your deck first. Any change in attack, health, or revival counts the card as newly entered.

Resolution:

$$H'_{cur} = H_{cur} - A_{stranger}$$
$$H'_{stranger} = H_{stranger} - A_{cur}$$
$$A'_{cur} = \max\left(1,\ \left\lfloor \frac{A_{base} \cdot H_{cur}}{H_{base}} \right\rfloor\right)$$

Scoring:

- If no candidate exists: The Stranger + 2.
- If $H'_{cur} \leq 0$: The Stranger + 2.
- If $H'_{stranger} \leq 0$: The Survivor + 2.
- If $0 < H'_{cur} \leq H_{base}$: The Stranger + 1.
- If $0 < H'_{stranger} \leq H_{base}$: The Survivor + 1.

Note: Both sides can score in the same round.

Example Battle Scenarios with Attack Updates

Scenario 1: The Survivor Wins and Kills

- **Card:** $H_{\text{cur}} = 5, A_{\text{cur}} = 4, A_{\text{base}} = 4, H_{\text{base}} = 5$
- **Stranger:** $H_{\text{stranger}} = 3, A_{\text{stranger}} = 3$
- **Priority Rule Applied:** First Priority
- **Resolution:**

$$H'_{\text{cur}} = 5 - 3 = 2, \quad H'_{\text{stranger}} = 3 - 4 = -1$$
$$A'_{\text{cur}} = \max\left(1, \left\lfloor \frac{4 \cdot 2}{5} \right\rfloor\right) = \max(1, 1) = 1$$

- **Scoring:** $H'_{\text{stranger}} \leq 0 \Rightarrow$ The Survivor $+2$, $H'_{\text{cur}} < H_{base} \Rightarrow$ The Stranger $+1$

Scenario 2: The Survivor Survives but Does Not Kill

- **Card:** $H_{\text{cur}} = 5, A_{\text{cur}} = 2, A_{\text{base}} = 2, H_{\text{base}} = 6$
- **Stranger:** $H_{\text{stranger}} = 5, A_{\text{stranger}} = 4$
- **Priority Rule Applied:** Second Priority
- **Resolution:**

$$H'_{\text{cur}} = 5 - 4 = 1, \quad H'_{\text{stranger}} = 5 - 2 = 3$$
$$A'_{\text{cur}} = \max\left(1, \left\lfloor \frac{2 \cdot 1}{6} \right\rfloor\right) = \max(1, 0) = 1$$

- **Scoring:** $0 < H'_{\text{cur}} < H_{\text{base}} \Rightarrow$ The Stranger $+1, H'_{\text{stranger}} < H_{str,base} \Rightarrow$ The Survivor $+1$

Scenario 3: The Survivor Cannot Survive but Can Kill

- **Card:** $H_{\text{cur}} = 2, A_{\text{cur}} = 6, A_{\text{base}} = 13, H_{\text{base}} = 5$
- **Stranger:** $H_{\text{stranger}} = 5, A_{\text{stranger}} = 4$
- **Priority Rule Applied:** Third Priority
- **Resolution:**

  $H'_{\text{cur}} = 2-4 = -2 \Rightarrow 0$ `(Clamped to 0 since the card is dead)`, $H'_{\text{stranger}} = 5-6 = -1$
- **Scoring:** $H'_{\text{cur}} \leq 0 \Rightarrow$ The Stranger $+2, H'_{\text{stranger}} < H_{str,base} \Rightarrow$ The Survivor $+2$

Scenario 4: The Survivor Cannot Survive and Kill

- **Card:** $H_{\text{cur}} = 2, A_{\text{cur}} = 3, A_{\text{base}} = 3, H_{\text{base}} = 5$
- **Stranger:** $H_{\text{stranger}} = 5, A_{\text{stranger}} = 4$
- **Priority Rule Applied:** Fourth Priority
- **Resolution:**

