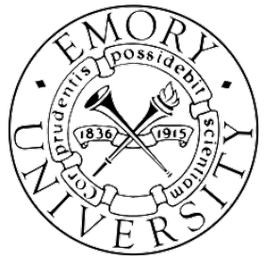
Dynamic Programming: Fibonacci

Data Structures and Algorithms

Emory University

Jinho D. Choi

EMORY
UNIVERSITY



Fibonacci Sequence

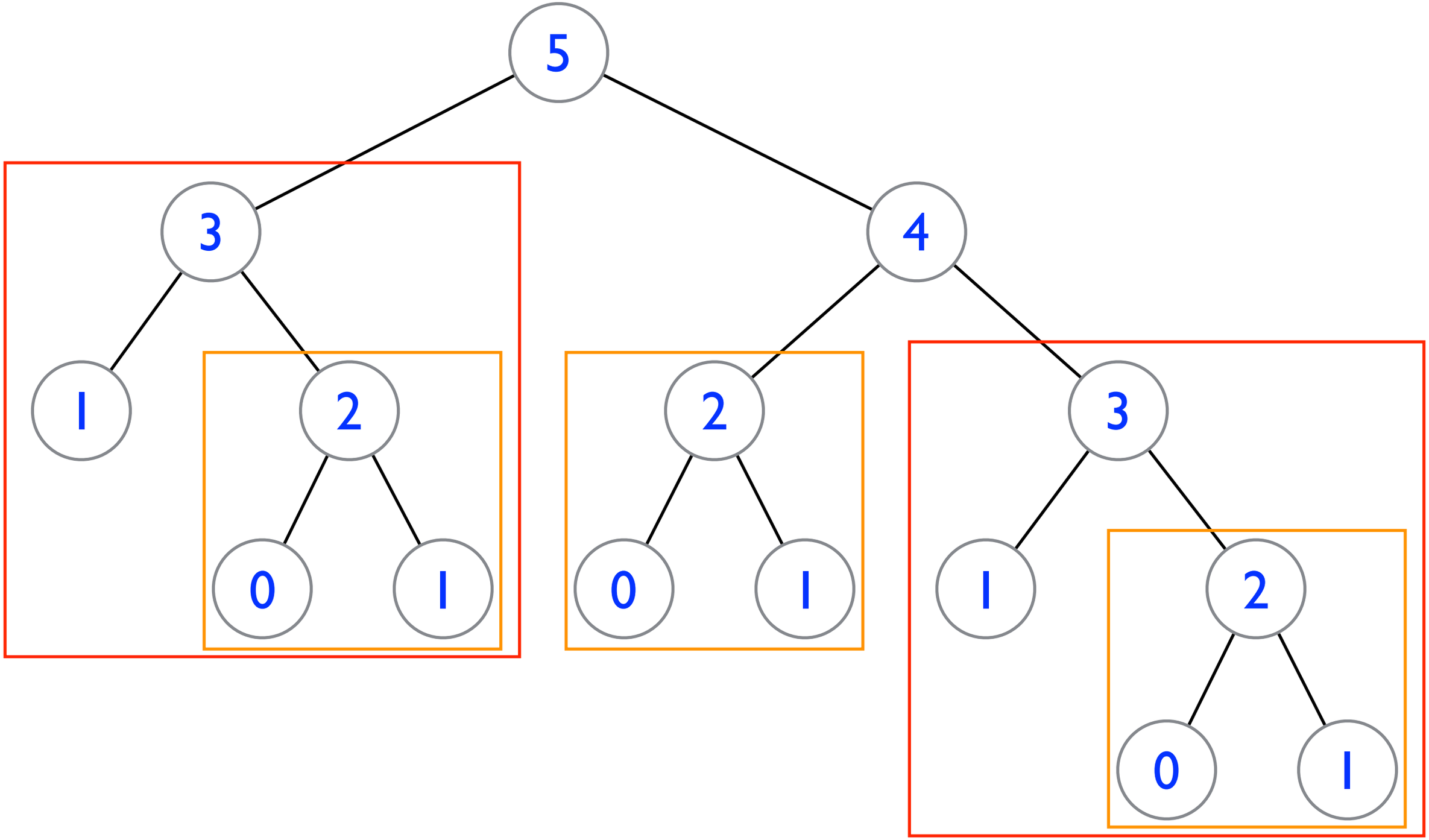
- Write a method that returns the k'th Fibonacci number.
  - $F_0 = 0, F_1 = 1, F_n = F_{n\text{-}1} + F_{n\text{-}2}$.

```
public abstract class AbstractFibonacci
{
  public int get(int k)
  {
    if (k < 0) throw new IllegalArgumentException("Invalid: "+k);
    switch (k)
    {
    case 0 : return 0;
    case 1 : return 1;
    default: return get2p(k);
    }
  }

  protected abstract int get2p(int k);
}
```

