Project 2 - GigMatch Pro

CmpE 250, Data Structures and Algorithms – Fall 2025

SAs: Bora Yılmazer, Onur Küçük
TA: İrem Urhan
Instructor: Atay Özgövde

Due: November 19th, 2025

1 Introduction

Your cousin just graduated from business school and came to you with an ambitious idea: building a freelancing/gig economy platform inspired by Armut, where customers can easily find and hire skilled freelancers for various services. The concept is simple yet powerful—customers post job requests within their budget, and the system intelligently matches them with the best available freelancers.

But here's the challenge: with potentially thousands of users and service requests happening simultaneously, your cousin needs a system that can handle the load efficiently. Customers expect instant results when they search for services, and freelancers want their profiles ranked fairly based on their performance.

Your cousin knows you're taking a data structures course and believes you have exactly the skills needed to build this system. The question is: **Can you design a system that scales gracefully, maintains freelancer rankings efficiently, and processes requests in real-time?** Your decisions will determine whether this platform succeeds or fails under real-world pressure.

2 Application Overview

The application should register customers and freelancers, process service requests and employments, support cancellations with consequences, provide user queries, and support a month simulation mechanism that updates burnout, skill evolution, and loyalty tiers.

2.1 Users

The application has two types of users:

- **Freelancer:** Freelancers are users who are willing to work in exchange for money. They provide one service for a fixed price, and they can work for one customer at a time. Their abilities are represented by five skill parameters. After each completed or cancelled job, their rating is updated based on performance. If they take on too many jobs within a short period, they experience burnout, which temporarily lowers their ranking and makes them less likely to be chosen by customers.

- **Customer:** Customers use the platform to find and hire freelancers for specific services. They pay for completed jobs and gain loyalty status based on their total spending. Customers can employ multiple freelancers at the same time.

2.2 Services

Services are the types of work offered on the platform, such as painting, web development, or tutoring. There are ten predefined service categories in total. Each service emphasizes different skills, so some freelancers are better suited for certain types than others. Freelancers can change the service they offer. If their skills are strong enough for the new category, this change is processed at the end of a month.

2.3 Employment

Employment represents the connection between a customer and a freelancer for a specific job. A customer can request a service and see a list of the most suitable freelancers based on a composite ranking, after which the system automatically assigns the best one. Alternatively, a customer may directly choose a freelancer by ID. An employment continues until it is completed or cancelled. If the customer cancels, they lose loyalty points; if the freelancer cancels, they receive a zero-star rating and skill degradation. Also, frequent cancellations can lead to blacklisting for freelancers. When a job is completed successfully, the customer rates the freelancer out of five stars, and the freelancer becomes available for new work.

2.4 Month Simulations

Month simulations represent the passage of time in the system. Each simulated month allows the platform to update user statuses and apply scheduled changes. After a month ends, the system checks freelancers for burnout or recovery, applies any requested service changes, and updates customer loyalty tiers based on their total spending. The duration of a simulated month is flexible and only serves to reflect varying levels of activity on the platform.

2.5 Blacklists

The platform supports two levels of blacklisting. Customers can personally blacklist freelancers, preventing them from appearing in that customer's future job requests until unblacklisted. Separately, the platform automatically applies a **platform-level blacklist** to freelancers who cancel five or more jobs within a simulated month, permanently removing them from all future employments across the system.

2.6 Preliminary Warnings

- The app can create customers and freelancers with IDs that must be globally unique across both user types.
- Each freelancer offers exactly one service at a fixed price and has a skill profile that evolves dynamically based on job completions and cancellations.

- Ratings are integers in $[0, 5]$, but the average rating may be fractional. Each newly registered freelancer is assumed to have **one rating of five stars**.
- **All prices and payments are integers** and remain integers after applying loyalty discounts.
- **Composite scores are integers** to avoid floating-point comparison issues.
- A freelancer can only be employed by one customer at a time and becomes unavailable until the employment is completed or cancelled.
- No employment tasks will remain unfinished until the end of the month.

3 Skill System

The platform uses a comprehensive skill system to match freelancers with appropriate services. Skills evolve dynamically based on job outcomes.