  $H'_{\text{cur}} = 2-4 = -2 \Rightarrow 0$ `(Clamped to 0 since the card is dead)`, $H'_{\text{stranger}} = 5-3 = 2$
- **Scoring:** $H'_{\text{cur}} \leq 0 \Rightarrow$ The Stranger $+2, H'_{\text{stranger}} < H_{str,base} \Rightarrow$ The Survivor $+1$

4.2.2 Healing Phase (for Type-2)

IMPORTANT: This part is not included in Type-1 and `heal_pool_points` value will always be zero for Type-1

After the duel, dead cards are sent to the discard pile, while surviving cards return to the deck with their damaged stats. However, death is not the end: with the warmth of the campfire reaching into the discard pile, fallen cards may be revived.

At every battle command you receive a healing pool of points, denoted as `heal_pool_points`, which can be spent to revive cards in either discard pile independently. Each card has a $H_{missing}$ value defined as

$$H_{missing} = H_{base} - revival\_progress,$$

which must be filled to bring the card back to life. `revival_progress` here means the current health points added to the dead cards. Cards can be revived either fully or partially depending on available health points. To get out of the discard pile and go back into pile their `revival_progress` should be equal to H_{base} and $H_{missing} = 0$. Similar to battle algorithm, you should get through some priority steps.

Healing Algorithm:

1. **First Priority:** Prioritize the card with the largest $H_{missing}$ that can be *fully revived* with the current healing pool. Subtract the revived card's H_{base} from $heal\_pool\_points$.
2. **Second Priority:** If points remain after a revival, repeat the process: select the next card with the largest $H_{missing}$ that is still smaller than or equal to remaining $heal\_pool\_points$. Continue until there is not a card left you can fully revive.
3. **Third Priority:** If no further card can be fully revived, apply the remaining points to the smallest $H_{missing}$ card, break ties by first points to first discarded. That card becomes *partially revived*.
4. If there are not enough cards to spent all of the healing points, than they go into waste.

Rules:

1. A *fully revived* card returns to the deck, while a *partially revived* card remains in the discard pile.
2. Attack points are reduced when revived:

$$attack' = \begin{cases} \lfloor attack \times 0.90 \rfloor & \text{if fully revived,} \\ \lfloor attack \times 0.95 \rfloor & \text{if partially revived,} \end{cases}$$

   meaning every revived card suffers at least a 10% reduction to its base attack score A_{base} and partial revive penalty applied at each time but when it revives fully, only fully revived point reduction penalty will be applied, not both.
3. All `heal_pool_points` must be spent each phase (if there are enough cards).
4. After partial revival, the card's $H_{missing}$ is updated, and it is treated as a newly discarded card for future healing.

4.2.3 Input/Output Formulation

Input: `battle <stranger_card_attack> <stranger_card_health> <heal_pool_amount>`

Parameters:

- `<priority_number>`: The priority step among battle steps that we have found the correct card. It must be between 1 and 4 included ($1 \leq priority\_score \leq 4$)
- `<stranger_card_attack>`: Attack score of the opponent's card.
- `<stranger_card_health>`: Health of the opponent's card.
- `<heal_pool_amount>`: Total points available for reviving cards this turn. (This part will be always 0 for Type-1 cases.)
- `<k>` : Total number of cards revived this night. Even if it is 0 you need to write it in the output. (For Type-1, it should always be 0)

Output (case-by-case):

- **Case 1: Card played and discarded:**

  `Output: Found with priority <priority_number>, Survivor plays <card_name>, the played card is discarded, <k> cards revived`

  The played card dies during the battle and `<k>` cards are revived using the healing pool.

- **Case 2: Card played and returned to deck**

  `Output: Found with priority <priority_number>, Survivor plays <card_name>, the played card returned to deck, <k> cards revived`

  The played card survives and returns to the deck, and `<k>` cards are revived using the healing pool.

- **Case 3: No valid card to play**

  `Output: No card to play, <k> cards revived`

  No card can be played against the opponent's card; no card is played and `<k>` cards are revived using the healing pool.

