First discussion section agenda

- Introductions
- HW1 context/advice/questions
- C++/general programming tips
- Suggestions for future topics

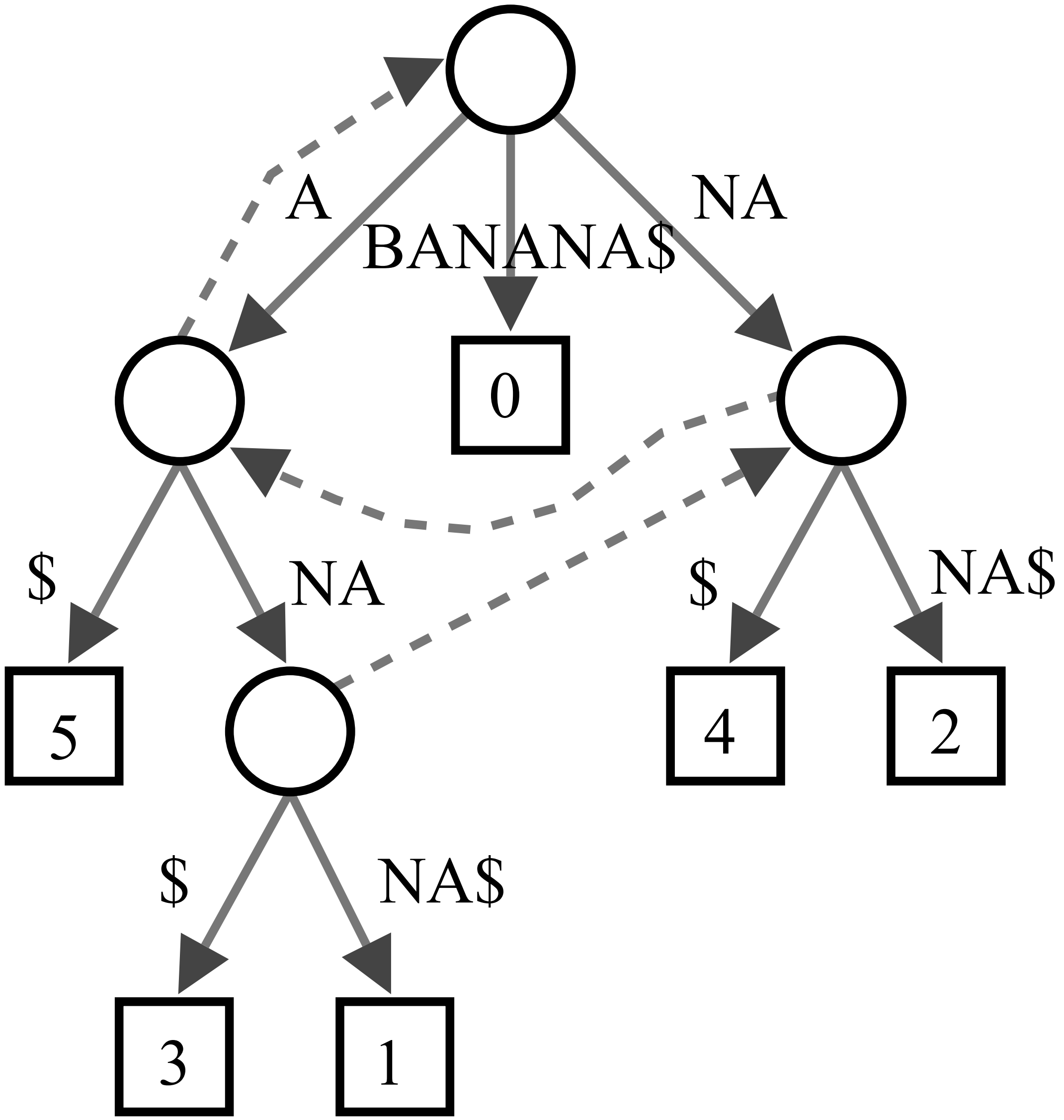
Introductions

- Who am I? (not 24601)
  - 4th year Genome Sciences graduate student, Borenstein lab
  - Microbial communities
    - Community design algorithms
    - Taxonomic structure-gene content relationships
  - Main languages: R, Python, C++, and recently some Javascript

- Who are you?
  - Department?
  - Programming experience/language of choice?

Suffix trees

- Applications:
  - Longest-repeated substring
  - String repetitions
  - Compression
  - Genetic sequence analysis
  - and more...