3.1 The Five Skill Categories

Each freelancer and each service is characterized by five skill dimensions, each with an integer value between 0 and 100:

1. **Technical Proficiency (T):** Technical skills and domain-specific knowledge required for the service
2. **Communication (C):** Ability to understand client requirements and provide clear updates
3. **Creativity (R):** Creative and problem-solving abilities needed for the service
4. **Efficiency (E):** How productively and quickly the freelancer completes tasks
5. **Attention to Detail (A):** Precision and quality focus in work execution

3.2 Skill Profiles

Both freelancers and services have skill profiles represented as:

$$\text{skillProfile} = (T, C, R, E, A)$$

These skill profiles represent different aspects for freelancers and services. For freelancers, the skill values indicate their proficiency level in each skill. For services, they indicate the importance or requirement level of that skill for the service. (i.e., a value of 0 means the skill is irrelevant, while a value of 100 means it is a crucial requirement for the service.)

3.3 Service Type Configuration

The system offers ten different types of services, each with a predefined skill requirement profile. These are:

Service Type	T	C	R	E	A
paint	70	60	50	85	90
web_dev	95	75	85	80	90
graphic_design	75	85	95	70	85
data_entry	50	50	30	95	95
tutoring	80	95	70	90	75
cleaning	40	60	40	90	85
writing	70	85	90	80	95
photography	85	80	90	75	90
plumbing	85	65	60	90	85
electrical	90	65	70	95	95

4 Dynamic Skill Evolution

Freelancer skills are not permanent, they change based on their job performance and behavior.

4.1 Mechanism 1: Skill Gains Following Job Completion

When a job is completed with `rating` ≥ 4, the freelancer gains skill points based on the service requirements.

4.1.1 Algorithm:

1. Identify the service's skill requirements: $(T_s, C_s, R_s, E_s, A_s)$
2. Find the **primary skill** - the skill with the highest value in the service profile
3. Find the **secondary skills** - the two skills with the second and third highest values
4. Apply gains:
   - Primary skill: +2 points
   - Secondary skills (both): +1 point each
   - Other skills: no change
5. Upper limit: No skill can exceed 100 points.

4.1.2 Examples

1. **Web Development Service**
   Service profile: web_dev = $(95, 75, 85, 80, 90)$
   Ranking: T=95 (primary), A=90 (secondary), R=85 (secondary), E=80, C=75
   If freelancer completes a web_dev job with rating 5:
   - Technical (T): +2

- Attention to Detail (A): +1
- Creativity (R): +1
- Efficiency (E): no change
- Communication (C): no change

2. **Painting Service**
   Service profile: paint = $(70, 60, 50, 85, 90)$
   Ranking: A=90 (primary), E=85 (secondary), T=70 (secondary), C=60, R=50
   If freelancer completes a painting job with rating 4:

   - Attention to Detail (A): +2
   - Efficiency (E): +1
   - Technical (T): +1
   - Communication (C): no change
   - Creativity (R): no change

Note: If rating < 4, no skill gains are awarded.

4.2 Mechanism 2: Skill Degradation Following Job Cancellation

When a freelancer cancels a job (freelancer-initiated cancellation), they suffer skill degradation across all dimensions.

Penalty:

- **All skills**: -3 points each
- **Minimum:** Skills cannot drop below 0

This represents the reputational damage and loss of practice from unreliable behavior.
Example:
Freelancer with skills $(80, 70, 65, 85, 90)$ cancels a job:
New skills: $(77, 67, 62, 82, 87)$

4.3 Tie-Breaker for Skill Rankings

When identifying primary and secondary skills for a service, if multiple skills have the same value, use following order of skill names for tie-breaking: T, C, R, E, A
Example: If both A and E are 90, E is ranked higher.

5 I/O Files

Each line in the input file represents a single instruction, followed by its parameters separated by whitespace. The program should read and execute these instructions in order, producing the corresponding outputs line by line in the output file.

Example Input

```
register_customer cust1
register_freelancer free1 paint 150 75 60 50 85 90
register_freelancer free2 paint 140 70 55 45 80 85
request_job cust1 paint 2
complete_and_rate free1 5
query_freelancer free1
cancel_by_frncer free2
query_freelancer free2
change_service free1 web_dev 200
simulate_month
query_freelancer free1
```