Fibonacci - Recursive

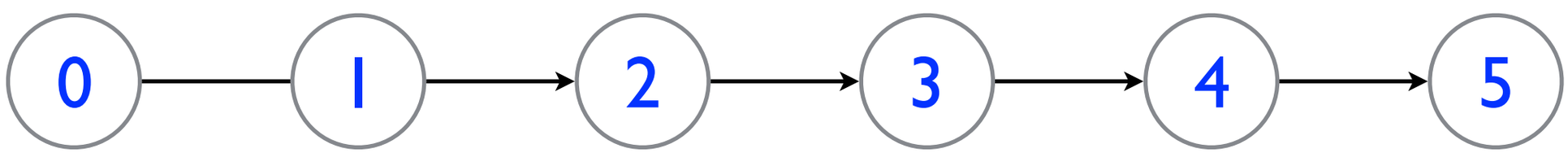
```
protected int get2p(int k)
{
  switch (k)
  {
    case 0 : return 0;
    case 1 : return 1;
    default: return get2p(k-1) + get2p(k-2);
  }
}
```

Fibonacci - Recursive

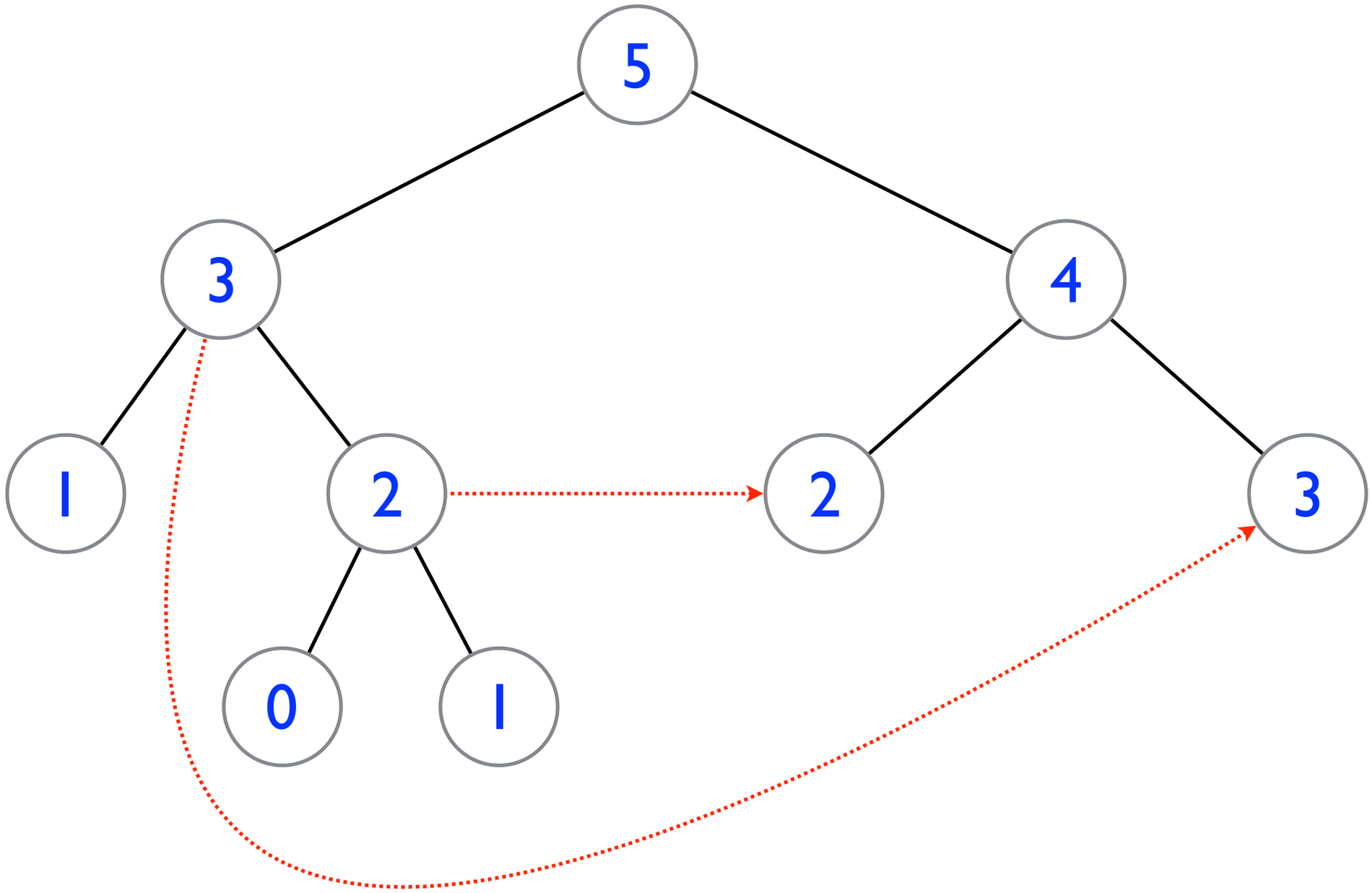


Fibonacci - Loop

```
protected int get2p(int k)
{
  int f0 = 0, f1 = 1, f2;

  for (int i=2; i<k; i++)
  {
    f2 = f0 + f1;
    f0 = f1;
    f1 = f2;
  }

  return f0 + f1;
}
```



Fibonacci - Dynamic Programming



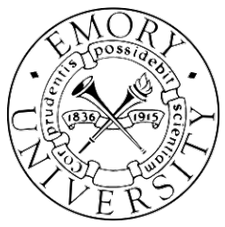
Fibonacci - Dynamic Programming

```
public int get2p(int k)
{
  return get2p(k, createTable(k));
}
```