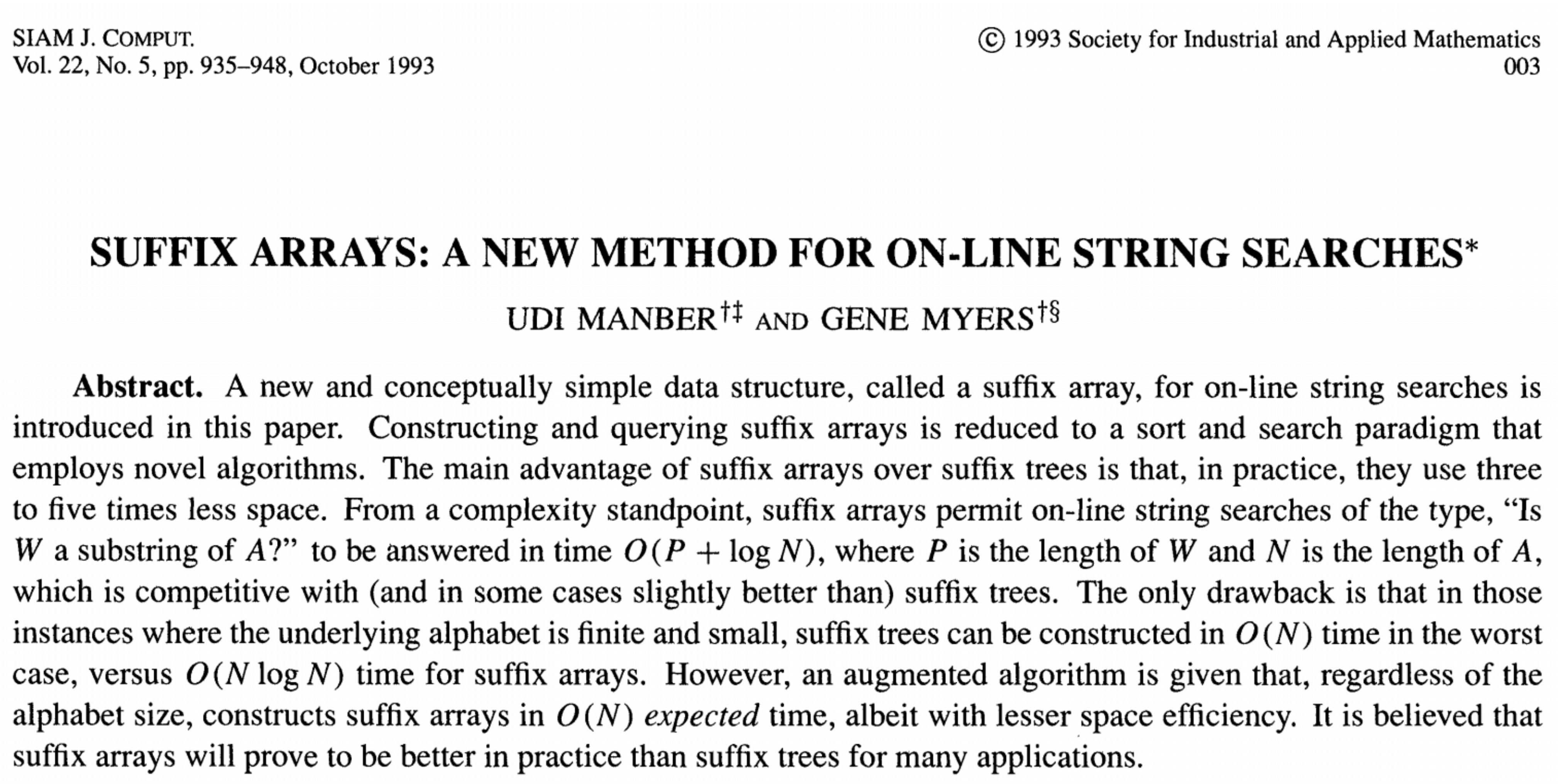


SIAM J. COMPUT.
Vol. 22, No. 5, pp. 935–948, October 1993

SUFFIX ARRAYS: A NEW METHOD FOR ON-LINE STRING SEARCHES*

UDI MANBER†‡ AND GENE MYERS†§

Abstract. A new and conceptually simple data structure, called a suffix array, for on-line string searches is introduced in this paper. Constructing and querying suffix arrays is reduced to a sort and search paradigm that employs novel algorithms. The main advantage of suffix arrays over suffix trees is that, in practice, they use three to five times less space. From a complexity standpoint, suffix arrays permit on-line string searches of the type, "Is W a substring of A?" to be answered in time $O(P + \log N)$, where P is the length of W and N is the length of A, which is competitive with (and in some cases slightly better than) suffix trees. The only drawback is that in those instances where the underlying alphabet is finite and small, suffix trees can be constructed in $O(N)$ time in the worst case, versus $O(N \log N)$ time for suffix arrays. However, an augmented algorithm is given that, regardless of the alphabet size, constructs suffix arrays in $O(N)$ *expected* time, albeit with lesser space efficiency. It is believed that suffix arrays will prove to be better in practice than suffix trees for many applications.



SIAM J. COMPUT.
Vol. 22, No. 5, pp. 935–948, October 1993

SUFFIX ARRAYS: A NEW METHOD FOR ON-LINE STRING SEARCHES*

UDI MANBER[†‡] AND GENE MYERS[†§]

Abstract. A new and conceptually simple data structure, called a suffix array, for on-line string searches is introduced in this paper. Constructing and querying suffix arrays is reduced to a sort and search paradigm that employs novel algorithms. The main advantage of suffix arrays over suffix trees is that, in practice, they use three to five times less space. From a complexity standpoint, suffix arrays permit on-line string searches of the type, "Is W a substring of A?" to be answered in time $O(P + \log N)$, where P is the length of W and N is the length of A, which is competitive with (and in some cases slightly better than) suffix trees. The only drawback is that in those instances where the underlying alphabet is finite and small, suffix trees can be constructed in $O(N)$ time in the worst case, versus $O(N \log N)$ time for suffix arrays. However, an augmented algorithm is given that, regardless of the alphabet size, constructs suffix arrays in $O(N)$ *expected* time, albeit with lesser space efficiency. It is believed that suffix arrays will prove to be better in practice than suffix trees for many applications.

Udi Manber: Yahoo!, Amazon, Google, Youtube, currently at the NIH

Gene Myers: One of the creators of BLAST, currently at the Max Planck Institute

Why suffix arrays?

Why suffix arrays?

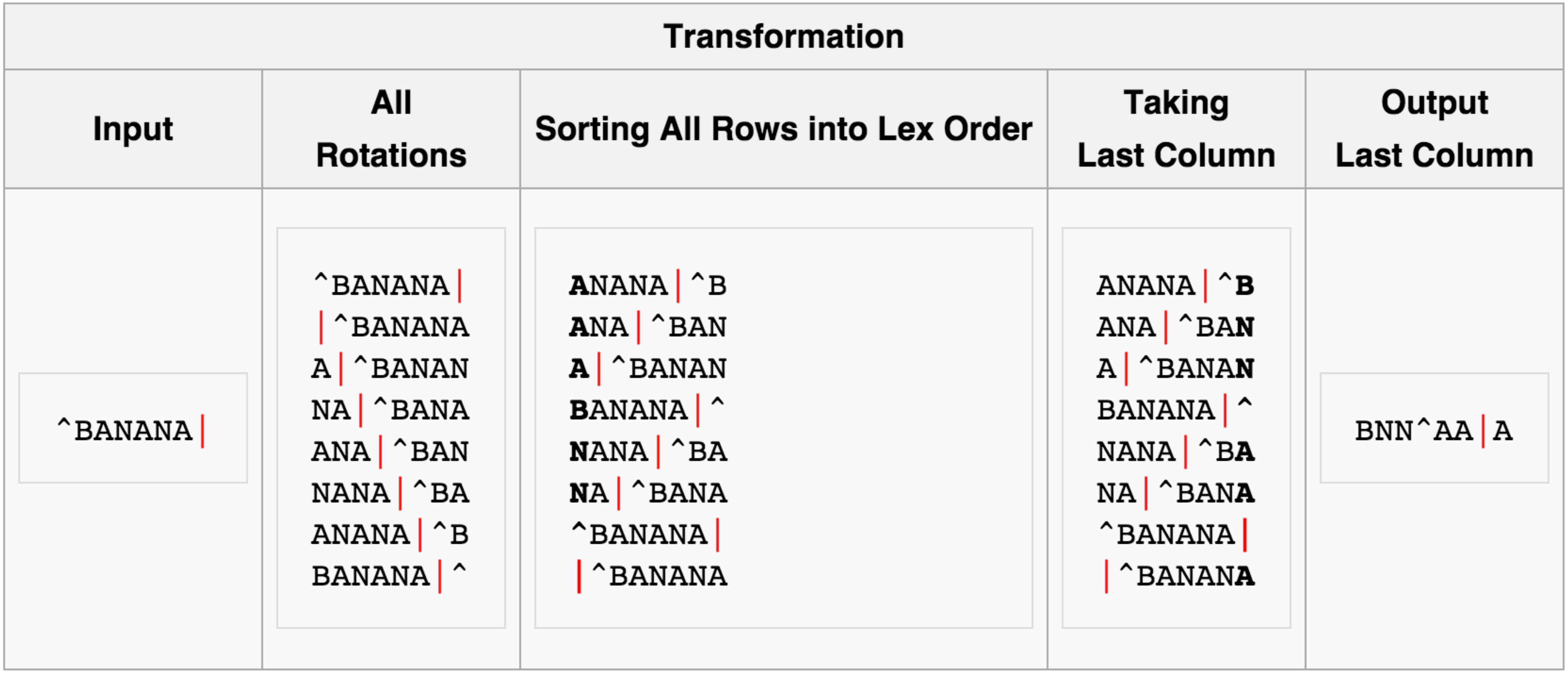
- Comparison to suffix trees
  - 5 times as space efficient as suffix trees
  - Search times are similar
  - Takes longer to construct (but technically can be done with the same asymptotic time complexity)