Example:

```
Input: battle 5 3 7
```

Case 1 – Card played and discarded:
```
Output: Found with priority 3, The Survivor plays StoneGolem, the played card is discarded, 1 cards revived
```

Case 2 – Card played and returned to deck:
```
Output: Found with priority 2, The Survivor plays FireElemental, the played card returned to deck, 2 cards revived
```

Case 3 – No valid card to play:
```
Output: No cards to play, 0 cards revived
```

4.3 Deck Card Count

You need to count number of cards in the deck and log it.

Input: `deck_count`

Output: `Number of cards in the deck: <k>`

`<k>` is the number of cards.

Example:

```
Input: deck_count
Output: Number of cards in the deck: 8
```

4.4 Discard Pile Card Count (for Type-2)

You need to count number of cards in the discard pile and log it.

Input: `discard_pile_count`

Output: `Number of cards in the discard pile: <k>`

`<k>` is the number of cards.

Example:

```
Input: discard_pile_count
Output: Number of cards in the discard pile: 13
```

4.5 Find Current Winner

The command `find_winning` determines which side is currently leading the game. The winner is decided by comparing accumulated points. Points come from only one source:

- Battle outcomes (each round contributes between 0 and +2 points to either side)

If both players have the same score, The Survivor is considered the winner.

Input:

```
find_winning
```

Output (case-by-case):

- **Case 1: The Survivor is winning**

  ```
  The Survivor, Score: <survivor_points>
  ```

- **Case 2: The Stranger is winning**

  ```
  The Stranger, Score: <stranger_points>
  ```

Examples:

```
Input:
find_winning

Output:
The Survivor, Score: 12
```

```
Input:
find_winning

Output:
The Stranger, Score: 14
```

4.6 Steal Card

Sometimes The Stranger will steal your cards. He aims for the cards with minimum attack power higher than `attack_limit` and among those one with the minimum health score higher than `health_limit`. As a tie-breaker use first card stolen first policy again. Stolen cards do not go into discard pile and they are instantly removed from the deck. If there is no such a card print "`No card to steal`".

Input:

```
steal_card <attack_limit> <health_limit>
```

Output:

```
The Stranger stole the card: <card_name>
```

Examples:

```
Input:
steal_card 10 8

Output:
The Stranger stole the card: BatchedOne
```

In this example for the given attack limit 10 and health limit 8; the card with the name `BatchedOne` should have at minimum 11 score and 9 health score

```
Input:
steal_card 1 800

Output:
No card to steal
```

5 Game Penalties and Boundaries

You need to keep track of all permanent and non-permanent penalties/bonuses while work inside of the parameter boundaries.

5.1 Penalties and Bonuses

This section distinguishes temporary (non-permanent) modifications to card stats from permanent changes. The rules below are deterministic and must be applied in the stated order.

Temporary (non-permanent) effects The following effects modify a card's *current* stats but do *not* change the card's baseline stats. These temporary changes are cleared when a card dies (moves to the discard pile).

- **Battle resolution scaling:** After any battle in which a card survives, recompute its `current_attack` A_{cur} from its baseline A_{base} and its new `current_health` H_{cur} using the formula below. This recomputed attack is temporary and is reset when the card dies.

All temporary recomputations use this exact formula (apply integer floor and then clamp):

$$A'_{cur} = \max\left(1,\ \left\lfloor \frac{A_{base} \cdot H_{cur}}{H_{base}} \right\rfloor\right)$$

Notes on temporary effects:

- Use integer arithmetic: compute the product, divide, then take the floor.
- The `max(1,...)` clamp forces the temporary current attack to be at least 1.
- These temporary values are recalculated from the latest baseline values (which may have changed previously due to revival penalties or `transform`).

Permanent effects (for Type-2): Only the following produce *permanent* changes to a card's baseline statistics.

