List Update Problem
List Accessing Problem

- The input is a set of requests to items in a list.
- The cost of accessing an item in index $i$ is $i$.

< d b b d c a c >
List Accessing Problem

- The input is a set of *requests* to items in a list.
- The cost of accessing an item in index $i$ is $i$.

```
< d b b d c a c >
cost: 4
```

Diagram:
```
a → b → c → d → e
```
Problem Statement

List Accessing Problem

- The input is a set of requests to items in a list.
- The cost of accessing an item in index $i$ is $i$.

$\langle d\ b\ b\ d\ c\ a\ c \rangle$

Cost: $4+2$
The input is a set of requests to items in a list.
The cost of accessing an item in index $i$ is $i$.

List: `<d b b d c a c>`
Cost: $4 + 2 + 2$
Problem Statement

List Accessing Problem

- The input is a set of requests to items in a list.
- The cost of accessing an item in index $i$ is $i$.

$< d \ b \ b \ d \ c \ a \ c >$

cost: $4 + 2 + 2 + 4$

\[
\begin{array}{c}
\text{a} \\
\rightarrow \\
\text{b} \\
\rightarrow \\
\text{c} \\
\rightarrow \\
\text{d} \\
\rightarrow \\
\text{e}
\end{array}
\]
List Accessing Problem

- The input is a set of requests to items in a list.
- The cost of accessing an item in index $i$ is $i$.

$\langle \text{d b b d c a c} \rangle$

Cost: $4 + 2 + 2 + 4 + 3$

![Diagram of a list with items a, b, c, d, e and arrows connecting them. The cost calculation is shown.]
Problem Statement

List Accessing Problem

- The input is a set of requests to items in a list.
- The cost of accessing an item in index $i$ is $i$.

\[
< \text{d b b d c a c} > \\
\text{cost: } 4 + 2 + 2 + 4 + 3 + 1
\]
List Accessing Problem

- The input is a set of *requests* to items in a list.
- The cost of accessing an item in index $i$ is $i$.

\[
< d \ b \ b \ d \ c \ a \ c >
\]
\[
\text{cost: } 4+2+2+4+3+1+3 = 19
\]
An instance of **self-adjusting data structures**.

The structure adjusts itself based on the input queries.
An instance of **self-adjusting data structures**.

The structure adjusts itself based on the input queries.

List update was formulated in 1984 by Sleator and Tarjan

- This result of Sleator and Tarjan made online algorithms popular in the following two decades
- There are applications in data-compression!
Problem Statement

Self-Adjusting Lists

- Update a list of length $k$ to adjust it to the patterns in the input sequence of length $n$ ($n \gg k$).
Problem Statement

Self-Adjusting Lists

Update a list of length $k$ to adjust it to the patterns in the input sequence of length $n$ ($n \gg k$).

- Free exchanges: Move a requested item closer to the front without any cost.

< d b b d c a c >

cost: 4
Problem Statement

Self-Adjusting Lists

- Update a list of length $k$ to adjust it to the patterns in the input sequence of length $n$ ($n \gg k$).
  - Free exchanges: Move a requested item closer to the front without any cost.

```
< d b b d e a c >
cost: 4
```

```
< d b b d c a c >
cost: 4
```
Problem Statement

Self-Adjusting Lists

Update a list of length \(k\) to adjust it to the patterns in the input sequence of length \(n\) \((n \gg k)\).

- Free exchanges: Move a requested item closer to the front without any cost.
- Paid exchanges: Swap positions of two consecutive items with a cost 1.

Example:

\[
< d\ b\ b\ d\ c\ a\ c >
\]

cost: 4
Problem Statement

Self-Adjusting Lists

Update a list of length $k$ to adjust it to the patterns in the input sequence of length $n$ ($n \gg k$).

- Free exchanges: Move a requested item closer to the front without any cost.
- Paid exchanges: Swap positions of two consecutive items with a cost 1.

< d b b d c a c >

cost: 4
In the offline version of the problem, you have access to the whole set at the beginning.

