Dynamic Programming

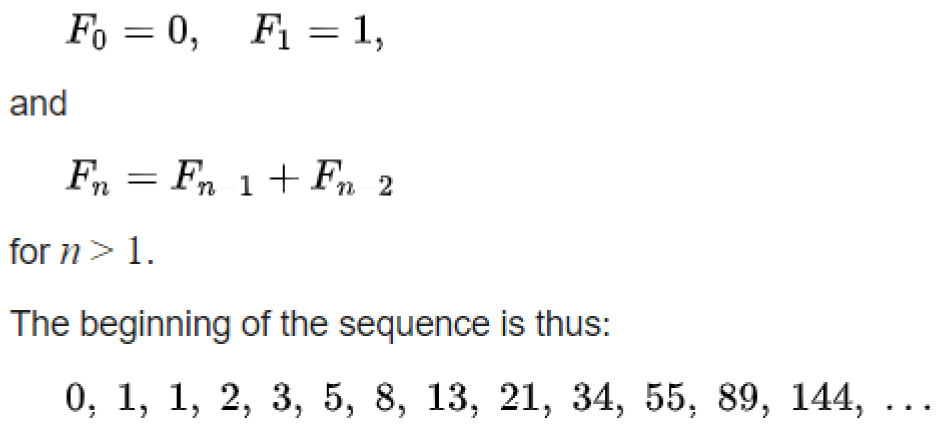
- Fibonacci numbers
- Sequence alignment
- Shortest path

Dynamic Programming

- This paradigm is most often applied in the construction of algorithms to solve a certain class of **Optimisation Problem**. That is: problems which require the minimisation or maximisation of some measure. One disadvantage of using **Divide-and-Conquer** is that the process of recursively solving separate sub-instances can result in the same computations being performed repeatedly since identical sub-instances may arise. The idea behind dynamic programming is to avoid this pathology by obviating the requirement to calculate the same quantity twice. The method usually accomplishes this by maintaining a table of sub-instance results.

http://www.csc.liv.ac.uk/~ped/teachadmin/algor/dyprog.html

Fibonacci numbers



$$F_0 = 0, \quad F_1 = 1,$$

and

$$F_n = F_{n-1} + F_{n-2}$$

for $n > 1$.

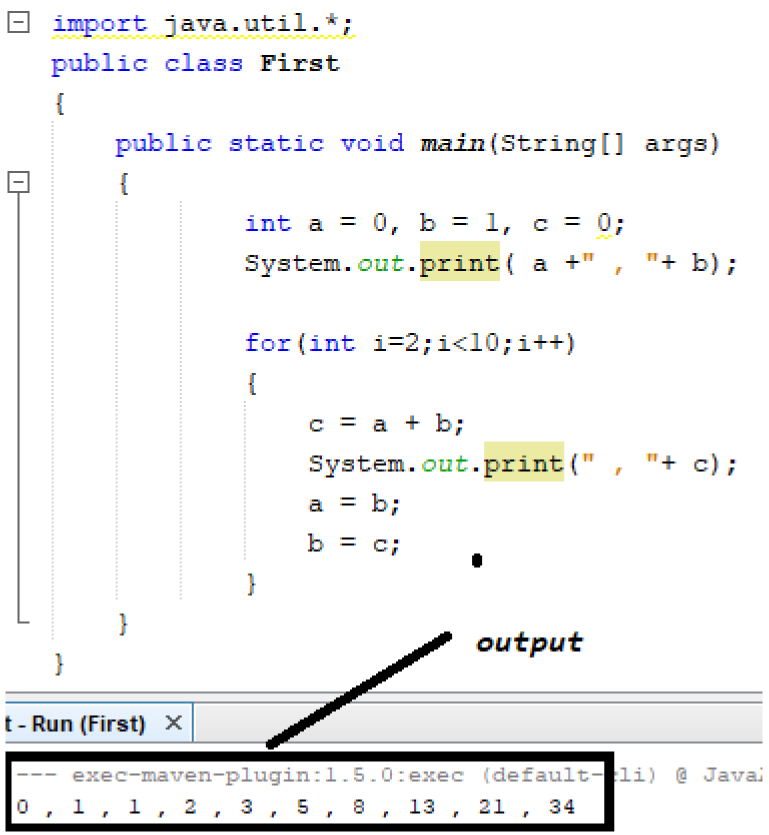
The beginning of the sequence is thus:

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ 89,\ 144,\ \ldots$$

Query: F_{200}?