- **Revival penalties:** Permanent reductions to a card's baseline attack occur *only* when a card is revived (partial or full revival). Let A_{base} denote the card's current baseline attack immediately before applying a revival penalty. When a revival penalty is applied, update the baseline attack multiplicatively and with integer flooring:

$$A_{\text{base}} \leftarrow \left\lfloor A_{\text{base}} \times (1 - p) \right\rfloor,$$

where

$$p = \begin{cases} 0.05 & \text{for a partial revive (each partial revive applies a 5\% reduction),} \\ 0.10 & \text{for a full revive (applies a 10\% reduction).} \end{cases}$$

Apply multiple penalties sequentially in chronological order (each penalty multiplies the current A_{base}). For example, two partial revives apply two successive 0.95 multipliers (with floor at each step). By design, a completed full revival yields at least a 10% permanent reduction compared to the pre-revival baseline.

Interactions and ordering

- Permanent penalties update the baseline values that are used for all later temporary recomputations. In particular, after a revival penalty is applied, subsequent temporary recomputations use the updated A_{base} and the current H_{base}.
- Temporary buffs/debuffs from battle commands do not change the baseline and are reset when the card dies. Permanent changes only arise via revival penalties changing the permanent H_{base}.

5.2 Boundaries

- **Health bounds:** For any card

$$0 \leq H_{cur} \leq H_{base}.$$

  If an operation would reduce health below 0, treat it as 0 (card is discarded).

- **Revive progress bounds (for Type-2):**

$$0 \leq \text{revive\_progress} \leq H_{base}, \qquad 0 \leq H_{missing} \leq H_{base}.$$

- **Attack bounds:** Attack values are integers and must satisfy

$$A_{cur} \geq 1.$$

  There is no explicit upper cap on attack in the rules.

- **Rounding/integers:** All percentage changes and derived attack values use the floor operator $\lfloor \cdot \rfloor$ and then are clamped by the bounds above.

6 Grading and Constraints

The project is evaluated using two types of test cases. Each type exercises a different feature set and according to passing test cases you will get partial points as indicated below. If all test cases for both types have passed you will get the total score of 100 points.

By only doing Type-1 correctly you can take maximum of 90 points. The Type-2 commands are written below and stated in their descriptions.

6.1 Project Types

Type 1 — Basic (maximum 90 pts)

Commands exercised:

- `draw_card`

- `battle` (with `heal_pool_amount` = 0 for every battle)
- `find_winning`
- `deck_count`
- `steal_card`

Notes:

- No healing phase inside the battle
- Discard pile is not used in these tests (no revival processes).
- Focuses on basic card drawing, battle resolution, and score tracking.

Type 2 — Full (maximum 100 pts)

Includes all Type 1 commands, plus:

- `battle` with varied `heal_pool_amount` values
- One discard pile with revival processes and associated penalties
- `discard_pile_count`

Notes:

- Tests partial/full revival, and associated penalties.
- Ensures correct handling of discard pile and resurrection logic.

6.2 Test Cases

For each project type there will be 4 level of test cases:

- `Demo:`
  - ∼`100` commands
  - 50-70 cards

  Important: These test cases are only for you to see and follow how the code works and they will not be used for grading.
- `Small:`
  - ∼`10k` commands
  - 5-7k cards

- `Medium:`
  - ∼`100k` commands
  - 50-70k cards
- `Large:`
  - ∼`550k` commands
  - 350-400k cards

There will be some test cases given to you in different sizes. However, they will not be the only test cases that can be used for grading. Each test cases' affect on the total score is same and there will be equal amount of test cases for all three size (small, medium, large). To a test case to succeed, not only the outputs should match but also should be done under time limit.

