Review & Plan
Today’s objectives

- Review of list-update with a focus on advice
- An introduction to bin packing
How many bits of advice are sufficient to achieve an optimal solution for a sequence $\sigma$ of length $n$?

We show $\text{OPT}(\sigma) - n$ bits are sufficient.
Optimal Solution with Advice

- We learned earlier that an optimal algorithm only needs to make paid exchanges
  - We can be more specific.
- There is an optimal solution which only uses **subset transfers**
  - Before accessing an item $x$, use paid exchanges to move a subset of preceding items to just after $x$ (we skip the proof).

![Diagram](image)

- Guide an algorithm to maintain the same lists as $OPT$ by encoding the subset to be transferred after each access.
Optimal Solution with Advice

- **What is the advice?** Before each access to an item \( x \) at index \( i \), we use one bit of advice for each item \( y \) preceding \( x \) to indicate whether \( y \) should be transferred to after \( x \).

- **What is the size of advice?** \( \text{OPT} \) incurs a cost of at least \( i \) for accessing \( x \) (it might be more).
  - There will be \( i - 1 \) bits of advice.
  - The total advice size is no more than the cost of \( \text{cost}(\text{OPT}) - n \).
  - For a list of length \( m \), we have \( \text{cost}(\text{OPT}) - n > n \cdot m/2 \) (static offline), so for lists of constant length \( \text{cost}(\text{OPT}) \in O(n) \).

- **How the algorithm uses advice?** For each request to item \( x \), it transfers the indicated subsets in the advice to right after \( x \).

- **Why it is optimal?** the algorithm mimic \( \text{OPT} \).

**Theorem**

*For lists of constant length, advice of size \( O(n) \) bits is sufficient to achieve an optimal algorithm for any sequence of length \( n \).*
Optimal Solution with Advice

- How many bits of advice are **required** to achieve an optimal solution for a sequence $\sigma$ of length $n$?
  - We show advice of linear size is required.
Optimal Solution with Advice

Consider a sequence of phases of requests to items $a$, $b$ which form a list of length 2.

- Each phase involves 6 requests and has type 0 or 1.
- A type 0 phase has requests → bbbaaa and a type 1 phase has requests baabaa.

\[< bbbaaa \ baabaa \ bbbaaa \ bbbaaa \ bbbaaa \ baabaa >\]

- At the beginning of each phase, the list of any reasonable algorithm looks like $a \rightarrow b$
  - Otherwise, the algorithm can be improved to have that without increasing its cost.

- At the first request to $b$ in each phase, depending on its type, an optimal algorithm should move $b$ to front (type 0) or keep it at second position (type 1).
  - The algorithm should guess the type of each phase at the first request.
Optimal Solution with Advice

< b b b a a a  b a a b a a  b b b a a a  b b b a a a  b a a b a a >

- phases of type 0 (b b b a a a):
  - The cost of OPT is 2 + 1 + 1 + 2 + 1 + 1 = 8.
  - The cost of Alg is 8 if it guesses the type correctly, and 9 otherwise

- phases of type 2 (b a a b a a):
  - The cost of OPT is 2 + 1 + 1 + 2 + 1 + 1 = 8.
  - The cost of Alg is 8 if it guesses the type correctly, and 9 otherwise

- Assume there are k phases and the algorithm correctly guesses half of them:
  - From binary guessing lemma, we know that requires advice of size $O(k) = O(n)$
  - the cost of the algorithm will be at least $(k/2) \cdot 8 + (k/2) \cdot 9 = 8.5k$
  - the cost of OPT will be $k \cdot 8$.
  - the competitive ratio will be $\frac{8.5}{8} = 17/16$. 
We showed that, in order to achieve a competitive ratio better than 17/16, advice of size $\Omega(n)$ is required.

**Theorem**

*For lists of small sizes, advice of linear size is required and sufficient to achieve an optimal solution.*
Bin Packing Problem
Bin Packing Problem

- The input is a sequence of items of various sizes which are revealed in a sequential, online manner.
  - E.g., \(<9, 3, 8, 5, 1, 1, 3, 2, 4, 2, 4, 5, 5, 8, 6, 4, 5, ... >\).

- The goal is to pack these items into a minimum number of bins of uniform capacity.
Bin Packing Application

- Bins can be servers (with uniform capacity in terms of memory, bandwidth, cpu, etc.) & items can be jobs/files/data assigned to servers.

- Cutting stock: bins can be ‘cakes’ and ‘items’ can be kids with different amount of requests.
  - or ‘bottles of water’ with different amount of water requests

- Bins can be trucks of uniform weight capacity and items can be commodities of different weights to be moved between two cities
Next Fit Algorithm

- Keep one open bin at each time:
  - If the open bin has enough space, use it; otherwise, close it and open a new bin
  - A closed bin is never referred again.

\[< 9 \ 3 \ 8 \ 5 \ 1 \ 1 \ 3 \ 2 \ 4 \ 2 \ 4 \ 5 \ 5 \ 8 \ 6 \ 4 \ 5 >\]
First Fit Algorithm

- Find the first bin which has enough space for the item, and place the item there.
- Open a new bin if such bin does not exist.

< 9 3 8 5 1 1 3 2 4 2 4 5 5 8 6 4 5 >
Best Fit Algorithm

- Find the bin with minimum left capacity which has enough space for the item, and place the item there

\[
< 9 \ 3 \ 8 \ 5 \ 1 \ 1 \ 3 \ 2 \ 4 \ 2 \ 4 \ 5 \ 5 \ 8 \ 6 \ 4 \ 5 >
\]

- The opposite of Best Fit is Worst Fit which places the item in the bin with maximum left capacity.
Competitive Analysis

- Compare the performance of an online algorithm $A$ with an optimal offline algorithm $\text{OPT}$:
  - $\text{OPT}$ knows the whole sequence in the beginning.
- Competitive ratio of $A$ is the maximum value of $A(\sigma)/\text{OPT}(\sigma)$ among all sequences $\sigma$.
  - We are interested in the asymptomatic competitive ratio where $\text{OPT}(\sigma)$ is arbitrary large.
- $\text{OPT}(\sigma) \geq S(\sigma)$, where $S(\sigma)$ is the total size of items in $\sigma$. 
Competitive Analysis

- In the next class, we see that Next Fit is 2-competitive (again, with respect to asymptotic competitive ratio)
- We also visit an ‘easy’ lower bound argument that shows no online algorithm can achieve a competitive ratio better than 4/3
  - This bound can be greatly improved