Why suffix arrays?

- Comparison to suffix trees
  - 5 times as space efficient as suffix trees
  - Search times are similar
  - Takes longer to construct (but technically can be done with the same asymptotic time complexity)

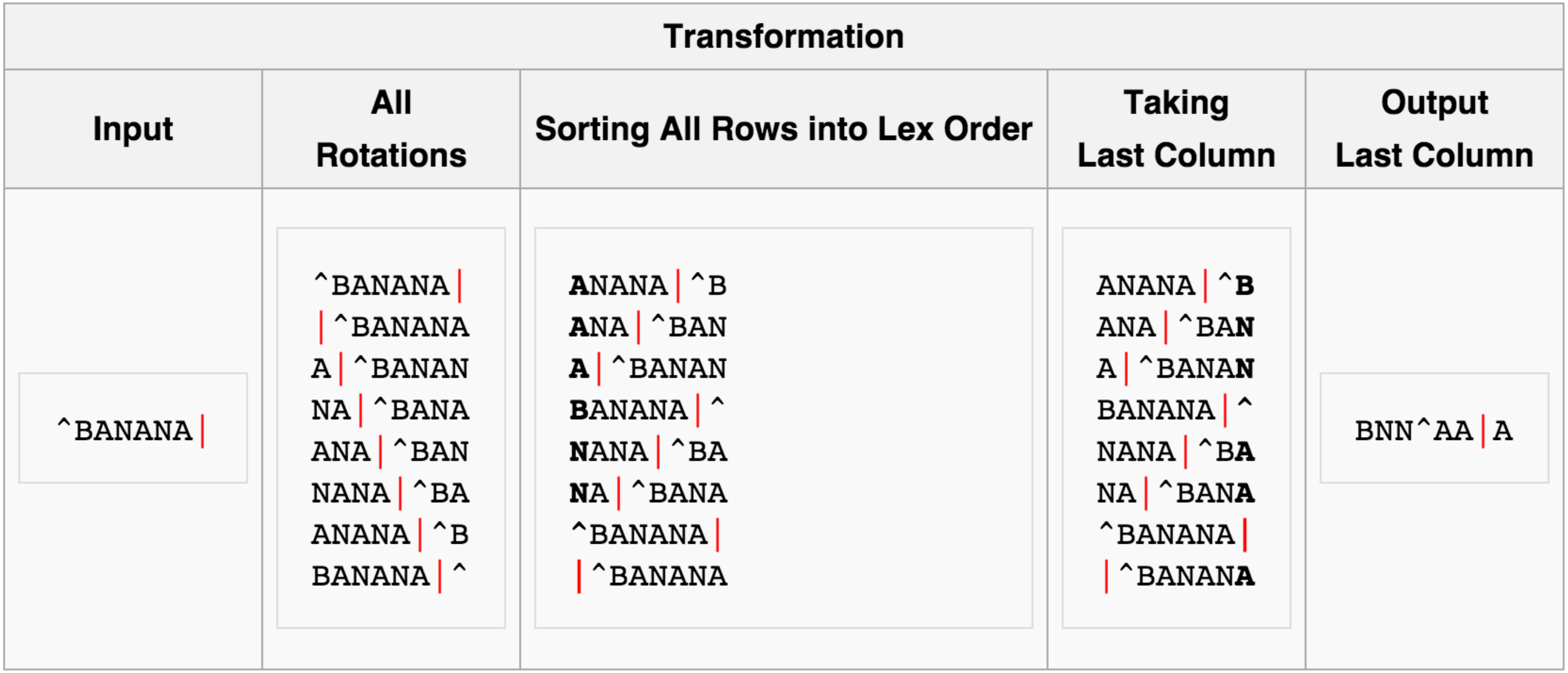
- “A primary motivation for this paper was to be able to efficiently answer on-line string queries for very long genetic sequences (on the order of one million or more symbols long). In practice, it is the space overhead of the query data structure that limits the largest text that may be handled.”

A cool example: Burrows-Wheeler transform



Transformation				
Input	All Rotations	Sorting All Rows into Lex Order	Taking Last Column	Output Last Column
^BANANA|	^BANANA| |^BANANA A|^BANAN NA|^BANA ANA|^BAN NANA|^BA ANANA|^B BANANA|^	**A**NANA|^B **A**NA|^BAN **A**|^BANAN **B**ANANA|^ **N**ANA|^BA **N**A|^BANA **^**BANANA| **|**^BANANA	ANANA|^**B** ANA|^BA**N** A|^BANA**N** BANANA|**^** NANA|^B**A** NA|^BAN**A** ^BANANA**|** |^BANAN**A**	BNN^AA|A

