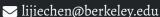
Computational Complexity Theory Fall 2025

T-time in \sqrt{T} -space September 18, 2025

Lijie Chen

University of California, Berkeley



Today's plan

- Oblivious Turing machines.
- The tree-evaluation problem.
- The main result: T-time in \sqrt{T} -space.

Oblivious Turing machines

Definition (Oblivious Turing Machine)

A multi-tape Turing machine M is **oblivious** if the movement of its heads depends only on the input length n and the time step t, but not on the input contents.

More precisely:

• For each tape i and time step $t \le T(n)$, there is a fixed position $p_i(t, n)$ where head i must be located.

Oblivious Turing machines

Definition (Oblivious Turing Machine)

A multi-tape Turing machine M is **oblivious** if the movement of its heads depends only on the input length n and the time step t, but not on the input contents.

More precisely:

- For each tape i and time step $t \le T(n)$, there is a fixed position $p_i(t, n)$ where head i must be located.
- The actual computation (state transitions, symbols written) can still depend on the input contents.

Oblivious Turing machines

Definition (Oblivious Turing Machine)

A multi-tape Turing machine M is **oblivious** if the movement of its heads depends only on the input length n and the time step t, but not on the input contents.

More precisely:

- For each tape i and time step $t \le T(n)$, there is a fixed position $p_i(t, n)$ where head i must be located.
- The actual computation (state transitions, symbols written) can still depend on the input contents.
- Every *T*-time multi-tape Turing machine can be made oblivious in time *O*(*T* log *T*).

How much space is needed to compute a recursive function?

```
DFS(u):
    if IS_LEAF(u):
        return LEAF_VALUE(u)
list = []
for v in CHILDREN(u):
        list.append(DFS(v))
result = COMBINE(u, list)
return result

# IS_LEAF(u): returns True if u is a leaf
# LEAF_VALUE(u): outputs at most m bits
# COMBINE(u, list): outputs at most m bits
# CHILDREN(u): at most 0(1) children
# Recursive depth is at most d
```

How much space is needed to compute a recursive function?

```
DFS(u):
    if IS_LEAF(u):
        return LEAF_VALUE(u)
list = []
for v in CHILDREN(u):
        list.append(DFS(v))
result = COMBINE(u, list)
return result

# IS_LEAF(u): returns True if u is a leaf
# LEAF_VALUE(u): outputs at most m bits
# COMBINE(u, list): outputs at most m bits
# CHILDREN(u): at most 0(1) children
# Recursive depth is at most d
```

• **Question**: how much space is needed to compute DFS(root)?

How much space is needed to compute a recursive function?

```
DFS(u):
    if IS_LEAF(u):
        return LEAF_VALUE(u)
list = []
for v in CHILDREN(u):
        list.append(DFS(v))
result = COMBINE(u, list)
return result

# IS_LEAF(u): returns True if u is a leaf
# LEAF_VALUE(u): outputs at most m bits
# COMBINE(u, list): outputs at most m bits
# CHILDREN(u): at most O(1) children
# Recursive depth is at most d
```

- **Question**: how much space is needed to compute DFS(root)?
- Naïve answer: $O(m \cdot d)$ bits

How much space is needed to compute a recursive function?

```
DFS(u):
    if IS_LEAF(u):
        return LEAF_VALUE(u)
list = []
for v in CHILDREN(u):
        list.append(DFS(v))
result = COMBINE(u, list)
return result

# IS_LEAF(u): returns True if u is a leaf
# LEAF_VALUE(u): outputs at most m bits
# COMBINE(u, list): outputs at most m bits
# CHILDREN(u): at most 0(1) children
# Recursive depth is at most d
```

- **Question**: how much space is needed to compute DFS(root)?
- Naïve answer: $O(m \cdot d)$ bits
- Cook-Mertz algorithm: O(m + d) bits!!!

Simulating *T*-time in \sqrt{T} -space

Theorem

Every T-time oblivious Turing machine can be simulated in $O(\sqrt{T})$ space.

Proof: see the whiteboard!