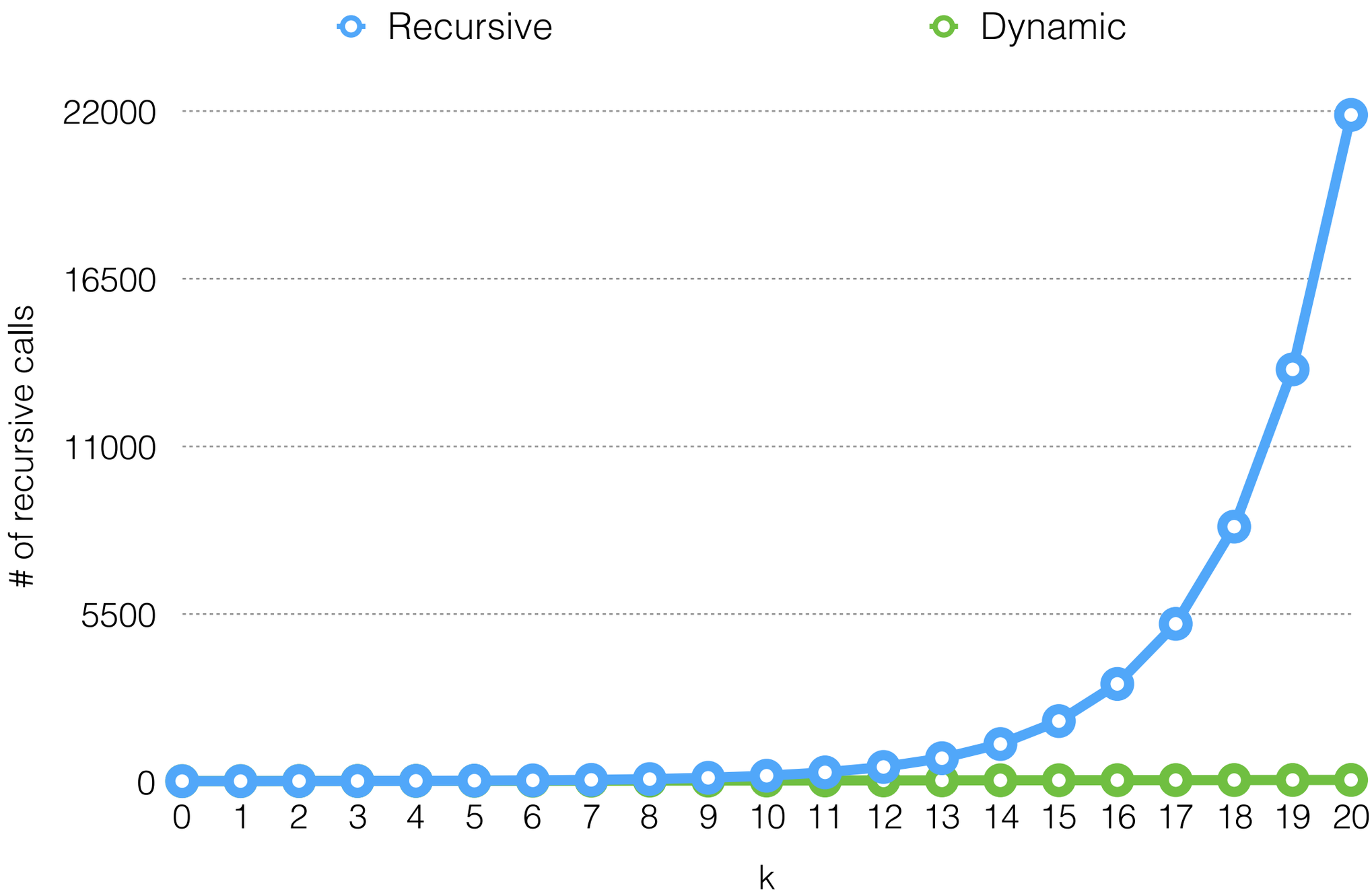
```
private int[] createTable(int k)
{
  int[] table = new int[k+1];
  table[0] = 0;
  table[1] = 1;
  Arrays.fill(table, 2, k+1, -1);
  return table;
}
```

```
private int get2p(int k, int[] table)
{
  if (table[k] < 0)
    table[k] = get2p(k-1, table) + get2p(k-2, table);

  return table[k];
}
```