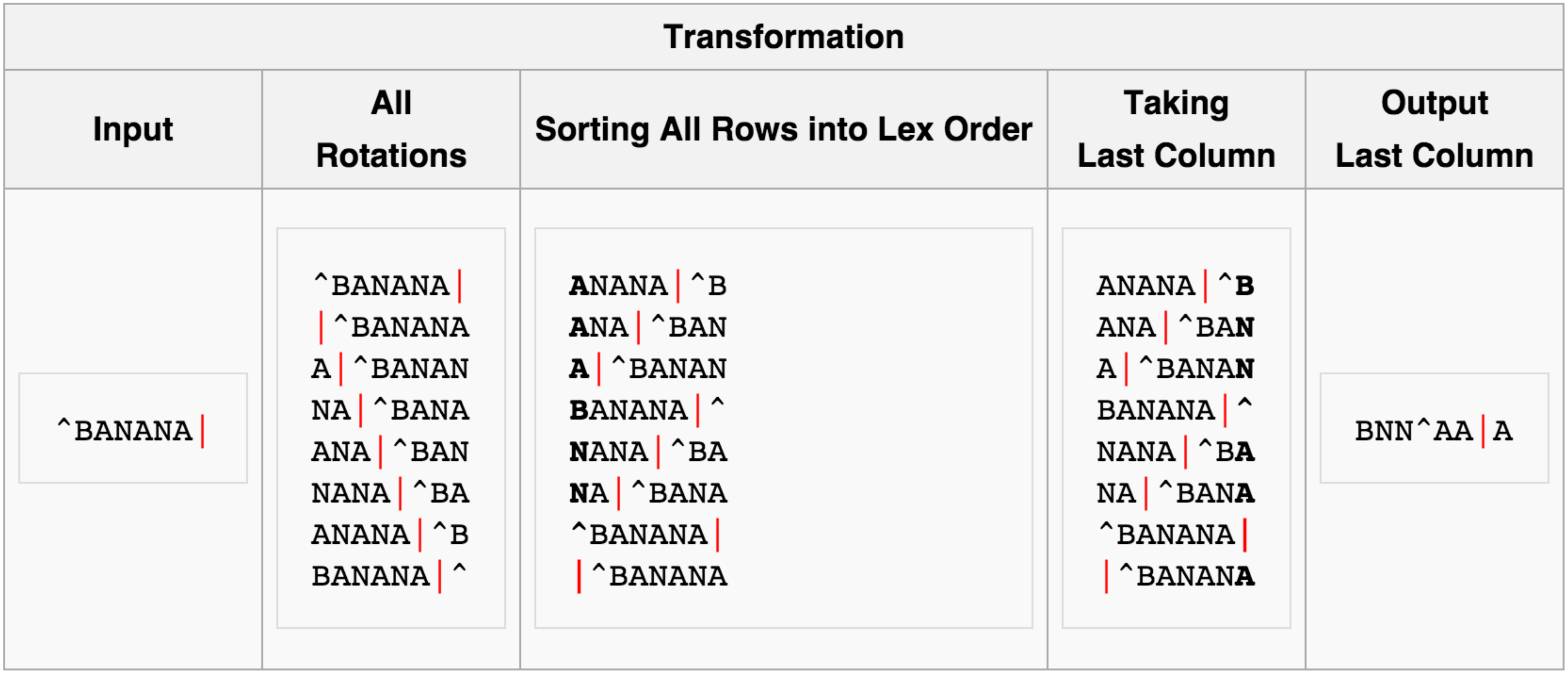
A cool example: Burrows-Wheeler transform



Transformation				
Input	**All Rotations**	**Sorting All Rows into Lex Order**	**Taking Last Column**	**Output Last Column**
^BANANA|	^BANANA|	**A**NANA|^B	ANANA|^**B**	BNN^AA|A
	|^BANANA	**A**NA|^BAN	ANA|^BA**N**	
	A|^BANAN	**A**|^BANAN	A|^BANA**N**	
	NA|^BANA	**B**ANANA|^	BANANA|**^**	
	ANA|^BAN	**N**ANA|^BA	NANA|^B**A**	
	NANA|^BA	**N**A|^BANA	NA|^BAN**A**	
	ANANA|^B	**^**BANANA|	^BANANA**|**	
	BANANA|^	**|**^BANANA	|^BANAN**A**	

- Leads to repetitive sequences, which are easier to compress, and is easily invertible

A cool example: Burrows-Wheeler transform



Transformation				
Input	All Rotations	Sorting All Rows into Lex Order	Taking Last Column	Output Last Column
^BANANA|	^BANANA| |^BANANA A|^BANAN NA|^BANA ANA|^BAN NANA|^BA ANANA|^B BANANA|^	**A**NANA|^B **A**NA|^BAN **A**|^BANAN **B**ANANA|^ **N**ANA|^BA **N**A|^BANA **^**BANANA| **|**^BANANA	ANANA|^**B** ANA|^BA**N** A|^BANA**N** BANANA|**^** NANA|^B**A** NA|^BAN**A** ^BANANA**|** |^BANAN**A**	BNN^AA|A

- Leads to repetitive sequences, which are easier to compress, and is easily invertible
- Used in Bowtie (short read aligner developed by Cole Trapnell)

Homework 1 tips

- Plan out your algorithm with pseudocode

Homework 1 tips

- Plan out your algorithm with pseudocode
- Think about what comparisons you need (and don't need) to make

Homework 1 tips

- Plan out your algorithm with pseudocode
- Think about what comparisons you need (and don't need) to make
- Get comfortable with pointers

Homework 1 tips

- Plan out your algorithm with pseudocode
- Think about what comparisons you need (and don't need) to make
- Get comfortable with pointers
- Think about how to store inputs