Example Output

```
registered customer cust1
registered freelancer free1
registered freelancer free2
available freelancers for paint (top 2):
free2 - composite: 7854, price: 140, rating: 0.0
free1 - composite: 7823, price: 150, rating: 0.0
auto-employed best freelancer: free2 for customer cust1
free1 completed job with rating 5
free1: paint, price: 150, rating: 5.0, completed: 1,
cancelled: 0, skills: (76,60,50,86,92), available: yes, burnout: no
cancelled by freelancer: free2
free2: paint, price: 140, rating: 0.0, completed: 0,
cancelled: 1, skills: (67,52,42,77,82), available: yes, burnout: no
service change for free1 queued from paint to web_dev
month complete
free1: web_dev, price: 200, rating: 5.0, completed: 0,
cancelled: 0, skills: (85,70,80,75,90), available: yes, burnout: no
```

6 Instructions

6.1 Customer Registration

Structure: `register_customer <customerID>`
Example: `register_customer cust1`

- Creates a new customer with the given ID.
- If the ID already exists (as a customer or freelancer), log:
  `Some error occurred in register_customer.`
- Upon success, log:
  `registered customer <customerID>`

6.2 Freelancer Registration

Structure: `register_freelancer <freelancerID> <service_type> <service_price> <T> <C> <R> <E> <A>`
Example: `register_freelancer free1 paint 150 75 60 50 85 90`

- Creates a new freelancer with the specified ID, service, price, and skill profile.
- Skills T, C, R, E, A must be integers in $[0, 100]$.
- Price must be a positive integer.
- If a freelancer or customer already uses this ID or any parameter is invalid, log:
  `Some error occurred in register_freelancer.`
- Upon success, log:
  `registered freelancer <freelancerID>`

6.3 Employing a Specific Freelancer

Structure: `employ <customerID> <freelancerID>`
Example: `employ cust1 free1`

- Employs the given freelancer if both the customer and freelancer exist, are not blacklisted, and the freelancer is available.
- Upon success, log:
  `<customerID> employed <freelancerID> for <service_type>`
- Otherwise, log:
  `Some error occurred in employ.`

6.4 Requesting a Job

Structure: `request_job <customerID> <service_type> <num_candidates>`
Example: `request_job cust1 paint 2`

- Ranks candidates based on integer composite score (see Section 7).
- Lists the top `num_candidates` of freelancers line by line in the log showing composite score and key metrics.
- **Automatically employs** the best-ranked candidate and logs:
  `auto-employed best freelancer: <freelancerID> for customer <customerID>`
- If the customer or the service does not exist log:
  `Some error occurred in request_job.`
- If there are no eligible freelancers log:
  `no freelancers available`

Example Input

```
request_job cust003 cleaning 3
```

Example Output

```
available freelancers for cleaning (top 3):
free546 - composite: 1250, price: 80, rating: 4.7
free378 - composite: 1180, price: 70, rating: 4.2
free982 - composite:  940, price: 85, rating: 3.8
auto-employed best freelancer: free546 for customer cust003
```

6.5 Cancelling an Employment

6.5.1 Customer-Initiated Cancellation

Structure: `cancel_by_customer <customerID> <freelancerID>`
Example: `cancel_by_customer cust1 free1`

- Cancels the active employment between the pair.
- Freelancer becomes available again.
- Customer loses loyalty points. This is done by excluding $250 from their total amount spent for each cancellation. Note that when querying the customer, you should still display the total amount spent **without applying this penalty**.
- If successful, log:
  `cancelled by customer:  <customerID> cancelled <freelancerID>`
- If the users don't exist, or such an employment was not made:
  `Some error occurred in cancel_by_customer.`

6.5.2 Freelancer-Initiated Cancellation

Structure: `cancel_by_freelancer <freelancerID>`
Example: `cancel_by_freelancer free1`

- Cancels the active employment of the freelancer.
- Freelancer becomes available again.
- A **zero-star review** (rating = 0) is applied to the freelancer's average.
- **Skill degradation**: All skills decrease by 3 points. (Skill points can't go below 0).
- If a freelancer cancels **five or more** employments within the same simulated month, they are **platform-blacklisted** and permanently removed from the pool of available freelancers.
- If successful, log:
  `cancelled by freelancer:  <freelancerID> cancelled <customerID>`

- Upon platform blacklisting, log:
  `platform banned freelancer: <freelancerID>`
- If the users don't exist, or such an employment was not made:
  `Some error occurred in cancel_by_freelancer`

6.6 Completing and Rating a Job

Structure: `complete_and_rate <freelancerID> <rating>`
Example: `complete_and_rate free1 5`

- Marks job as completed and updates rating.
- Rating update formula:

$$\text{new\_avg} = \frac{(\text{old\_avg} \times n) + \text{rating}}{n+1} \quad \text{where} \quad n = n_{\text{completed}} + n_{\text{cancelled}}$$