EMORY
UNIVERSITY



Fibonacci - Recursive vs. Dynamic



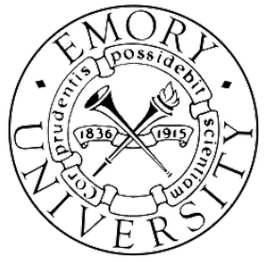
Dynamic Programming: Towers of Hanoi

Data Structures and Algorithms

Emory University

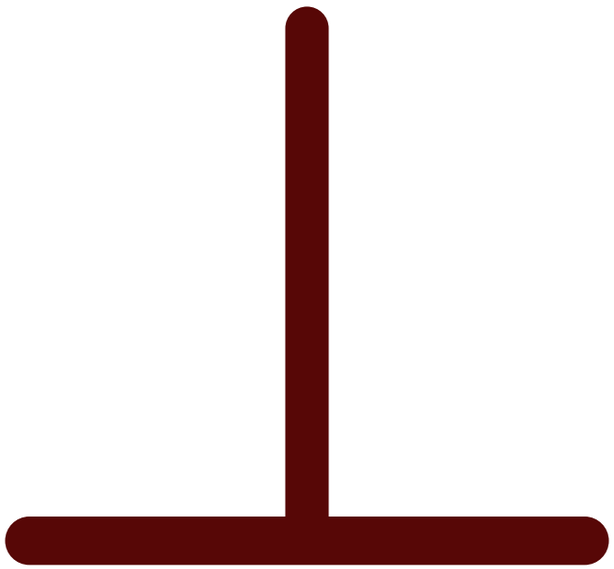
Jinho D. Choi

EMORY
UNIVERSITY

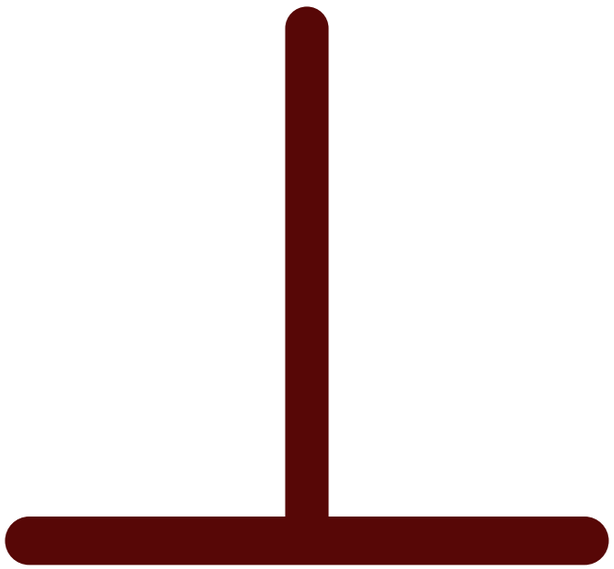


Towers of Hanoi

S



I



D

EMORY UNIVERSITY

Towers of Hanoi

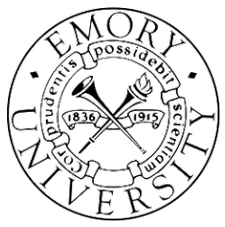
```
public abstract class AbstractHanoi
{
    public abstract List<String> solve(int n,
                    char source, char intermediate, char destination);

    protected String getKey(int n, char source, char destination)
    {
        return n+":"+source+"->"+destination;
    }
}
```

Hanoi - Recursive

```
public List<String> solve(int n,
                          char source, char intermediate, char destination)
{
  List<String> list = new ArrayList<>();
  solve(list, n, source, intermediate, destination);
  return list;
}
```

```
private void solve(List<String> list, int n,
                   char source, char intermediate, char destination)
{
  if (n == 0) return;
  solve(list, n-1, source, destination, intermediate);
  list.add(getKey(n, source, destination));
  solve(list, n-1, intermediate, source, destination);
}
```