Query: F_{200}?

- Writing a ‘for’ loop?



```java
import java.util.*;
public class First
{
    public static void main(String[] args)
    {
            int a = 0, b = 1, c = 0;
            System.out.print( a +" , "+ b);

            for(int i=2;i<10;i++)
            {
                c = a + b;
                System.out.print(" , "+ c);
                a = b;
                b = c;
            }
    }
}
```

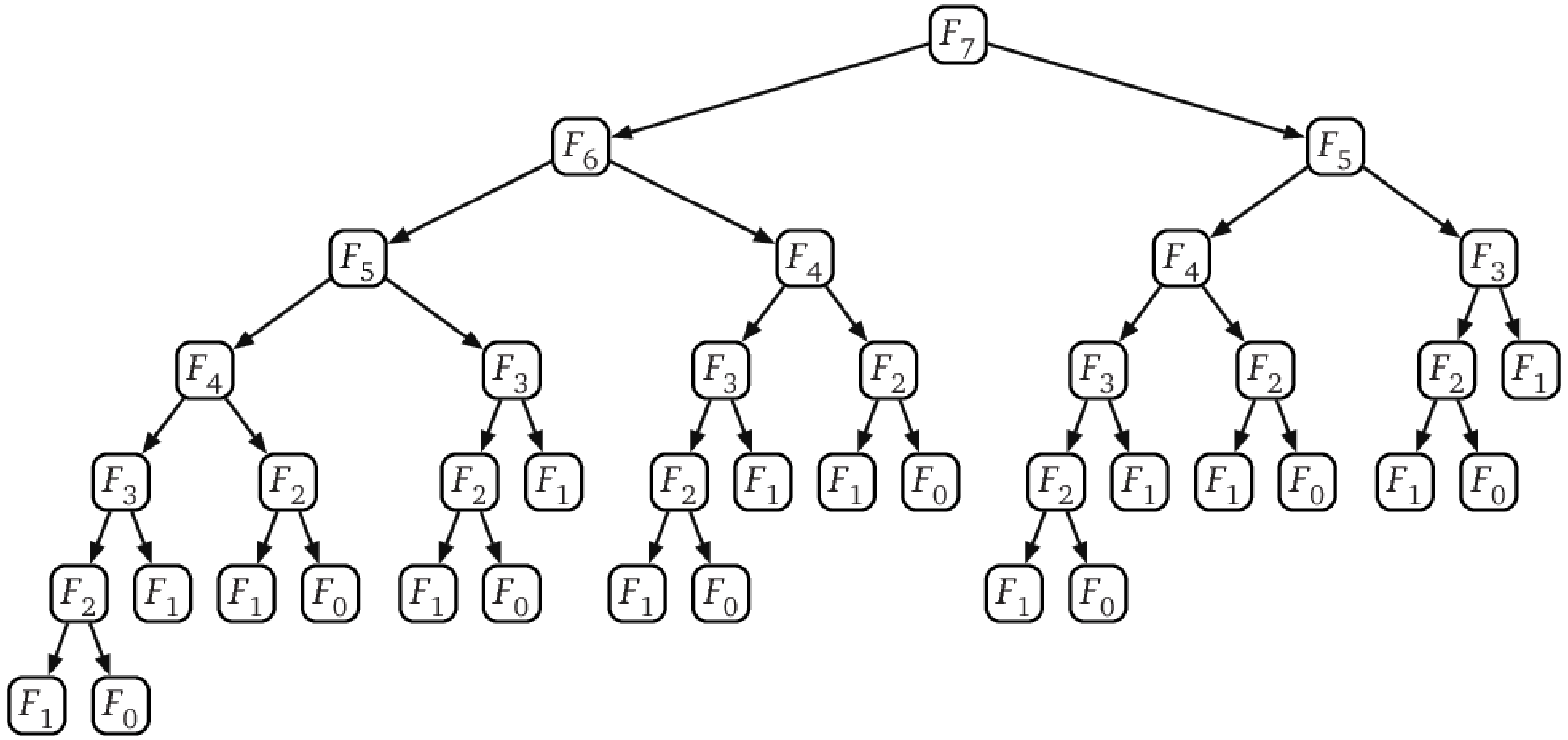
```
--- exec-maven-plugin:1.5.0:exec (default-cli) @ Java
0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34
```