- If rating ≥ 4: Apply skill gains (see Section 4.1).
- Freelancer becomes available again.
- Upon success, log:
  `<freelancerID> completed job for <customerID> with rating <rating>`
- Errors (invalid rating, no active employment, etc.):
  `Some error occurred in complete_and_rate.`

6.7 Changing Service Type

Structure: `change_service <freelancerID> <new_service_type> <new_price>`
Example: `change_service free1 web_dev 200`

- Queues a service change request to be applied at the next `simulate_month`.
- Price must be a positive integer.
- Upon queuing successfully, log:
  `service change for <freelancerID> queued from <old_service> to <new_service>`
- Errors (log: `Some error occurred in change_service.`):
  - Freelancer does not exist
  - Service does not exist.

Application at Month-End:
When `simulate_month` is executed, all queued service changes are applied - for each freelancer, the service type, price, and skill profiles are updated.

6.8 Query Commands

Gives further information about users.

6.8.1 Query Freelancer

Structure: `query_freelancer <freelancerID>`
Example: `query_freelancer free1`

- Logs: service, price, rating, completed jobs, cancelled jobs, skills, availability, burnout status
- Format:
  `freelancerID: service, price: X, rating: Y, completed: Z, cancelled: W, skills: (T,C,R,E,A), available: yes/no, burnout: yes/no`
- If freelancer does not exist:
  `Some error occurred in query_freelancer.`

6.8.2 Query Customer

Structure: `query_customer <customerID>`
Example: `query_customer cust1`

- Logs: total spending (integer), loyalty tier, current employments, total employment count
- Format:
  `customerID: total spent: $X, loyalty tier: TIER, blacklisted freelancer count: B, total employment count: C`
- Total spending is the sum of all customer payments going back to initial registration (after loyalty discounts applied)
- Example output:
  `cust1: total spent: $1250, loyalty tier: SILVER, blacklisted freelancer count: 3, total employment count: 10`
- If customer does not exist:
  `Some error occurred in query_customer.`

6.9 Blacklist Management

Customers can maintain their blacklists if they wish to not receive any services from particular freelancers.

6.9.1 Blacklist a Freelancer

Structure: `blacklist <customerID> <freelancerID>`
Example: `blacklist cust1 free3`

- Adds the given freelancer to the customer's personal blacklist.
- Blacklisted freelancers will not appear among eligible candidates when the customer makes job requests.

- If the customer or the freelancer does not exist, log:
  `Some error occurred in blacklist.`
- If the freelancer is already blacklisted by that customer, log:
  `Some error occurred in blacklist.`
- Upon success, log:
  `<customerID> blacklisted <freelancerID>`

6.9.2 Unblacklist a Freelancer

Structure: `unblacklist <customerID> <freelancerID>`
Example: `unblacklist cust1 free3`

- Removes the freelancer from the customer's personal blacklist.
- Once unblacklisted, the freelancer can again be matched to the customer in future job requests.
- If the customer or the freelancer does not exist, log:
  `Some error occurred in unblacklist.`
- If the freelancer is not blacklisted by that customer, log:
  `Some error occurred in unblacklist.`
- Upon success, log:
  `<customerID> unblacklisted <freelancerID>`

6.10 Manual Freelancer Skill Updates

In addition to dynamically evolving skill profiles through job completions and cancellations, the system also allows for direct, real-time updates to a freelancer's skill profile via the `update_skill` instruction.
Structure: `update_skill <freelancerID> <T> <C> <R> <E> <A>`
Example: `update_skill free546 90 85 80 75 70`

- Updates the freelancer's skill profile across all five attributes.
- Each skill value will be an integer in the range $[0, 100]$.
- After updating, the freelancer's composite score is recalculated based on the skill requirements of their service.
- If the freelancer does not exist or any skill value is invalid, log:
  `Some error occurred in update_skill.`

Example Input

```
update_skill free546 90 85 80 75 70
```

Example Output

```
updated skills of free546 for cleaning
```

7 Composite Ranking System

When ranking freelancers for a job request, the system calculates an **integer composite score**. The freelancer with the highest composite score is the most desirable one for the job.

7.1 Integer Composite Score Formula

$$\text{compositeScore} = \lfloor 10000 \times (w_s \cdot \text{skillScore} + w_r \cdot \text{ratingScore} + w_l \cdot \text{reliabilityScore} - \text{burnoutPenalty}) \rfloor$$

where $\lfloor \cdot \rfloor$ denotes the floor function (round down to nearest integer).