- The problem is NP-hard.
Problem Statement

List Update Problem

- In the offline version of the problem, you have access to the whole set at the beginning.
  - The problem is NP-hard.
- In the online setting, the requests appear in an online, sequential manner.
  - An online algorithm should reorder the list without looking at the future requests.
Move-To-Front (MTF)

- After each access, move the requested item to the front.
- It only uses free exchanges.

```
< d b b d c a c >
cost: 4
```

```
a b c d e
< d b b d c a c >
cost: 4
```
Online Algorithms for List Update

Move-To-Front (MTF)

- After each access, move the requested item to the front.
- It only uses free exchanges.

\[
< d \ b \ b \ d \ c \ a \ c >
\]

Cost: 4
Online Algorithms for List Update

Move-To-Front (MTF)

- After each access, move the requested item to the front.
- It only uses free exchanges.

< d b b d c a c >

cost: 4 + 3
Online Algorithms for List Update

Move-To-Front (MTF)

- After each access, move the requested item to the front.
- It only uses free exchanges.

\[
\langle d\ b\ b\ d\ c\ a\ c \rangle
\]

\[
\text{cost: } 4 + 3
\]
Move-To-Front (MTF)

- After each access, move the requested item to the front.
  - It only uses free exchanges.

< d b b d c a c >

cost: 4+3+1
Move-To-Front (MTF)

- After each access, move the requested item to the front.
- It only uses free exchanges.

\[
< d \ b \ b \ d \ c \ a \ c >
\]

\[\text{cost: } 4 + 3 + 1 + 2\]
Move-To-Front (MTF)

- After each access, move the requested item to the front.
- It only uses free exchanges.

< d b b d c a c >

cost: 4+3+1+2+4
Online Algorithms for List Update

Move-To-Front (MTF)

- After each access, move the requested item to the front.
  - It only uses free exchanges.

\[
< d \ b \ b \ d \ c \ a \ c >
\]

Cost: \(4+3+1+2+4+4\)
Online Algorithms for List Update

Move-To-Front (MTF)

- After each access, move the requested item to the front.
- It only uses free exchanges.

\[
< d \ b \ b \ d \ c \ a \ c > \\
\text{cost: 4+3+1+2+4+4+2}
\]
Online Algorithms for List Update

Move-To-Front (MTF)

- After each access, move the requested item to the front.
- It only uses free exchanges.

\[
< \text{d b b d c a c} >
\]

\[
\text{cost: } 4 + 3 + 1 + 2 + 4 + 4 + 2
\]
After an access to $x$, move $x$ to the front of the first item $y$ which has been requested at most once since the last access to $x$.

Do nothing if such an item $y$ does not exist.

< d b b d c a c >

cost: 4
After an access to $x$, move $x$ to the front of the first item $y$ which has been requested at most once since the last access to $x$.

- Do nothing if such an item $y$ does not exist.

$\langle d\ b\ b\ d\ c\ a\ c \rangle$

\text{cost: } 4+2
Online Algorithms

After an access to $x$, move $x$ to the front of the first item $y$ which has been requested at most once since the last access to $x$.

- Do nothing if such an item $y$ does not exist.

\[
< \text{d b b d c a c} > \\
\text{cost: } 4 + 2 + 2
\]
After an access to $x$, move $x$ to the front of the first item $y$ which has been requested at most once since the last access to $x$.

Do nothing if such an item $y$ does not exist.

\[ \langle d \ b \ b \ d \ c \ a \ c \rangle \]

cost: 4+2+2
After an access to $x$, move $x$ to the front of the first item $y$ which has been requested at most once since the last access to $x$.

- Do nothing if such an item $y$ does not exist.

$\langle d\ b\ b\ d\ c\ a\ c\ \rangle$

cost: $4+2+2+4$
After an access to $x$, move $x$ to the front of the first item $y$ which has been requested at most once since the last access to $x$.

- Do nothing if such an item $y$ does not exist.

\[
\langle d, b, b, d, c, a, c \rangle \\
\text{cost: } 4+2+2+4
\]
After an access to $x$, move $x$ to the front of the first item $y$ which has been requested at most once since the last access to $x$.

- Do nothing if such an item $y$ does not exist.

```
< d b b d c a c >
cost: 4+2+2+4+4
```
After an access to \( x \), move \( x \) to the front of the first item \( y \) which has been requested at most once since the last access to \( x \).

