Nice, presentation time 😎 I'll give you a**7-slide deck** with short bullet points + optional speaker notes so you can easily cover it in ~4 minutes.

🟦 Slide 1 — Title & Authors

Title: NAO Planning Challenge – Algorithm A (and Dance Execution)

Subtitle: Planning and executing a 2-minute choreography for NAO

Authors:

- Sad Salman — ssad.sadadan@studio.unibo.it
- Hafiz Mudahir Khan — hafizmdadd.khan@studio.unibo.it

Speaker notes (≈20s): "We'll briefly present how we solved the NAO Planning Challenge: how we planned a valid choreography (Algorithm A) and how we execute it on the robot with music."

🟦 Slide 2 — Problem & Constraints

Goal: Make NAO perform a **choreography** using given moves.

Constraints (from assignment):

- Must visit **mandatory positions P1…P8** in order (StandInit, Sit, WipeForehead, Hello, SitRelax, Stand, StandZero, Crouch)
- Insert **intermediate moves** between them
- **Total time ≤ 120 s**
- Use **≥ 5 intermediate moves** overall
- Planning must be **testable on simulated robot** (no real music there)

Speaker notes (≈30s): "Task: we must pass through 8 key poses in order, add intermediate moves, stay under 2 minutes and use at least 5 intermediates. The planner must work in simulation without music."

🟦 Slide 3 — Measuring Move Durations

Why: Algorithm A needs realistic **durations** for each move.

How we measured:

- Created a **separate Choregraphe box** for each move
- Set **Motor speed = 100%** (Preferences → General)
- In each box:
  - `start = time.time()` before motion
  - `end = time.time()` after motion
  - `self.logger.info("Move X: %.2f s" % (end - start))`
- Ran moves many times (Sit, SitRelax, Stand, etc.) and took **stable values**

Result:

- Built a Python dictionary `DUR = { "P1": 4.04, "Sit": 31.36, ..., "rotation_handgun": 11.87, ... }`

Speaker notes (≈30s): "First we made NAO perform each move alone and logged its execution time, so Algorithm A knows how long each block of the choreography really takes."

🟦 Slide 4 — Algorithm A (Greedy Planner)

Input:

- Mandatory sequence: `P1…P8`
- Intermediate moves (e.g. `rotation_handgun`, `union_arms`, `move_forward`, …)
- Durations in `DUR`

Idea:

- For each pair **Pi → P(i+1)** → choose exactly **one intermediate** in between: `Pi → [Intermediate] → P(i+1)`

Greedy choice criteria:

1. **Required moves first** (we want some specific moves to appear at least once, e.g. rotation_handgun, move_forward, union_arms, workout, guitar…)
2. Prefer moves with **lower usage so far** → avoids always picking the same one
3. Among those, choose the **shortest duration** → keeps total time small

Speaker notes (≈40s): "Algorithm A is greedy: between each pair of mandatory poses we pick one intermediate move based on three priorities: must-use moves, balanced usage, then duration. This guarantees variety and keeps us under the time limit."

Gotcha, let's make Slide 5 more "story" and less "list of moves".

Here's a revised **Slide 5 – Results of Algorithm A** that focuses on *properties* of the choreography instead of exact move names.

🖼️ Slide 5 — Results of Algorithm A

Choreography properties:

- **Duration:**
  - Planned total time ≈ **100–110 seconds**
  - Safely **below the 120 s** limit
- **Structure:**
  - Starts from a **neutral standing pose**
  - Includes **sitting**, **upper-body gestures**, **forward/diagonal steps**, and a **final crouch pose**
  - Every mandatory position (P1…P8) is visited **in order**
- **Use of intermediate moves:**
  - **At least 7** intermediate moves inserted
  - No single move is over-used → choreography looks **varied**, not repetitive
  - Specific "highlight" moves are forced to appear at least once (e.g. a rotation, a forward step, and a couple of expressive arm gestures)
- **Automatic validation (printed by `main.py`):**
  - `Time <= 120 s ? OK`
  - `#Intermediates >= 5 ? OK`
  - Summary of how many times each move was selected

(Optional speaker line, ~10s): "So Algorithm A doesn't just spit out a random sequence: it builds a dance that respects all constraints, stays under 2 minutes, uses enough intermediate moves, and still looks reasonably rich and varied."

🖼️ Slide 6 — Algorithm B: Execution with Music

(High-level description, not run in simulation)

Goal: execute planned moves **in sync with music** on real NAO.

Steps:

1. **Compute schedule**
   - Cumulative sum of `DUR[move]` → planned start time for each move
2. **Start music** with `ALAudioPlayer.playFileFromPosition`
   - Save `t_start_music = time.time()`
3. For each move in the plan:
   - Compute `planned_start = previous_durations_sum`
   - Compute `delay = t_start_music + planned_start - time.time()`
   - If `delay > 0` : `time.sleep(delay)`
   - Then start the corresponding **Choregraphe behavior** (StandInit, Sit, rotation_handgun, etc.)

Note: Music cannot be tested in the **simulator**, so Algorithm B is only for **real robot execution**.

Speaker notes (≈45s): "We execute the plan open-loop: we start the song, record its start time, and for each move we wait until the correct timestamp relative to the music, then trigger the corresponding behavior. This syncs the dance to the rhythm on the real robot."

🖼️ Slide 7 — Implementation & How to Run

Environment:

- **NAOUbuntu VM** (provided by course)
- **Choregraphe** with Virtual Robot NAO H25 (V50)
- **Python 3** for planning (no external libraries needed)

Repository:

- GitHub: `https://github.com/Saad-Salman101/Nao-Challenge.git`

Steps to run planner:

```
 sudo apt-get update
sudo apt-get install vlc git
git clone https://github.com/Saad-Salman101/Nao-Challenge.git
cd Nao-Challenge/robo_dance
python3 main.py
```

Choregraphe setup:

- Edit → Preferences → General → **Motor speed 100%**
- Virtual Robot → **NAO H25 (V50)**

Speaker notes (≈40s): "Everything is reproducible: clone the repo, run `main.py` to see the plan and constraints, then reproduce the same sequence of behaviors in Choregraphe on the virtual robot, and finally use Algorithm B with music on a real NAO."

Slide 8 — Conclusions & Possible Improvements

What we achieved:

- Collected **realistic timings** for all moves
- Implemented a **deterministic greedy Algorithm A**:
    - Respects 120 s limit
    - Uses ≥ 5 intermediate moves
    - Encourages **variety** and required moves
- Designed **Algorithm B** to execute the plan in sync with music on the real robot

Possible extensions:

- Use a more advanced **search algorithm** (e.g. A*, IDDFS) to optimize time or "smoothness"
- Include **beat detection** or manual keyframes to align specific moves to musical accents
- Add **feedback** from sensors (e.g. detect if a move took longer than expected and adjust timing on the fly)

Speaker notes (≈40s): "Overall we built a full pipeline: measure, plan, and execute the choreography. In the future we could use optimal search instead of greedy, align moves more tightly to music beats, and even correct timing online using feedback."

If you want, I can also turn this into a **slide-by-slide script** (exact sentences) so you can just read it and stay under 4 minutes.