6.3 Variable Constraints

Table 1: Supported Command Syntax by Test Type

Command	**Syntax**	**T1**	**T2**	**Description**
Draw Card	`draw_card <name> <att> <hp>`	✓	✓	Add card to deck
Battle (no heal)	`battle <att> <hp> 0`	✓	✓	Battle with no healing
Battle (with heal)	`battle <att> <hp> <heal>`	×	✓	Battle with heal pool
Steal Card	`steal_card <att_lim> <hp_lim>`	✓	✓	Steal cards within limits
Deck Count	`deck_count`	✓	✓	Query deck size
Discard Count	`discard_pile_count`	×	✓	Query discard pile size
Find Winner	`find_winning`	✓	✓	Find winning card

Table 2: Approximate Command Usage by Test Type and Size

(a) Type 1

Command	**Demo**	**Small**	**Medium**	**Large**
draw_card	60	6K	60K	350K
battle (heal_pool=0)	25	2.8K	28K	130K
steal_card	5	600	6K	30K
deck_count	8	400	4K	15K
discard_pile_count	0	0	0	0
find_winning	5	200	2K	8K
Total	**103**	**10K**	**100K**	**533K**

(b) Type 2

Command	**Demo**	**Small**	**Medium**	**Large**
draw_card	60	6K	60K	350K
battle (all types)	25	2.8K	28K	130K
steal_card	5	600	6K	30K
deck_count	5	300	3K	12K
discard_pile_count	5	300	3K	12K
find_winning	3	200	2K	8K
Total	**103**	**10.2K**	**102K**	**542K**

6.4 Time Constraints

To get full points you need to pass the test cases under the time limits shown below.

Table 3: Time limits (seconds) by test type and size

Test Type	Small	Medium	Large
Type 1 (Basic)	1	3	15
Type 2 (Full)	1	3	15

In our code small cases took less than 0.2 seconds, medium cases took less than 1 second and large cases took less than 6 seconds.

7 Submission

- **ZIP filename:** Submit a single ZIP archive named `<your_student_number>.zip`. Example: `2023000000.zip`. Files that do not follow this naming convention may incur score penalties.
- **Archive contents:** The ZIP should contain all files and folders sent to you (test_runner.py, src/, testcase_inputs/, testcase_outputs/, and output/). You will add your implemented classes inside the src/ folder. Do not create new folder and when sending the project through moodle, ensure only `*.java` files are in the src/ folder (no other files). Also, do not change the name of your main file: `Main.java`, but you can change the content of it as you wish. Failure to comply with these standard can produce point reductions.

  Important: Do not upload txt files for both inputs and outputs. Tehir folders testcase_inputs and testcase_outputs should stay.
- **Compilation and execution:** Your code must compile and run with the following commands (run from the directory where the `*.java` files are located):

```
javac *.java
java Main <input_file> <output_file>
```

  As a more recommended way, you can also test your code with `python` commands:

```
python3 test_runner.py
python3 test_runner.py --type type1
python3 test_runner.py --type type2
```

  For more detailed python commands look to the `src/Main.java` file. For this option to work you should have python3 installed in your computer. This python script runs all test cases, compares the outputs with the correct outputs and shows time they took to execute.
- **Implementation notes:**

  - Ensure the main class of your code's name is `Main.java`
  - The program should read from the filename given as `<input_file>` and write results to the filename given as `<output_file>`.

- **Important:** Double-check the ZIP name and that all required source files are present before submitting. Missing files or incorrect archive names are a common cause of automatic penalties.

8 Warnings

- Any data structure other than `ArrayList` is forbidden; you may not use other `java.util` classes.
- Use of `java.Math` is suggested for mathematical operations.
- All source codes are checked automatically for similarity with other submissions and exercises from previous years. Make sure you write and submit your own code. Any sign of cheating will be penalized by at least -100 points on the first attempt and may lead to disciplinary action in case of recurrence.
- Document your code with necessary inline comments and use meaningful variable names. *Do not* over-comment, and do not make your variable names unnecessarily long. This is important for partial grading.
- Make sure that the whitespace in your output is correct. Trailing whitespace at the end of a line may be disregarded, but internal spacing and line breaks must match the specification.
- Please use the discussion forum on Moodle for questions, and check whether your question has already been answered before posting.