Homework 1 tips

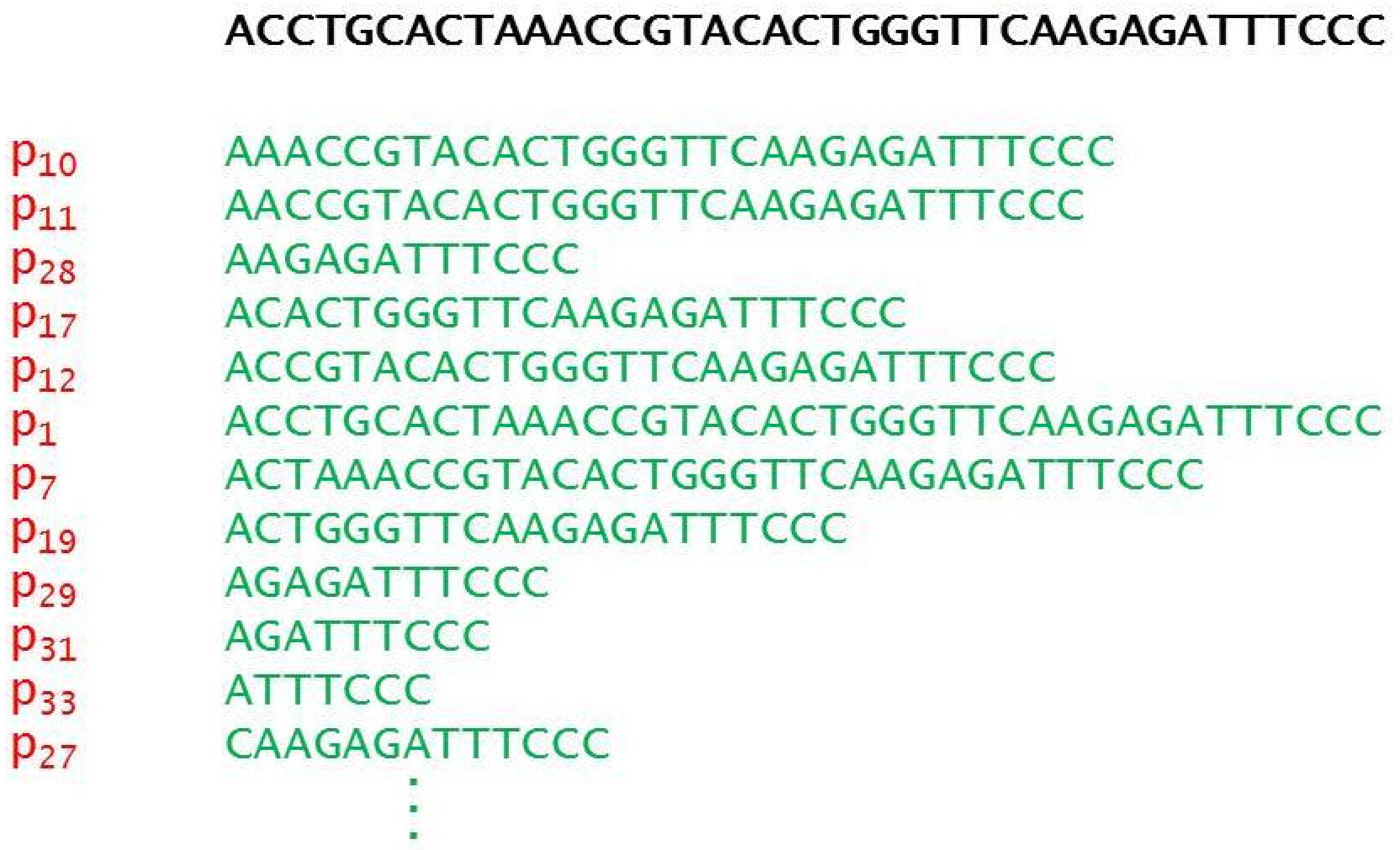
- Plan out your algorithm with pseudocode
- Think about what comparisons you need (and don't need) to make
- Get comfortable with pointers
- Think about how to store inputs
- Think about how to store results (and intermediate results)

Homework 1 tips

- Plan out your algorithm with pseudocode
- Think about what comparisons you need (and don't need) to make
- Get comfortable with pointers
- Think about how to store inputs
- Think about how to store results (and intermediate results)
- Try to format your output to match the template

Creating a suffix array

- Step 3:
  - Sort the suffix pointers lexicographically



ACCTGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC

p_{10}	AAACCGTACACTGGGTTCAAGAGATTTCCC
p_{11}	AACCGTACACTGGGTTCAAGAGATTTCCC
p_{28}	AAGAGATTTCCC
p_{17}	ACACTGGGTTCAAGAGATTTCCC
p_{12}	ACCGTACACTGGGTTCAAGAGATTTCCC
p_{1}	ACCTGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC
p_{7}	ACTAAACCGTACACTGGGTTCAAGAGATTTCCC
p_{19}	ACTGGGTTCAAGAGATTTCCC
p_{29}	AGAGATTTCCC
p_{31}	AGATTTCCC
p_{33}	ATTTCCC
p_{27}	CAAGAGATTTCCC
	⋮

Programming style tips

Programming style tips

- The more readable your code is
  - The easier it will be for me to help
  - The more useful it will be to you later (especially if you TA this class)

Programming style tips

- The more readable your code is
  - The easier it will be for me to help
  - The more useful it will be to you later (especially if you TA this class)
- Tips for readability
  - Intuitive and meaningful variable/function names

Programming style tips

- The more readable your code is
  - The easier it will be for me to help
  - The more useful it will be to you later (especially if you TA this class)
- Tips for readability
  - Intuitive and meaningful variable/function names
  - Comments
    - Describe what the different parts of your program are doing

Programming style tips

- The more readable your code is
  - The easier it will be for me to help
  - The more useful it will be to you later (especially if you TA this class)
- Tips for readability
  - Intuitive and meaningful variable/function names
  - Comments
    - Describe what the different parts of your program are doing
      - Especially if you're doing something complicated

Programming style tips

- The more readable your code is
  - The easier it will be for me to help
  - The more useful it will be to you later (especially if you TA this class)
- Tips for readability
  - Intuitive and meaningful variable/function names
  - Comments
    - Describe what the different parts of your program are doing
      - Especially if you're doing something complicated
  - Simplicity is better to begin with, optimize later if necessary

Programming style tips

- The more readable your code is
  - The easier it will be for me to help
  - The more useful it will be to you later (especially if you TA this class)
- Tips for readability
  - Intuitive and meaningful variable/function names
  - Comments
    - Describe what the different parts of your program are doing
      - Especially if you're doing something complicated
  - Simplicity is better to begin with, optimize later if necessary
- Match the output template (for grading purposes)

Programming tips: testing

- Create small, easily-verified test cases
  - Try to cover any edge cases you can think of
- Print intermediate output
  - Is the processed data as expected?
- Write incrementally, test as you go
  - Assertion statements are helpful

Programming tips: efficiency

- Remove unnecessary operations from loops

Programming tips: efficiency

- Remove unnecessary operations from loops
- Slow comparisons mean slow sorting

Programming tips: efficiency

- Remove unnecessary operations from loops
- Slow comparisons mean slow sorting
- Profiling tools
  - line_profiler (python)
  - gprof, valgrind (C/C++) [valgrind is also good for identifying memory leaks]
  - dprofpp (Perl, though I hope you're not using Perl)
  - Use various test data sizes to get an idea

Pointers in C++

Pointers in C++

- Pointers are memory addresses, they tell you where the actual things are

Pointers in C++

- Pointers are memory addresses, they tell you where the actual things are
- The address-of operator (&) obtains the memory address of a variable