- Do nothing if such an item \( y \) does not exist.

\[
\langle d \ b \ b \ d \ c \ a \ c \rangle
\]

\text{cost:} \quad 4+2+2+4+4+3
After an access to $x$, move $x$ to the front of the first item $y$ which has been requested at most once since the last access to $x$.

Do nothing if such an item $y$ does not exist.

$< d \ b \ b \ d \ c \ a \ c >$

Cost: $4+2+2+4+4+3+4$
After an access to $x$, move $x$ to the front of the first item $y$ which has been requested at most once since the last access to $x$.

- Do nothing if such an item $y$ does not exist.

```
< d b b d c a c >
cost: 4+2+2+4+4+3+4
```
Online Algorithms

Optimal Static Algorithm

- Look at the sequence of requests, sort items by the frequency of their accesses.
  - The most accessed item will be at the beginning of the list.

The cost of the algorithm would be at most $nk/2$. 

COMP 7720 - Online Algorithms

List Update & Compression
Online Algorithms

Optimal Static Algorithm

- Look at the sequence of requests, sort items by the frequency of their accesses.
  - The most accessed item will be at the beginning of the list.
- The cost of the algorithm would be at most $nk/2$. 
Consider a **cruel** sequence in which the adversary always asks for the last item in the list!

What will be the cost of the algorithm?
Consider a **cruel** sequence in which the adversary always asks for the last item in the list!

What will be the cost of the algorithm?

- It will be $nk$. 
Consider a **cruel** sequence in which the adversary always asks for the last item in the list!

What will be the cost of the algorithm?

- It will be \( nk \).

What is the cost of \( \text{OPT} \)?

- We know the optimal static algorithm has a cost of \( n(k + 1)/2 \).
- So the cost of \( \text{OPT} \) is no more than \( n(k + 1)/2 \).
Consider a **cruel** sequence in which the adversary always asks for the last item in the list!

What will be the cost of the algorithm?
- It will be $nk$.

What is the cost of $\text{OPT}$?
- We know the optimal static algorithm has a cost of $n(k + 1)/2$.
- So the cost of $\text{OPT}$ is no more than $n(k + 1)/2$.

The competitive ratio of any online list update algorithm is at least
\[
\frac{nk}{nk/2} = 2.
\]
Competitiveness of MTF
On the Nature of Opt

- There is an optimal algorithm that only uses paid exchanges!
On the Nature of Opt

There is an optimal algorithm that only uses paid exchanges!

Assume $OPT$ uses a free exchange after accessing item $x$ at position $i$ to move it closer to the front to position $j$

- The cost will be $i$.

![Diagram showing the movement of item $x$ in a list with two access points at $i = 3$ and $i = 5$.]
There is an optimal algorithm that only uses paid exchanges!

Assume $OPT$ uses a free exchange after accessing item $x$ at position $i$ to move it closer to the front to position $j$.

- The cost will be $i$.

In a new scheme, **before the access** apply $i - j$ paid exchanges to move $i$ to position $j$.

- The new cost will be $i - j$ for paid exchanges and $j$ for the access, which sums to $i$. 
Theorem

*Move-To-Front has competitive ratio of 2.*

- We prove it through potential function method
  - And it takes a few slides :'-)
At a given time, two items \( x \) and \( y \) form an **inversion** if their relative order is different in the lists of MTF and OPT.

**Question:** what is the maximum number of inversions for a list of length \( k \)?

- (a) \( k/2 \)
- (c) \( k(k - 1)/2 \)
- (c) \( k \)
- (d) \( k^2 - k/2 \)
Competitiveness of MTF

Potential Function

- Assume MTF and $OPT$ are running the same input in parallel.
- Assume we are at the $t$'th request, and there is a request to an item $x$.
- Define the potential at time $t$ to be the total number of inversions before accessing $x$. 

Inversions are $(b, c)$, $(b, d)$, $(b, e)$, $(c, d)$. So, $\Phi(t) = 4$. 

COMP 7720 - Online Algorithms
List Update & Compression
Assume MTF and OPT are running the same input in parallel.

Assume we are at the $t$'th request, and there is a request to an item $x$.