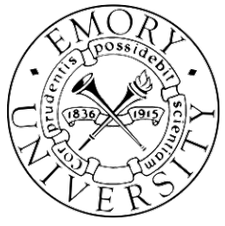
EMORY
UNIVERSITY



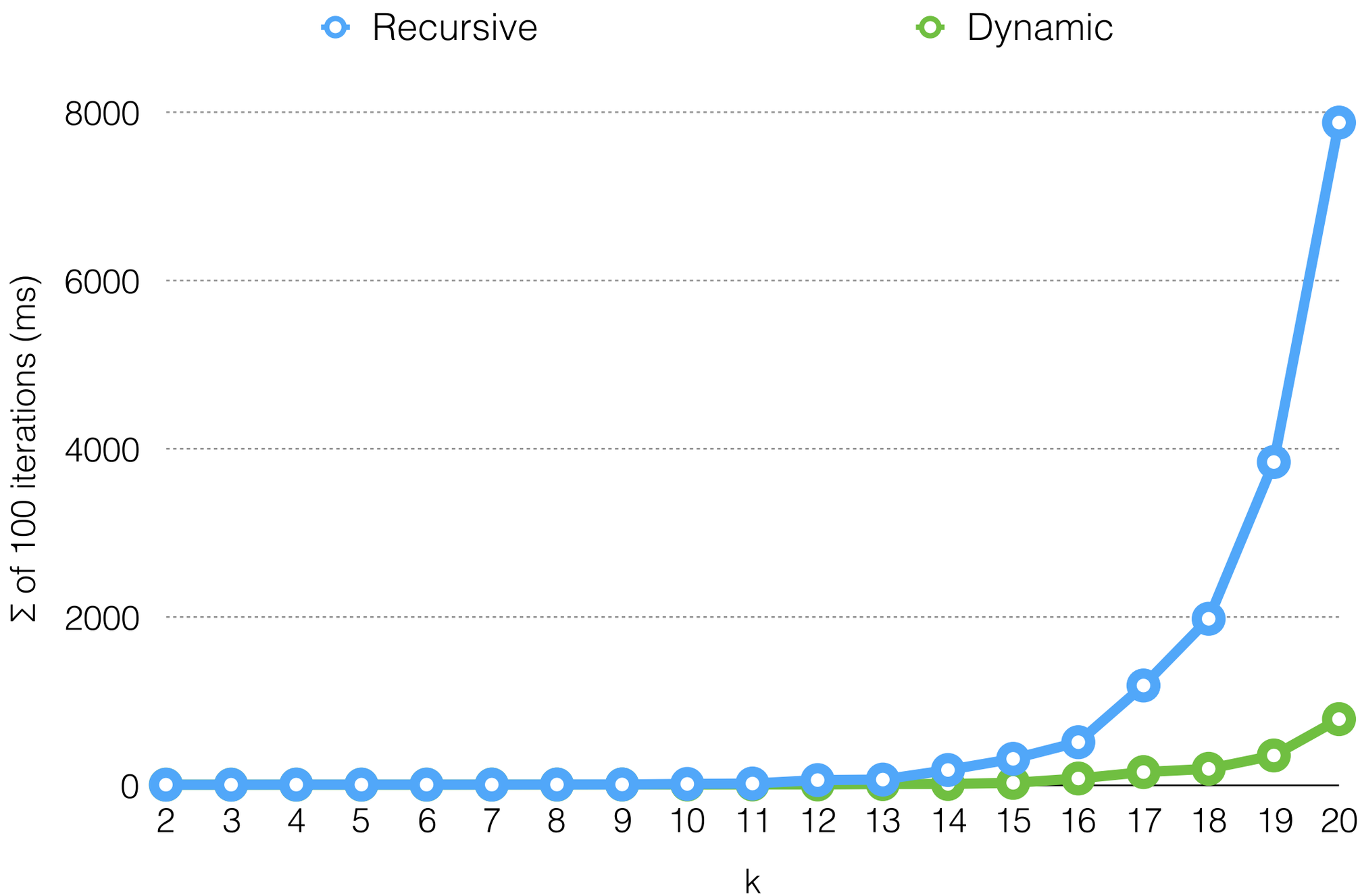
Hanoi - Dynamic Programming

```
private void solve(List<String> list, int n,
        char source, char intermediate, char destination, Map<String,int[]> map)
{
  if (n == 0) return;
  int fromIndex = list.size();

  int[] sub = map.get(getKey(n-1, source, intermediate));
  if (sub != null)   addAll(list, sub[0], sub[1]);
  else    solve(list, n-1, source, destination, intermediate, map);

  String key = getKey(n, source, destination);
  list.add(key);

  sub = map.get(getKey(n-1, intermediate, destination));
  if (sub != null)   addAll(list, sub[0], sub[1]);
  else    solve(list, n-1, intermediate, source, destination, map);

  if (!map.containsKey(key))
   map.put(key, new int[]{fromIndex, list.size()});
}
```

EMORY
UNIVERSITY



Hanoi - Recursive vs. Dynamic



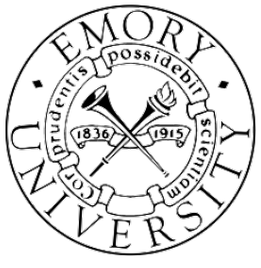
Dynamic Programming: Longest Common Subsequence