Pointers in C++

- Pointers are memory addresses, they tell you where the actual things are
- The address-of operator (&) obtains the memory address of a variable
- The dereference operator (*) accesses the value stored at the pointed-to memory location

Pointers in C++

- Pointers are memory addresses, they tell you where the actual things are
- The address-of operator (&) obtains the memory address of a variable
- The dereference operator (*) accesses the value stored at the pointed-to memory location
- Passing arguments by reference saves memory and time

Pointers in C++

- Pointers are memory addresses, they tell you where the actual things are
- The address-of operator (&) obtains the memory address of a variable
- The dereference operator (*) accesses the value stored at the pointed-to memory location
- Passing arguments by reference saves memory and time
- -> is a shortcut for accessing attributes of pointed-to structures
  - a->element is the same as (*a).element

Arrays are pointers to blocks of memory

Arrays are pointers to blocks of memory

- Pointers “know” the size of the thing they point to

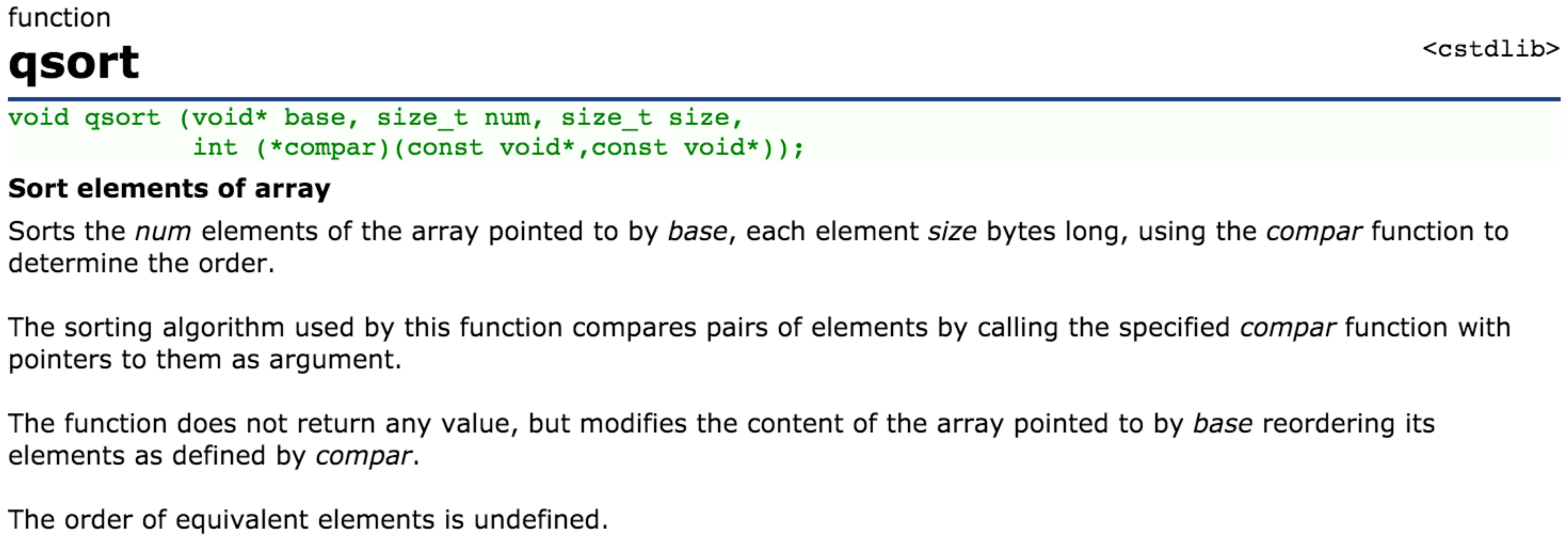
Arrays are pointers to blocks of memory

- Pointers “know” the size of the thing they point to
- Array indices are really just pointer arithmetic and dereferencing
  - a[12] is the same as *(a + 12)
  - &a[3] is the same as a + 3

Arrays are pointers to blocks of memory

- Pointers “know” the size of the thing they point to
- Array indices are really just pointer arithmetic and dereferencing
  - a[12] is the same as *(a + 12)
  - &a[3] is the same as a + 3
- Large arrays should be dynamically allocated (on the heap)
  - Remember to delete anything created with “new”!

Sorting in C++



function

qsort

<cstdlib>

```
void qsort (void* base, size_t num, size_t size,
            int (*compar)(const void*,const void*));
```