Define the potential at time $t$ to be the total number of inversions before accessing $x$.

Inversions are $(b, c), (b, d), (b, e), (c, d)$.

So, $\Phi(t) = 4$. 

- Inversions are $(b, c), (b, d), (b, e), (c, d)$.
- So, $\Phi(t) = 4$. 

COMP 7720 - Online Algorithms  List Update & Compression
Intuitively, if we have a high potential, we are in bad state.

Amortized Cost

- Define the amortized cost at time $t$ (when answering the $t$th request) as:
  \[
  \text{amortized cost}(t) = \text{actual cost}(t) + \Phi(t+1) - \Phi(t)
  \]

- Example: assume at time $t$, there is a request to $b$, and Opt does not rearrange the list for accessing $t$.
- For MTF, we have $\text{actual cost}(t) = 5$, $-\Phi(t) = 4$ and $\Phi(t+1) = 2$.
- Amortized cost is $5 + 2 - 4 = 3$. 
Amortized Cost

- Intuitively, if we have a high potential, we are in bad state.
- Define the amortized cost at time $t$ (when answering the $t$th request) as:

$$\text{amortized} \_ \text{cost}(t) = \text{actual} \_ \text{cost}(t) + \Phi(t+1) - \Phi(t)$$
Intuitively, if we have a high potential, we are in bad state.

Define the amortized cost at time $t$ (when answering the $t$th request) as:

$$amortized\_cost(t) = actual\_cost(t) + \Phi(t + 1) - \Phi(t)$$

Example: assume at time $t$, there is a request to $b$, and $OPT$ does not rearrange the list for accessing $t$. 

![Diagram of MTF and Opt lists]

- List of MTF: $a \rightarrow d \rightarrow c \rightarrow e \rightarrow b$
- List of Opt: $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$
Amortized Cost

- Intuitively, if we have a high potential, we are in bad state.
- Define the **amortized cost** at time \( t \) (when answering the \( t \)th request) as:

\[
amortized\_cost(t) = \text{actual\_cost}(t) + \Phi(t + 1) - \Phi(t)
\]

- Example: assume at time \( t \), there is a request to \( b \), and \( \text{OPT} \) does not rearrange the list for accessing \( t \).

![Diagram of MTF and Opt lists](image)
Competitiveness of MTF

Amortized Cost

- Intuitively, if we have a high potential, we are in bad state.
- Define the **amortized cost** at time $t$ (when answering the $t$th request) as:

  $$\text{amortized cost}(t) = \text{actual cost}(t) + \Phi(t + 1) - \Phi(t)$$

- Example: assume at time $t$, there is a request to $b$, and OPT does not rearrange the list for accessing $t$.
  - For MTF, we have $\text{actual cost}(t) = 5$, $-\Phi(t) = 4$ and $\Phi(t + 1) = 2$.
  - $\text{amortized cost}$ is $5 + 2 - 4 = 3$. 

\[ \text{list of MTF} \begin{array}{cccc} t & a & d & c & e & b \end{array} \]

\[ \begin{array}{cccc} t+1 & b & a & d & c & e \end{array} \]

\[ \text{list of Opt} \begin{array}{cccc} t & a & b & c & d & e \end{array} \]

\[ \begin{array}{cccc} t+1 & a & b & c & d & e \end{array} \]
Lemma

At any time $t$, $\text{amortized\_cost}(t) \leq 2 \cdot \text{OPT}(t)$, i.e., the amortized cost of MTF for the $t$'th request is at most twice that of OPT.
How many inversions are removed by moving $x$ to front?

- Before moving to front, there are $i-1$ items before $x$ in MTF list.
- At most $j-1$ of them can also appear before $x$ in $\text{OPT}$ list (are non-inversions) $\Rightarrow$ the rest, at least, $i-1-(j-1) = i-j$ are inversions $\Rightarrow$ By moving to front at least $i-j$ inversions are removed.
How many inversions are added by moving $x$ to front?

- $x$ is in front of MTF list after the move and at position $j$ of $O^P$'s list
- items that appear after $x$ in MTF and before $x$ in $O^P$ are at most $j - 1$
- At most $j - 1$ inversions are added
When moving $x$ to front:

- Actual cost is $i$, at least $i - j$ inversions are removed, at most $j - 1$ inversions are added
Potential Function Method (cntd.)