Data Structures and Algorithms

Emory University

Jinho D. Choi

EMORY
UNIVERSITY



Longest Common Subsequence

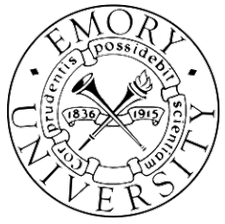
- "ABCDE":
  - Substring: {"A", "BC", "CDE", …}.
  - Subsequence: {all substrings, "AC", "ACE", …}.
  - Not subsequence: {"BA", "DAB", …}
- Longest common subsequence
  - The longest subsequence commonly shared by multiple strings.
  - e.g., "baal" is a LCS of "bilabial" and "balaclava".
  - Can there be more than one LCS?

    blal → bilabial balaclava
    blaa → bilabial balaclava
- Application
  - Find LCSs in DNA (e.g., GAATGTCCTTTCTCTAAGTCCTAAG).

Abstract LCS

```
public abstract class AbstractLCS
{
  /** @return a longest common sequence of the specific strings a and b. */
  public String solve(String a, String b)
  {
    return solve(a.toCharArray(), b.toCharArray(), a.length()-1, b.length()-1);
  }

  protected abstract String solve(char[] c, char[] d, int i, int j);
}
```

EMORY
UNIVERSITY



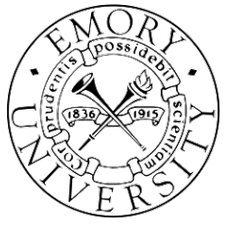
LCS - Recursive

```
(8, 8)
(7, 8)  G
(6, 7)
  ...
(0, 7)  A
(-1, 6) A
(1, 6)
  ...
(-1, 6)
 (0, 5)
(-1, 5)
  ...
 (0, 4)
(-1, 4)
```

0	1	2	3	4	5	6	7	8
A	C	G	T	C	G	T	G	T
C	T	A	G	T	G	G	A	G

```
protected String solve(char[] c, char[] d, int i, int j)
{
  if (i < 0 || j < 0)
    return "";

  if (c[i] == d[j])
    return solve(c, d, i-1, j-1) + c[i];

  String c1 = solve(c, d, i-1, j);
  String d1 = solve(c, d, i, j-1);
  return (c1.length() > d1.length()) ? c1 : d1;
}
```

EMORY
UNIVERSITY



LCS - Dynamic Programming

	0	1	2	3	4	5	6	7	8
0									
1									
2									
3									
4									
5									
6									
7									
8									

0	1	2	3	4	5	6	7	8
A	C	G	T	C	G	T	G	T
C	T	A	G	T	G	G	A	G

```
final int N = c.length, M = d.length;
int[][] table = new int[N][M];

for (int i=1; i<N; i++)
  for (int j=1; j<M; j++)
  {
    table[i][j] = (c[i] == d[j])
          ? table[i-1][j-1] + 1
          : Math.max(table[i-1][j],
                     table[i][j-1]);
 }
```

LCS - Dynamic Programming

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
2	0	0	0	1	1	1	1	1	1
3	0	1	1	1	2	2	2	2	2
4	0	1	1	1	2	2	2	2	2
5	0	1	1	2	2	3	3	3	3
6	0	1	1	2	3	3	3	3	3
7	0	1	1	2	3	4	4	4	4
8	0	1	1	2	3	4	4	4	4

C T G T G

0	1	2	3	4	5	6	7	8
A	C	G	T	C	G	T	G	T
C	T	A	G	T	G	G	A	G

```
if (i < 0 || j < 0)
  return "";

if (c[i] == d[j])
  return get(c, d, i-1, j-1, table) + c[i];

if (i == 0)
  return get(c, d, i, j-1, table);

if (j == 0)
  return get(c, d, i-1, j, table);

return (table[i-1][j] > table[i][j-1])
       ? get(c, d, i-1, j, table)
       : get(c, d, i, j-1, table);
```

Complexity?

EMORY
UNIVERSITY