Sort elements of array

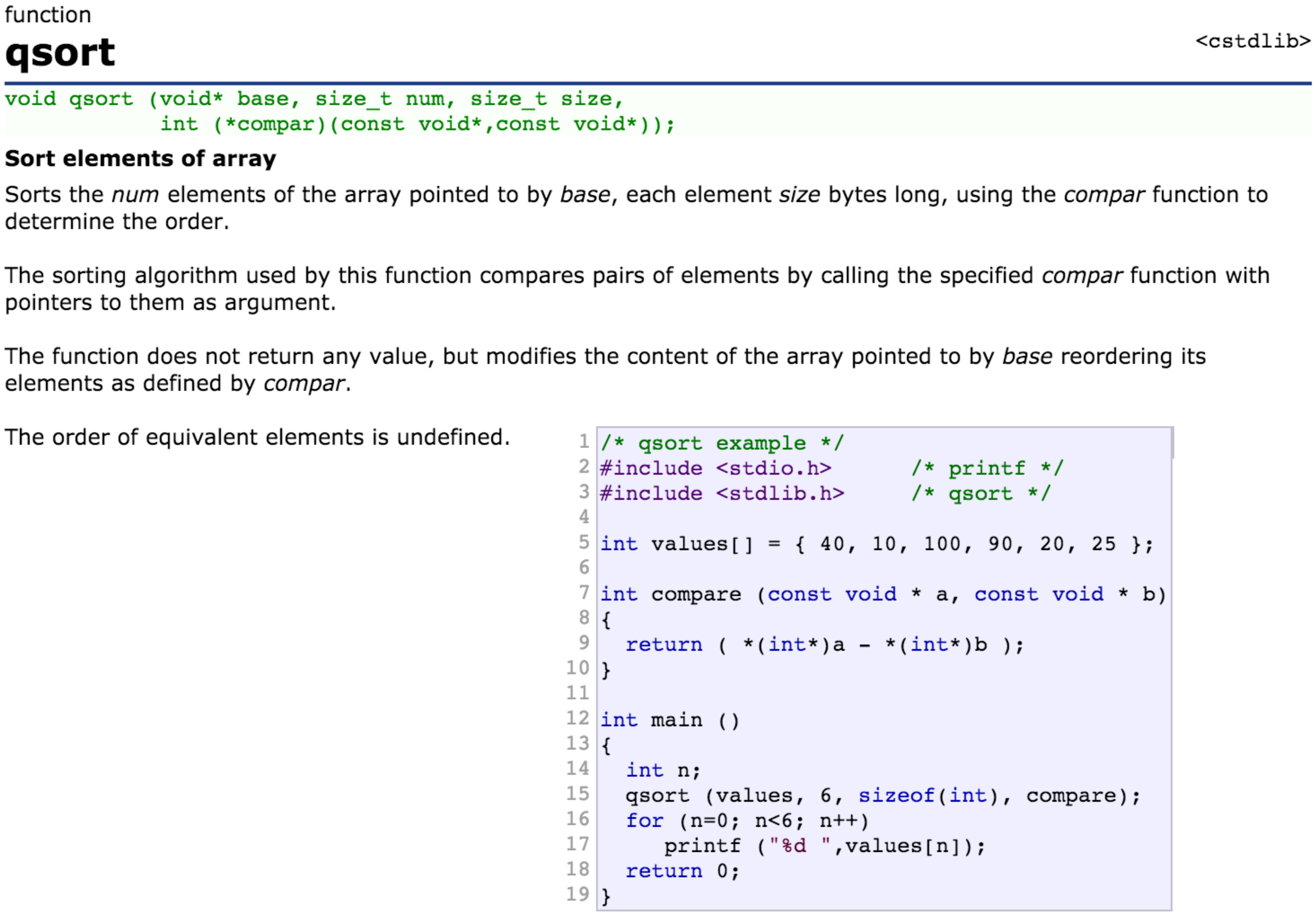
Sorts the *num* elements of the array pointed to by *base*, each element *size* bytes long, using the *compar* function to determine the order.

The sorting algorithm used by this function compares pairs of elements by calling the specified *compar* function with pointers to them as argument.

The function does not return any value, but modifies the content of the array pointed to by *base* reordering its elements as defined by *compar*.

The order of equivalent elements is undefined.

Sorting in C++



function

qsort

<cstdlib>

```
void qsort (void* base, size_t num, size_t size,
            int (*compar)(const void*,const void*));
```

Sort elements of array

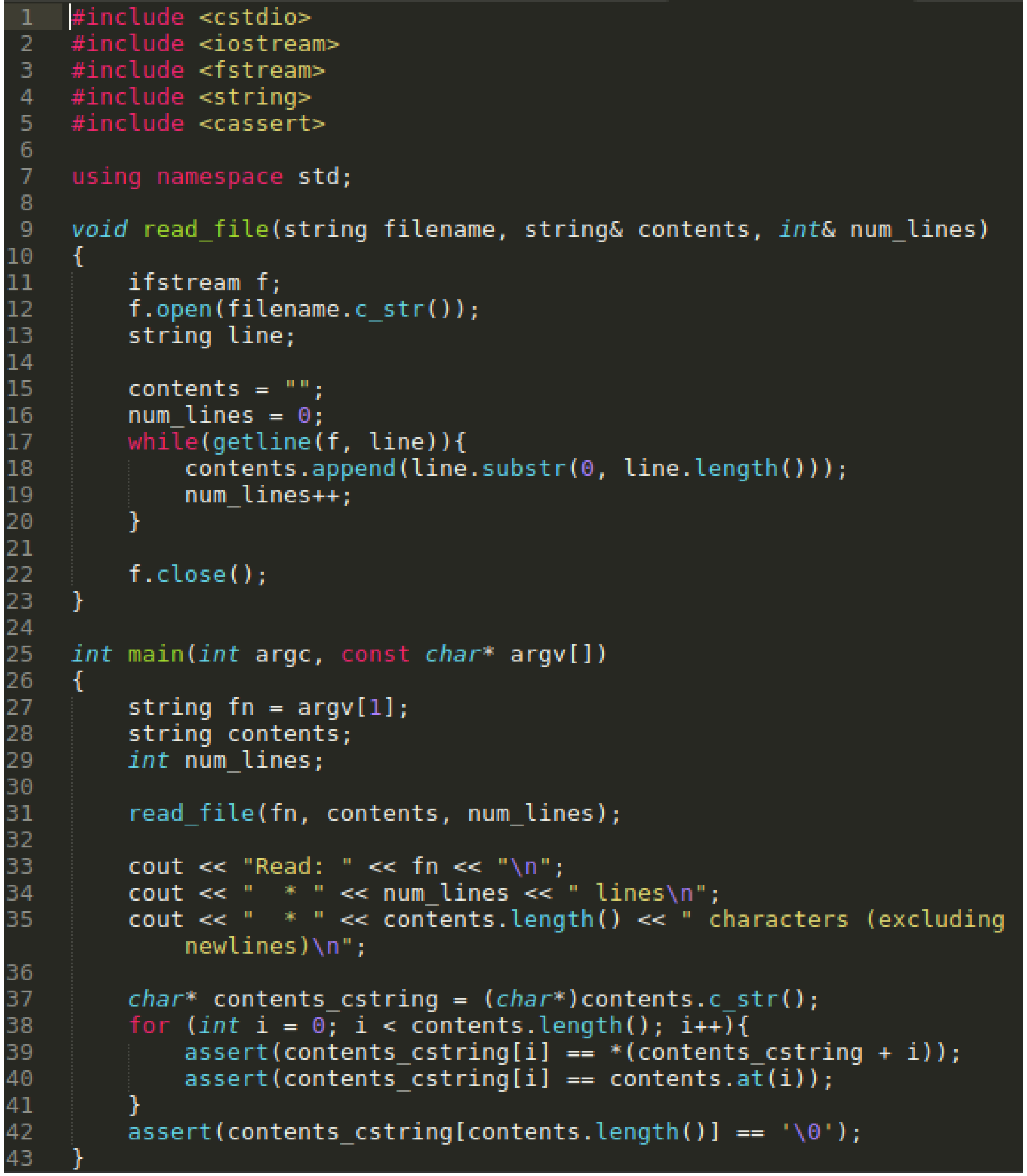
Sorts the *num* elements of the array pointed to by *base*, each element *size* bytes long, using the *compar* function to determine the order.

The sorting algorithm used by this function compares pairs of elements by calling the specified *compar* function with pointers to them as argument.