- Writing a recursive function?

Algorithm 1: F(n)

```
Input: Some non-negative integer $n$
Output: The $nth$ number in the Fibonacci Sequence
if $n \leq 1$ then
    return $n$
else
    return $F(n-1) + F(n-2)$;
```

Using Recursion



Sequence Alignment

Chien-Yu Chen@BIME.NTU

Sequences

- DNA/RNA sequences
  - strings composed of an alphabet of 4 letters
- Protein sequences
  - alphabet of 20 letters

Alignment

- Any assignment of correspondences that preserves the order of the residues within the sequences is an alignment.
- Introducing gaps

```
A reasonable alignment would be    A A C D E –
                                   A – C D E G
```

Many Possibilities

- An uninformative alignment:

```
--------gctgsscg
ctataatc--------
```

- An alignment without gaps:

```
gctgaacg
ctataatc
```

- An alignment with gaps:

```
gctga-a--cg
--ct-ataatc
```

- And another:

```
gctg-aa-cg
-ctataatc-
```

Find the Best

- Need a way to examine all possible alignments systematically
- Compute a score reflecting the quality of each possible alignment
- To identify the alignment with the optimal score
- Several different alignments may give the same best score
- Many different scoring schemes

Formal Description

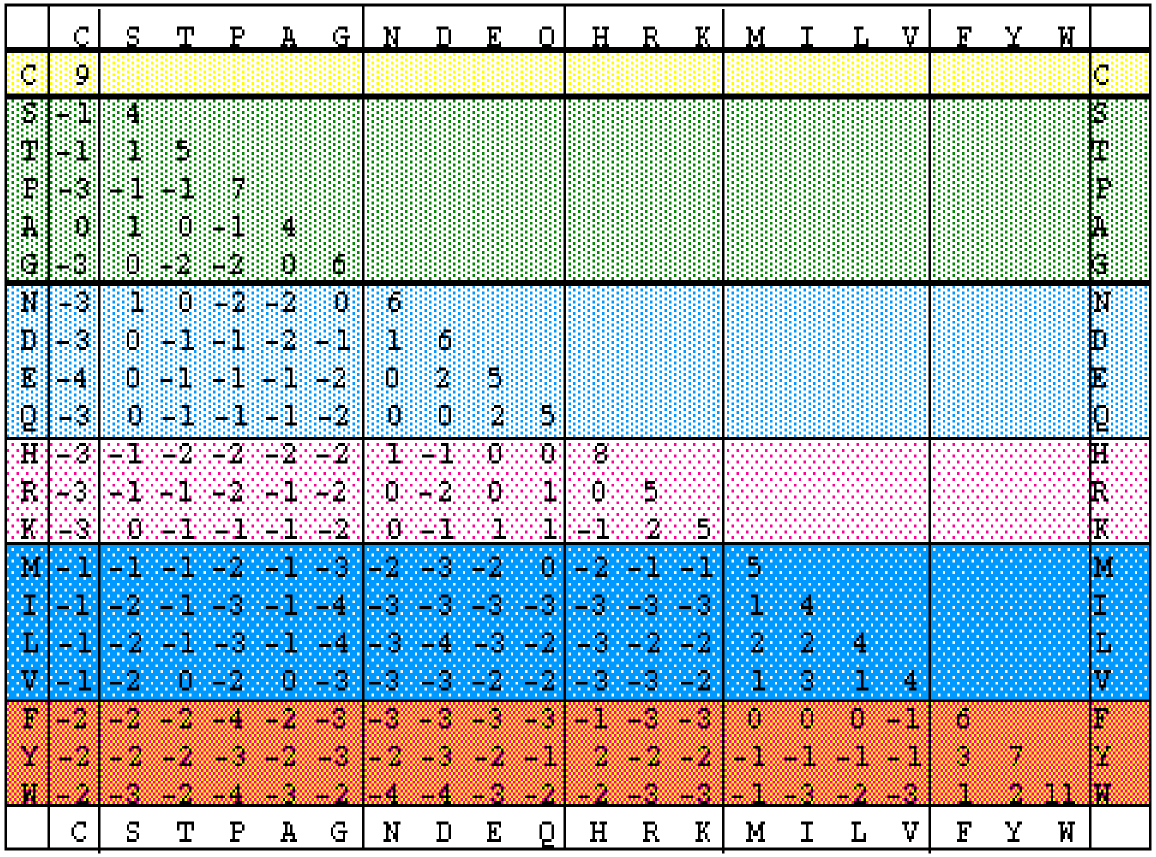
- *Problem:* **PairSeqAlign**
- *Input:* Two sequences *x,y*

