

좋아요. 스스로 점검·복습할 수 있게 “질문 → 내가 한 답변 → 개선 답변(10초/30초/답다이브) → 추가 꼬리질문 대응 → 기억 포인트” 형식의 스크립트를 드릴게요. 발표 직전 리허설에도 그대로 쓰면 됩니다.

Q1. “빅-O가 왜 $n \log n$ 인가요?”(정렬 기준)

내 답변 요지

- “명단을 한 번씩 훑는데 $\log n$ 이고... sort가 최악 대비라서 그런 것 같다.”

개선 답변

- 10초: “비교 기반 정렬은 결정트리 하한 때문에 최소 비교 횟수가 $\Omega(n \log n)$ 이어서, 퀵/힙/병합 같은 일반 정렬은 평균/최악이 $n \log n$ 급입니다.”
- 30초: “비교 정렬은 결과가 $n!$ 가지인데, 이것 구분하려면 결정트리 높이가 $\log_2(n!) \approx n \log n$ 입니다. 그래서 표준 sort(퀵/힙/병합 계열)는 평균·최악이 $n \log n$ 입니다. 다만 키 범위가 작게 제한되면 카운팅/래딕 스처럼 비교를 안 쓰는 방식으로 $O(n + K)$ 도 가능합니다.”
- 답다이브: “우린 ‘우선순위 점수’를 비교해 정렬하므로 비교 기반 모델이 맞고, 따라서 하한 $\Omega(n \log n)$ 을 피할 수 없습니다. 정렬 외의 per-item 계산은 선형이어서 전체 병목은 정렬 단계입니다.”

꼬리질문에 이렇게

- “왜 퀵정렬은 평균 $n \log n$ 인데 최악 n^2 이라 하죠?” “피벗이 계속 한쪽에 몰리면 분할이 비대칭이 됩니다. 표준 라이브러리는 피벗 전략·하이브리드로 최악을 회피합니다.”
- “그럼 병합정렬?” “항상 $n \log n$ 이지만 추가 메모리 $O(n)$ 이 듭니다.”

기억 포인트

- 키워드: “결정트리 하한”, “비교 기반”, “ $n!$ 개의 순열 $\rightarrow \log(n!) \approx n \log n$ ”.

Q2. “서버 배포한 건가요?”

내 답변 요지

- “로컬호스트로 들어가니 배포는 못한 것 같다.”

개선 답변

- 10초: “아직 배포 안 했습니다. 로컬 Flask 개발 서버(127.0.0.1:5001)에서만 동작합니다.”
- 30초: “개발 단계라 `flask run`/`python run.py`로 로컬에서 확인합니다. 운영 배포를 하려면 WSGI 서버(gunicorn/uwsgi)+리버스 프록시(nginx) 또는 도커/클라우드에 올려 고정 URL과 SSL을 붙여야 합니다.”
- 답다이브: “현재는 파일 기반 I/O와 로컬 모델 파일(.pkl)에 의존합니다. 운영 배포 시엔 파일 경합 해결(락/원자치환) 또는 DB 전환(PostgreSQL)과 스토리지 경로 표준화가 선행되어야 합니다.”

꼬리질문에 이렇게

- “배포 계획은?” “단기: Dockerfile+gunicorn+nginx. 중기: DB 전환 후 클라우드(예: Render/EC2/Cloud Run)로 CI/CD 연결.”

기억 포인트

- 키워드: “로컬 개발 서버 ≠ 운영 배포”, “WSGI+nginx”, “Docker/CI/CD”, “파일 경합→DB”.

Q3. “ML이 머신러닝이면, 프로젝트에서 어떻게 사용했나요?”

내 답변 요지

- “피클 파일이 있고 pip install 했고, 엔진 파이썬에서 재가공된 데이터가 차트로... 그래서 ML 같다.”

개선 답변

- 10초: “ML은 ‘탐승확률’을 예측하는 데 씁니다. 학습된 모델을 .pkl로 저장해 로드하고, 엔진 결과에 확률이 없거나 비면 이 모델로 채웁니다.”
- 30초: “두 층입니다. ① C++의 베이스라인(가중합+시그모이드), ② Python의 학습 모델(.pkl, scikit-learn). 실행 시 `ensure_model()`이 모델을 로드/없으면 학습해 저장합니다. `load_waitlist()`가 엔진 산출을 읽고, 필요 시 ML로 확률을 보정합니다. Plotly는 시각화일 뿐 ML이 아닙니다.”
- 딥다이브: “특징량은 마일리지·가입일·대기열 위치 등으로 정규화 후 학습합니다. 예측은 DataFrame으로 받아 `predict_proba` 혹은 회귀 예측을 확률로 사용합니다. 모델은 joblib로 직렬화(.pkl)해 재사용합니다.”

꼬리질문에 이렇게

- “왜 굳이 C++ 베이스라인과 Python ML을 같이 써요?” “엔진 실패/모델 부재 시에도 ‘설명 가능한’ 최소 기능을 보장하려고 베이스라인을 넣었습니다. 반대로 품질을 높일 땐 ML을 사용합니다.”
- “학습은 어디서 언제?” “첫 실행 시 `ensure_model()`에서. 데이터가 바뀌면 재학습하고 새 .pkl로 교체합니다.”

기억 포인트

- 키워드: “확률 예측에 사용”, “.pkl은 모델 직렬화”, “Plotly는 시각화(ML 아님)”, “엔진 베이스라인 + Python ML”.
-

정리 답변(발표 때 바로 읽을 수 있는 버전)

1. 정렬이 왜 $n \log n$? “비교 기반 정렬은 결정트리 하한 때문에 최소 비교가 $\Omega(n \log n)$ 이라서 표준 sort가 $n \log n$ 입니다. 우린 ‘우선순위 점수’를 비교해 정렬하므로 이 하한을 따릅니다.”
 2. 서버 배포? “아직입니다. 로컬 Flask 개발 서버에서만 동작합니다. 배포하려면 gunicorn+nginx 또는 Docker/클라우드에 올려야 합니다.”
 3. ML 사용? “탐승확률 예측에 씁니다. 학습된 모델을 .pkl로 저장해 실행 시 로드하고, 엔진 산출에 확률이 없거나 비면 ML로 보정합니다. Plotly는 시각화 도구라 ML이 아닙니다.”
-

추가 피드백/복습 플랜 A. 핵심 코드 캡처 리스트(발표 슬라이드에 넣기)

- 우선순위 계산: `calculatePriority`
- 확률 계산(베이스라인): `predictBoardingProbability`
- 모델 준비/로드: `ensure_model`

- 산출 로드/보정: `load_waitlist`
- 정렬 호출부 및 비교 기준: `sortByPriority/comparator B`. 개념 스냅샷 4개
- 비교 정렬 결정트리 그림(왜 $n \log n$)
- 로컬 개발 서버 vs 운영 배포 아키텍처(WSGI+nginx)
- ML 파이프라인(학습→.pkl→로드→예측)
- 파일 경합→DB 전환 흐름도 C. 90분 셀프 실습(권장)
- scikit-learn으로 작은 분류/회귀 모델 학습→`joblib.dump()` 저장→샘플 입력으로 `joblib.load()` 예측→Plotly로 시각화.
- 정렬 실험: 난수 10만 개 벡터를 파이썬/넘파이/정렬 알고리즘(가능하면 C++도)로 타이밍 비교.

원하시면 위 캡처 포인트를 실제 코드 기준으로 표시해서 슬라이드 초안까지 만들어 드리겠습니다.