When moving $x$ to front:

- Actual cost is $i$, at least $i - j$ inversions are removed, at most $j - 1$ inversions are added.

Assume $\text{OPT}$ makes $k$ paid exchanges.

- Recall that it does no free exchange.
- The cost of $\text{OPT}$ will be $j + k$.
- Each paid exchange increases potential by 1 $\rightarrow$ potential increases by at most $k$. 

\[ \Phi(t + 1) - \Phi(t) = \text{added inversions} - \text{removed inversions} \leq (j + k - 1) - (i - j) = 2j + k - i - 1. \]

Amortized cost $= \text{actual cost} + \Phi(t + 1) - \Phi(t) \leq i + 2j + k - i - 1 = 2j + k - 1.$

Cost of $\text{OPT}$ is $j + k$ and amortized cost is less than $2j + k$. 

Lemma: At any time $t$, amortized cost $\Phi(t) < 2 \text{OPT}(t)$. 

When moving $x$ to front:
- Actual cost is $i$, at least $i - j$ inversions are removed, at most $j - 1$ inversions are added.

Assume $\text{OPT}$ makes $k$ paid exchanges.
- Recall that it does no free exchange.
- The cost of $\text{OPT}$ will be $j + k$.
- Each paid exchange increases potential by 1 $\rightarrow$ potential increases by at most $k$.

$$\Phi(t + 1) - \Phi(t) = \text{added inversions} - \text{removed inversions} \leq (j + k - 1) - (i - j) = 2j + k - i - 1.$$
Competitiveness of MTF

Potential Function Method (cntd.)

- When moving $x$ to front:
  - Actual cost is $i$, at least $i - j$ inversions are removed, at most $j - 1$ inversions are added

- Assume $\text{OPT}$ makes $k$ paid exchanges.
  - Recall that it does no free exchange.
  - The cost of $\text{OPT}$ will be $j + k$.
  - Each paid exchange increases potential by 1 $\rightarrow$ potential increases by at most $k$.

- $\Phi(t + 1) - \Phi(t) = \text{added inversions} - \text{removed inversions} \leq (j + k - 1) - (i - j) = 2j + k - i - 1$.

- $\text{amortized cost} = \text{actual cost} + \Phi(t + 1) - \Phi(t) \leq i + 2j + k - i - 1 = 2j + k - 1$
Competitiveness of MTF

Potential Function Method (cntd.)

- When moving $x$ to front:
  - Actual cost is $i$, at least $i - j$ inversions are removed, at most $j - 1$ inversions are added
- Assume $\text{OPT}$ makes $k$ paid exchanges.
  - Recall that it does no free exchange.
  - The cost of $\text{OPT}$ will be $j + k$.
  - Each paid exchange increases potential by 1 → potential increases by at most $k$.

- $\Phi(t + 1) - \Phi(t) = \text{added inversions} - \text{removed inversions} \leq (j + k - 1) - (i - j) = 2j + k - i - 1$.
- $\text{amortized cost} = \text{actual cost} + \Phi(t + 1) - \Phi(t) \leq i + 2j + k - i - 1 = 2j + k - 1$
- Cost of $\text{Opt}$ is $j + k$ and $\text{amortized cost}$ is less than $2j + k$.

Lemma

At any time $t$, $\text{amortized cost}(t) < 2 \text{OPT}(t)$. 
A quick example

Assume at time $t$:

- the list of MTF is
  
  $8 \to 7 \to 6 \to 5 \to 4 \to 3 \to 2 \to 1$

- the list of OPT is
  
  $1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 7 \to 8$

Assume $x$ is 3, which means $i = 6$ and $j = 3$.

The number of removed inversions in this case is at least $i - j = 3$. In fact, it turns out to be 5 because all $1, 2, 4, 5, 6, 7, 8$ form inversions with 3 which will be removed by moving 3 to the front.