Weights:

- $w_s = 0.55$ (skill matching)
- $w_r = 0.25$ (rating quality)
- $w_l = 0.20$ (reliability)

Burnout Penalty:

$$\text{burnoutPenalty} = \begin{cases} 0.45 & \text{if burnout status is TRUE} \\ 0.0 & \text{otherwise} \end{cases}$$

Score Range: $[0, 10000]$ (integer)

7.2 Component Scores (All Normalized to [0,1])

7.2.1 Skill Score (Weighted Average Score)

The skill score measures a freelancer's proficiency, weighted by the priorities of the service. It is normalized such that a freelancer with 100 in all skills will receive a perfect 1.0 score for any service.

The score is calculated as the dot product of the freelancer and service vectors, divided by the theoretical maximum score for that specific service (which would be achieved by a freelancer with 100 in all skills).

$$\text{skillScore} = \frac{\vec{F} \cdot \vec{S}}{100 \times \sum S_i}$$

where:

- $\vec{F} = (T_f, C_f, R_f, E_f, A_f)$ is the freelancer's skill vector
- $\vec{S} = (T_s, C_s, R_s, E_s, A_s)$ is the service requirement vector item $\vec{F} \cdot \vec{S} = (T_f \cdot T_s) + (C_f \cdot C_s) + (R_f \cdot R_s) + (E_f \cdot E_s) + (A_f \cdot A_s)$
- $\sum S_i = T_s + C_s + R_s + E_s + A_s$

7.2.2 Rating Score

$$\text{ratingScore} = \frac{\text{averageRating}}{5.0}$$

7.2.3 Reliability Score

$$\text{reliabilityScore} = \begin{cases} 1.0 & \text{if completedTotal + cancelledTotal = 0} \\ 1.0 - \frac{\text{cancelledTotal}}{\text{completedTotal+cancelledTotal}} & \text{otherwise} \end{cases}$$

7.3 Tie-Breaking

If two freelancers have identical integer composite scores, use lexicographically smaller freelancer ID.

8 Monthly Simulation

Structure: `simulate_month`
Advances the system by one month. Output:
`month complete`

8.1 Burnout System

Freelancers can experience burnout if they complete too many jobs. Because burnout can affect their health and job performance, the system deprioritizes freelancers who are burned out.

8.1.1 Trigger

If a freelancer completes a minimum of five jobs in a month and they weren't burned out from previous months, the system marks them as burned out.

8.1.2 Recovery

An already burned out freelancer can return to normal if they complete a maximum of two jobs in a month.

8.2 Loyalty System

Based on their all time spending, customers can get special discounts from the system. In this case, the freelancer's payment is unaffected as the system subsidizes the discount. Tiers determined by all-time total spending by the customer:

- **BRONZE:** \$0–\$499 (0% subsidy)
- **SILVER:** \$500–\$1,999 (5% subsidy)
- **GOLD:** \$2,000–\$4,999 (10% subsidy)
- **PLATINUM:** \$5,000+ (15% subsidy)

Integer Payment Formula:
To ensure all customer payments remain integers:

$$\text{customerPayment} = \lfloor \text{freelancerPrice} \times (1 - \text{subsidy}) \rfloor$$

Platform Subsidy Model:

- **Customer pays:** $\lfloor \text{freelancerPrice} \times (1 - \text{subsidy}) \rfloor$ (integer)
- **Freelancer receives:** `freelancerPrice` (always full price, integer)
- **Platform subsidizes:** freelancerPrice − customerPayment (the difference)

Examples:

- **BRONZE:** Price = 150 → Customer pays: $\lfloor 150 \times 1.0 \rfloor = 150$, Freelancer gets: 150, Platform pays: 0
- **SILVER:** Price = 150 → Customer pays: $\lfloor 150 \times 0.95 \rfloor = 142$, Freelancer gets: 150, Platform pays: 8
- **GOLD:** Price = 150 → Customer pays: $\lfloor 150 \times 0.90 \rfloor = 135$, Freelancer gets: 150, Platform pays: 15
- **PLATINUM:** Price = 150 → Customer pays: $\lfloor 150 \times 0.85 \rfloor = 127$, Freelancer gets: 150, Platform pays: 23
- **Edge Case:** Price = 147, SILVER → Customer pays: $\lfloor 147 \times 0.95 \rfloor = 139$, Freelancer gets: 147, Platform pays: 8

8.3 Service Changes

Apply all queued service changes from `change_service` commands, which means service type and skills have to be updated.