Two sequences	*x,y*
Scoring matrix	*s*
Gap penalty	*gap_penalty*

- *Output:* The optimal sequence alignment

Scoring/Substitution Matrices

BLOSUM62



	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	
C	9																				C
S	-1	4																			S
T	-1	1	5																		T
P	-3	-1	-1	7																	P
A	0	1	0	-1	4																A
G	-3	0	-2	-2	0	6															G
N	-3	1	0	-2	-2	0	6														N
D	-3	0	-1	-1	-2	-1	1	6													D
E	-4	0	-1	-1	-1	-2	0	2	5												E
Q	-3	0	-1	-1	-1	-2	0	0	2	5											Q
H	-3	-1	-2	-2	-2	-2	1	-1	0	0	8										H
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	5									R
K	-3	0	-1	-1	-1	-2	0	-1	1	1	-1	2	5								K
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	5							M
I	-1	-2	-1	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	4						I
L	-1	-2	-1	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	4					L
V	-1	-2	0	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	4				V
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	6			F
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	7		Y
W	-2	-3	-2	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	11	W
	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	

Gap Penalties

- Linear gap penalty
  - cost of gap (length g) depends linearly on *gap penalty*
    - f(*length*)= – *length* × gap*_penalty*

Global Alignment

- Needleman-Wunsch 1970
- Idea: Build up optimal alignment from optimal alignments of subsequences

The Recurrence Relations

$$F(i,j) = \max\begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ F(i-1, j) + w \\ F(i, j-1) + w \end{cases}$$