The number of new inversions will be at most $j - 1 = 2$. In fact, it is 0 as no new inversion is added.
For the cost of MTF, we have

\[ MTF = \text{actual}\_\text{cost}(1) + \text{actual}\_\text{cost}(2) + \ldots + \text{actual}\_\text{cost}(n) \]
\[ = (\text{actual}\_\text{cost}(1) + \Phi(2) - \Phi(1)) \]
\[ + (\text{actual}\_\text{cost}(2) + \Phi(3) - \Phi(2)) \]
\[ + \ldots \]
\[ + (\text{actual}\_\text{cost}(n) + \Phi(n + 1) - \Phi(n)) - (\Phi(n + 1) - \Phi(1)) \]
\[ = \text{amortized}\_\text{cost}(1) + \ldots + \text{amortized}\_\text{cost}(n) - (\Phi(n + 1) - \Phi(1)) \]
\[ < 2 \text{Opt}(1) + \ldots + 2 \text{OPT}(n) - \mathcal{O}(L^2) \approx 2 \text{Cost}\_\text{Opt}(n) \]

{recall that \( n \gg L \)}

Note that in the second line, we just added and removed values (i.e., we added \( \Phi(1) - \Phi(1) + \Phi(2) - \Phi(2) + \ldots + \Phi(n + 1) - \Phi(n + 1) = 0 \)).
Theorem

Competitive ratio of $\text{MTF}$ is at most 2

- No deterministic algorithm can have a competitive ratio better than 2.
  - MTF is an optimal list-update algorithm.
  - Timestamp is another optimal deterministic algorithm.
Competitiveness of MTF

Theorem

**Competitive ratio of MTF is at most 2**

- No deterministic algorithm can have a competitive ratio better than 2.
  - MTF is an optimal list-update algorithm.
  - Timestamp is another optimal deterministic algorithm.
- Later, we will see randomized algorithms that achieve better competitive ratios.
Theorem

*Competitive ratio of MTF is at most 2*

- No deterministic algorithm can have a competitive ratio better than 2.
  - MTF is an optimal list-update algorithm.
  - Timestamp is another optimal deterministic algorithm.
- Later, we will see randomized algorithms that achieve better competitive ratios.
- Potential function method is a general framework for analysis of many online algorithms!
Assume you face an online problem where the input is a sequence of requests that require you to change the state of a problem.

These states should be finite and independent of the input length!
Assume you face an online problem where the input is a sequence of requests that require you to change the state of a problem.

- These states should be finite and independent of the input length!

Define a ‘potential’ as a function of the state of the algorithm and that of $OPT$ (e.g. no. inversions).

- This is the critical part :-)}
Assume you face an online problem where the input is a sequence of requests that require you to change the state of a problem.

- These states should be finite and independent of the input length!

Define a ‘potential’ as a function of the state of the algorithm and that of $\text{OPT}$ (e.g. no. inversions).

- This is the critical part :-)

Define the amortized cost at a given time $t$ as the actual cost algorithm plus the difference in potential after the request is served (same for all problems).
Assume you face an online problem where the input is a sequence of requests that require you to change the state of a problem.

- These states should be finite and independent of the input length!

Define a ‘potential’ as a function of the state of the algorithm and that of $\text{OPT}$ (e.g. no. inversions).

- This is the critical part :-) 

Define the amortized cost at a given time $t$ as the actual cost algorithm plus the difference in potential after the request is served (same for all problems).

The potential should be defined in a way so that you can show $\text{amortized\_cost}(t) \leq c \text{OPT}(t)$. 


Assume you face an online problem where the input is a sequence of requests that require you to change the state of a problem.

- These states should be finite and independent of the input length!
- Define a ‘potential’ as a function of the state of the algorithm and that of \( \text{OPT} \) (e.g. no. inversions).
  - This is the critical part :-)
- Define the amortized cost at a given time \( t \) as the actual cost algorithm plus the difference in potential after the request is served (same for all problems).

The potential should be defined in a way so that you can show

\[
\text{amortized cost}(t) \leq c \text{OPT}(t).
\]

Using a telescopic sum, the competitive ratio will be at most \( c \) (same for all problems).
Competitiveness of MTF

Potential Function Method

Did we survive?
One important application of list update is in data compression.

Given a data-sequence (e.g., an English text), we want to compress it.