9 Submission

First, all of your source files should be collected under Project2/src. Name of your main class must be Main.java. Then, you should zip the Project2 folder and rename it to

$$\langle student\_id \rangle.zip$$

This zip file will be submitted through moodle. Make sure your code compiles and runs with following commands:

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

10 Grading

Grades will be evaluated based on three testcase types:

- **Type 1 - Core Operations (40 points):**
  - Registrations with `register_customer, register_freelancer`
  - Employments utilizing the `employ` instruction
  - Blacklisting

  - Cancellations by both parties
  - Job completions and rating updates using `complete_and_rate`
  - Query commands for customers and freelancers with `query_customer` or `query_freelancer`

- **Type 2 - Dynamic Simulation Operations (55 points):** Includes the instructions of Type 1 testcases (excluding `employ`), plus:
  - Finding the ideal freelancer using `request_job`
  - Service type changes using `change_service`
  - Monthly simulation effects with `simulate_month` (burnout, loyalty updates, and application of service-changes)

- **Type 3 - Advanced Round (5 points):** Includes the instructions of Type 2 testcases, plus `update_skill`

Type 1 and Type 2 will have their own small, medium and large test cases. For Type 3, there will only be the large test case. To receive full credit for a test case, your submission should finish execution no longer than twice the specified expected run time.

Table 1: Grading Distribution by Test Type and Size

Type	Small	Medium	Large	Type Total
Type 1	8%	12%	20%	**40%**
Type 2	12%	18%	25%	**55%**
Type 3	—	—	5%	**5%**
Size Total	**20%**	**30%**	**50%**	**100%**

10.1 Test Case Scaling

Test cases are categorized by size with the following specifications:

- **Small:** 2,000–10,000 users (freelancers + customers)
- **Medium:** 10,000–100,000 users (freelancers + customers)
- **Large:** 100,000–500,000 users (freelancers + customers)

10.2 Command Distribution by Test Type

Table 2 shows which commands are supported in each test type.

Table 2: Supported Command Syntax by Test Type

Command	Syntax	T1	T2	T3	Description
Register Customer	`register_customer <id>`	✓	✓	✓	Create customer
Register Freelancer	`register_freelancer <id> <srv> <pr> <T> <C> <R> <E> <A>`	✓	✓	✓	Create freelancer
Employ Freelancer	`employ_freelancer <custID> <freelID>`	✓	×	×	Manual employment
Request Job	`request_job <custID> <service> <K>`	×	✓	✓	Auto-employ best
Complete & Rate	`complete_and_rate <freelID> <rating>`	✓	✓	✓	Finish job
Cancel by Customer	`cancel_by_customer <custID> <freelID>`	✓	✓	✓	Customer cancels
Cancel by Freelancer	`cancel_by_freelancer <freelID>`	✓	✓	✓	Freelancer cancels
Blacklist	`blacklist <custID> <freelID>`	✓	✓	✓	Add to blacklist
Unblacklist	`unblacklist <custID> <freelID>`	✓	✓	✓	Remove blacklist
Change Service	`change_service <freelID> <srv> <pr>`	×	✓	✓	Queue change
Simulate Month	`simulate_month`	×	✓	✓	Month-end update
Query Freelancer	`query_freelancer <freelID>`	✓	✓	✓	Get freelancer info
Query Customer	`query_customer <custID>`	✓	✓	✓	Get customer info
Update Skill	`update_skill <freelID> <skill> <val>`	×	×	✓	Bonus: manual skill

11 Warnings

- This is an **individual** project.
- 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** at first attempt and disciplinary action in case of recurrence.
- Make sure you document your code with necessary inline comments and use meaningful variable names. Do not over-comment, or make your variable names unnecessarily long. This is very important for partial grading.
- You can use all libraries in java.io. You should implement your own data structures and classes for further needs. If you need a standard library, you should not

use it without asking us. You are allowed to use as a start: java.util.ArrayList, java.util.LinkedList.

- You are required to write and use appropriate data structures covered in lectures to achieve decent running times for large inputs. Don't forget that you may not get any points if your code takes so much time to complete. Therefore, using correct data structures is an integral part of this project.
- Use the Moodle forum designated for this project for any questions or clarifications.