Three Steps of Dynamic Programming

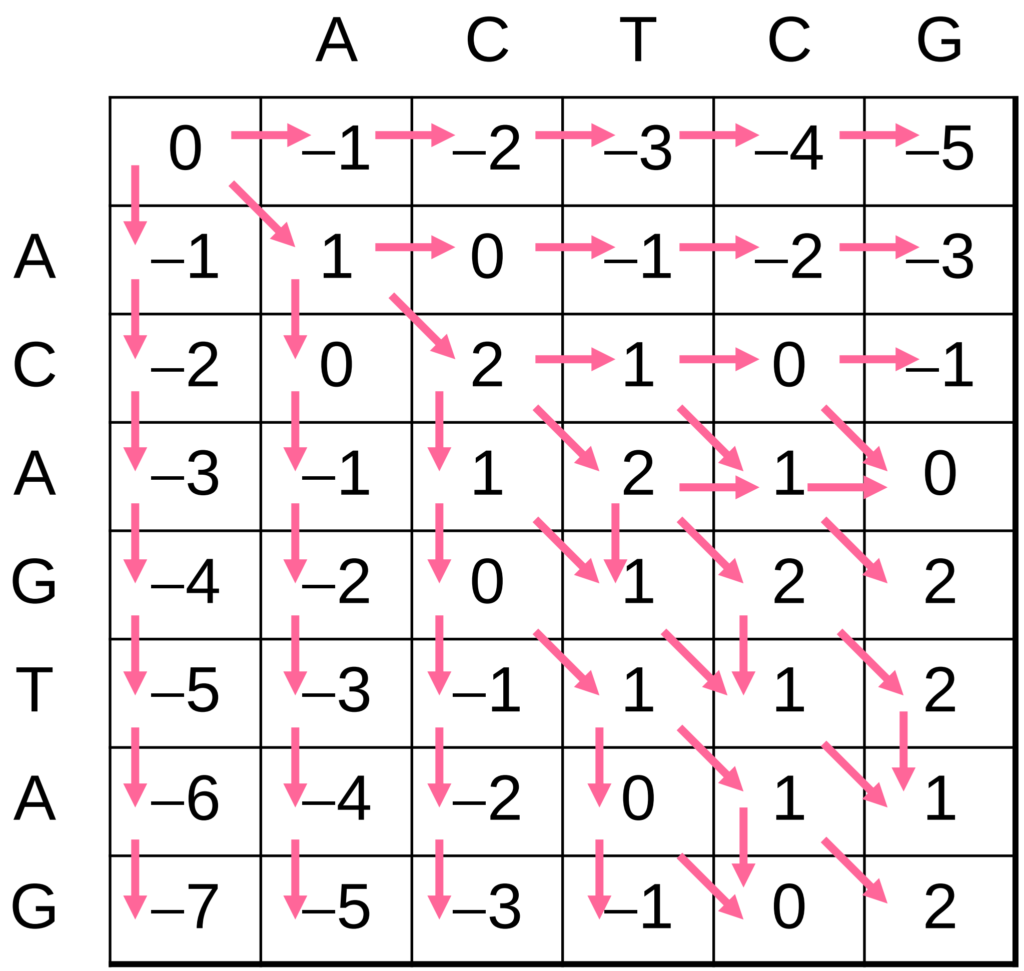
- Three steps in dynamic programming
  - Initialization
  - Matrix fill (scoring)
  - Traceback (alignment)

Initialization

gap penalty = –1
match score = 1
mismatch score = 0

		A	C	T	C	G
	0	→ –1	→ –2	→ –3	→ –4	→ –5
A	↓ –1					
C	↓ –2					
A	↓ –3					
G	↓ –4					
T	↓ –5					
A	↓ –6					
G	↓ –7					

Matrix fill (scoring)