We should be able to recover the exact text from the compressed one.
MTF Example

Solution 1: write the ASCII or Unicode code for each character
MTF Example

- Solution 1: write the ASCII or Unicode code for each character
- Use MTF index to encode the characters

\[
S = \text{INEFFICIENCIES}
\]

\[
C = 8
\]
MTF Example

- Solution 1: write the ASCII or Unicode code for each character
- Use MTF index to encode the characters

\[ S = \text{INEFFICIENCIES} \]
\[ C = 8 \ 13 \]
Competitiveness of MTF

MTF Example

- Solution 1: write the ASCII or Unicode code for each character
- Use MTF index to encode the characters

\[ S = \text{INEFFICIENCIES} \]

\[ C = 8 \ 13 \ 6 \]
MTF Example

- Solution 1: write the ASCII or Unicode code for each character
- Use MTF index to encode the characters

\[ S = \text{INEFFICIENCIES} \]
\[ C = 8 \ 13 \ 6 \ 7 \]
MTF Example

- Solution 1: write the ASCII or Unicode code for each character
- Use MTF index to encode the characters

\[ S = \text{INEFFICIENCIES} \]
\[ C = 8 \ 13 \ 6 \ 7 \ 0 \]
MTF Example

- Solution 1: write the ASCII or Unicode code for each character
- Use MTF index to encode the characters

\[ S = \text{INEFFICIENCIES} \]
\[ C = 8 \ 13 \ 6 \ 7 \ 0 \ 3 \]
MTF Example

- Solution 1: write the ASCII or Unicode code for each character
- Use MTF index to encode the characters

\[
S = \text{INEFFICIENCIES}
\]

\[
C = 8\ 13\ 6\ 7\ 0\ 3\ 6
\]
## MTF Example

- **Solution 1:** write the ASCII or Unicode code for each character
- **Use MTF index to encode the characters**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| C | I | F | E | N | A | B | D | G | H | J | K | L | M | O | P | Q | R | S | T | U | V | W | X | Y | Z |

$$S = \text{INEFFICIENCIES}$$

$$C = 8\ 13\ 6\ 7\ 0\ 3\ 6\ 1$$
Competitiveness of MTF

MTF Example

- Solution 1: write the ASCII or Unicode code for each character
- Use MTF index to encode the characters

\[
\begin{align*}
S &= \text{INEFFICIENCIES} \\
C &= 8 13 6 7 0 3 6 1 3 4 3 3 3 18
\end{align*}
\]

- What does a run in \( S \) encode to in \( C \)?
- This results in good compression if we have high \textit{locality} in the input.
Increase locality using Burrows-Wheeler Transform!
Competitiveness of MTF

Burrows-Wheeler Transform

- Increase locality using Burrows-Wheeler Transform!

- How it works?
  - Create all rotations of a given sequence.
  - Sort those rotations into lexicographic order.
  - Take as output the last column!

Why it is useful?
- Creates output with high locality!
- This is reversible
Increase locality using Burrows-Wheeler Transform!

How it works?
- Create all rotations of a given sequence.
- Sort those rotations into lexicographic order.
- Take as output the last column!

Why it is useful?
- Creates output with high locality!
- This is reversible

BWT(banana) = annb$aa
Assume we want to compress a data sequence $S$:

- Apply BWT on $S$ to increase its locality
Assume we want to compress a data sequence $S$:

- Apply BWT on $S$ to increase its locality
- Apply MTF on BWT output and encode the indices in the list
  - You expect to see a lot of 1’s and 2’s.
Competitiveness of MTF

B-Zip2 compression scheme

- Assume we want to compress a data sequence $S$:
  - Apply BWT on $S$ to increase its locality
  - Apply MTF on BWT output and encode the indices in the list
    - You expect to see a lot of 1’s and 2’s.
  - Use run-length encoding to store these indices
    - Write down the length of each run!
    - $\langle 1 \ 1 \ 1 \ 1 \ 2 \ 2 \ 2 \ 2 \ 1 \ 1 \ 4 \ 4 \ 4 \ 4 \ 4 \rangle \rightarrow \langle (1 \ 5) \ (2 \ 4) \ (1 \ 2) \ (4 \ 3) \rangle$
In the next class, we see how advice can potentially improve this competitive scheme!