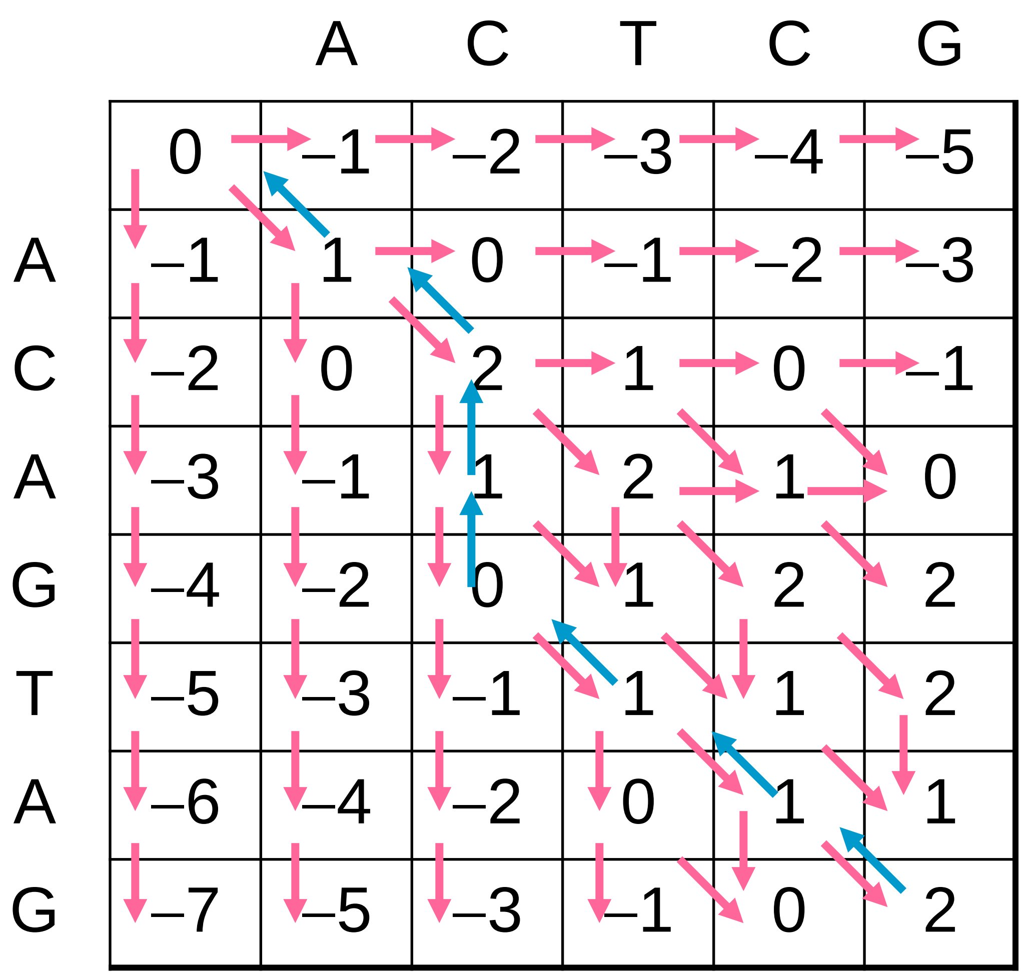
		A	C	T	C	G
	0	–1	–2	–3	–4	–5
A	–1	1	0	–1	–2	–3
C	–2	0	2	1	0	–1
A	–3	–1	1	2	1	0
G	–4	–2	0	1	2	2
T	–5	–3	–1	1	1	2
A	–6	–4	–2	0	1	1
G	–7	–5	–3	–1	0	2

Traceback (alignment)

gap penalty = –1
match score = 1
mismatch score = 0

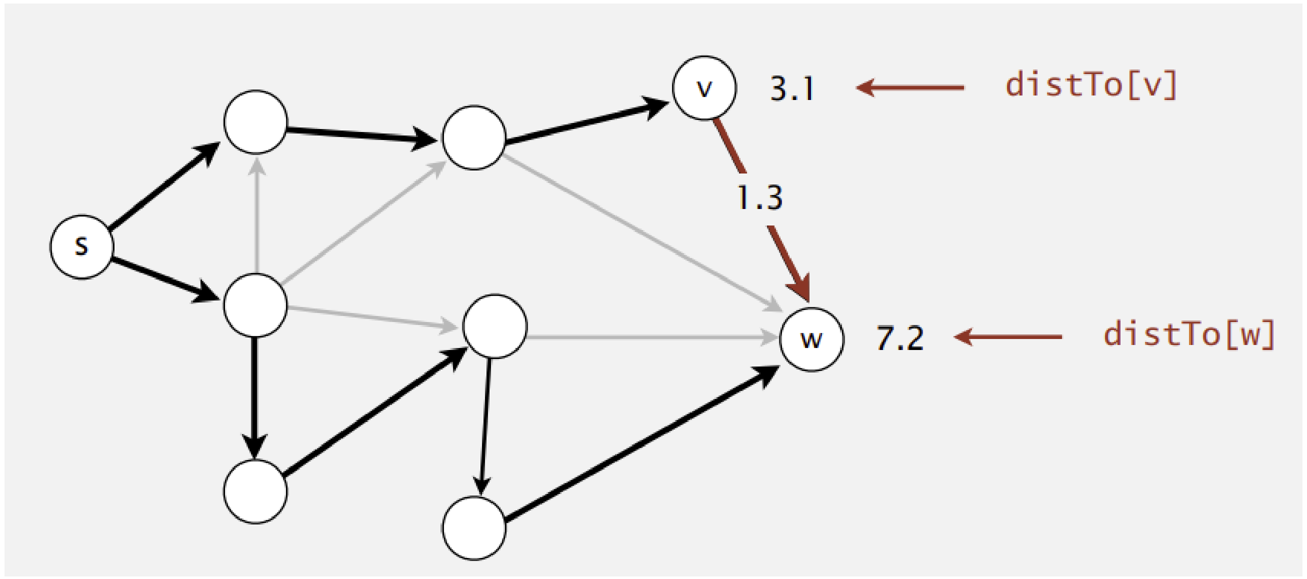
Output:

```
AC--TCG
ACAGTAG
```



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

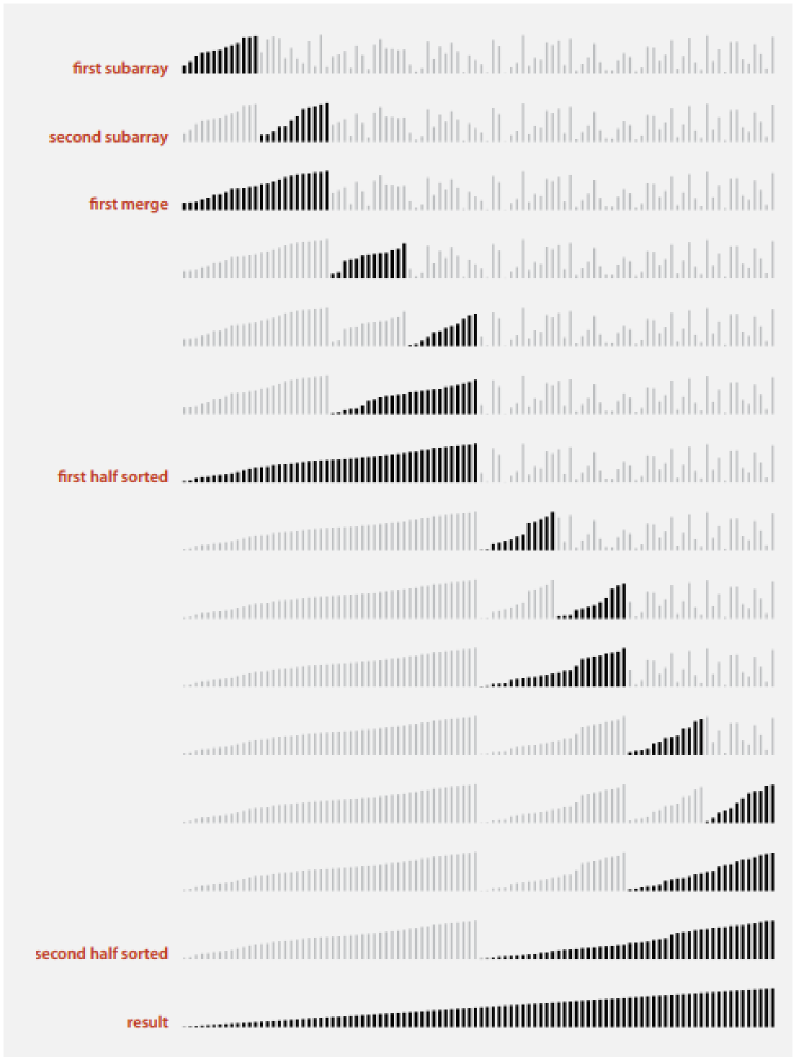
Shortest Path



Divide and Conquer

- Mergesort
  - Top-down
  - Bottom-up

Top-down vs. Bottom-up


first subarray
second subarray
first merge
first half sorted
second half sorted
result