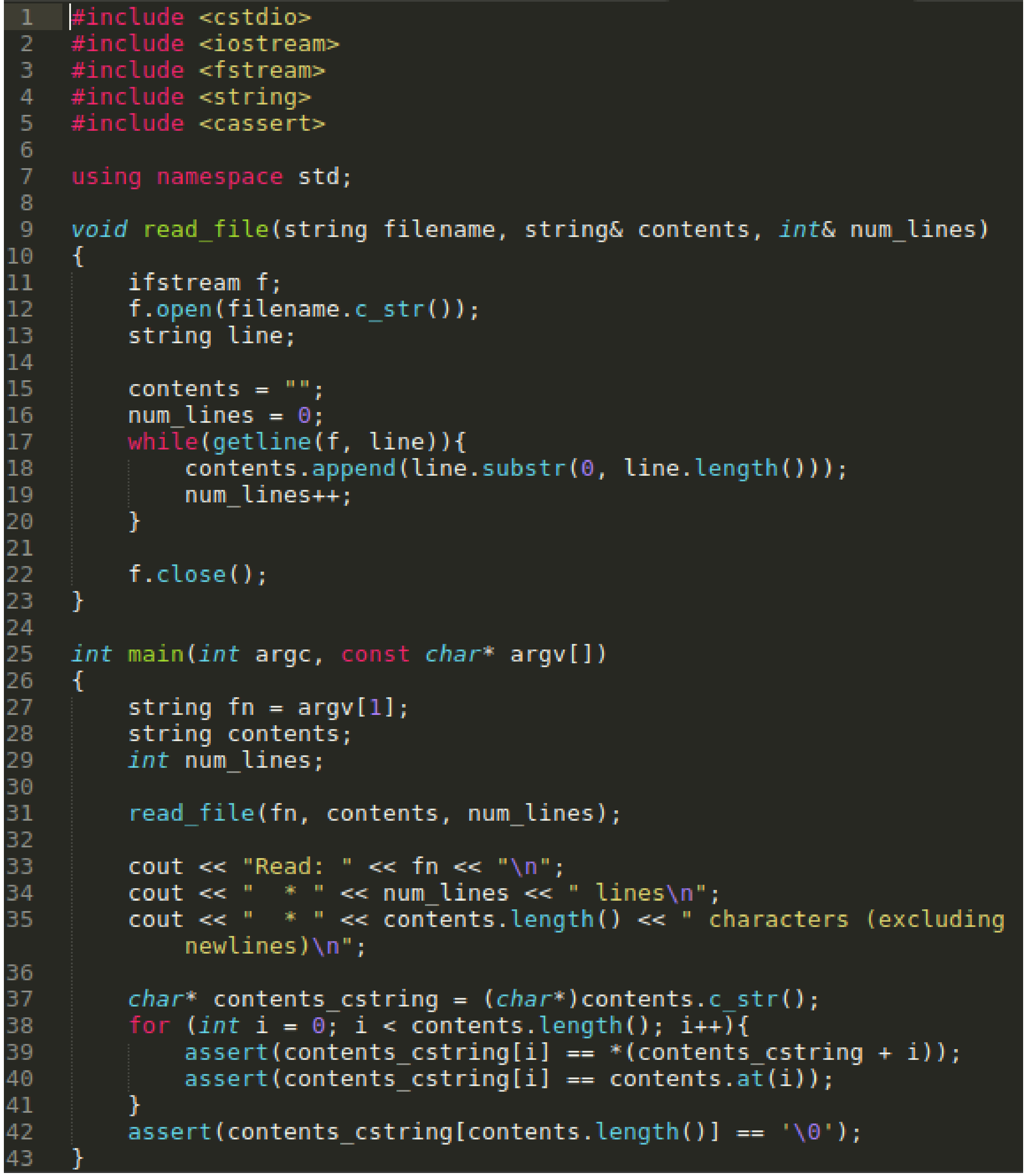
The function does not return any value, but modifies the content of the array pointed to by *base* reordering its elements as defined by *compar*.

The order of equivalent elements is undefined.

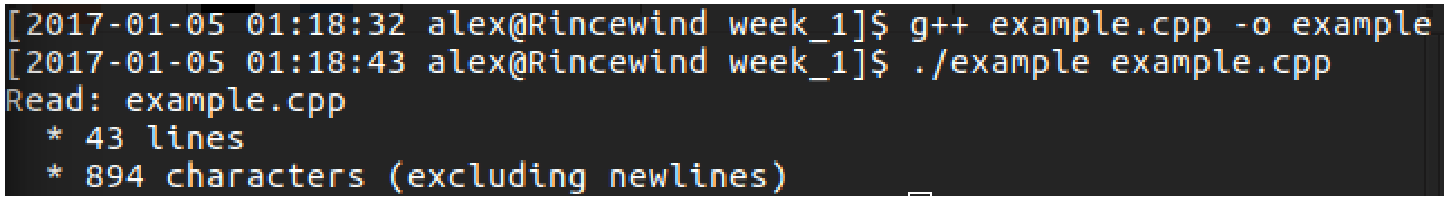
```
/* qsort example */
#include <stdio.h>      /* printf */
#include <stdlib.h>     /* qsort */

int values[] = { 40, 10, 100, 90, 20, 25 };

int compare (const void * a, const void * b)
{
  return ( *(int*)a - *(int*)b );
}

int main ()
{
  int n;
  qsort (values, 6, sizeof(int), compare);
  for (n=0; n<6; n++)
     printf ("%d ",values[n]);
  return 0;
}
```



```
#include <cstdio>
#include <iostream>
#include <fstream>
#include <string>
#include <cassert>

using namespace std;

void read_file(string filename, string& contents, int& num_lines)
{
    ifstream f;
    f.open(filename.c_str());
    string line;

    contents = "";
    num_lines = 0;
    while(getline(f, line)){
        contents.append(line.substr(0, line.length()));
        num_lines++;
    }

    f.close();
}

int main(int argc, const char* argv[])
{
    string fn = argv[1];
    string contents;
    int num_lines;

    read_file(fn, contents, num_lines);

    cout << "Read: " << fn << "\n";
    cout << "  * " << num_lines << " lines\n";
    cout << "  * " << contents.length() << " characters (excluding
        newlines)\n";

    char* contents_cstring = (char*)contents.c_str();
    for (int i = 0; i < contents.length(); i++){
        assert(contents_cstring[i] == *(contents_cstring + i));
        assert(contents_cstring[i] == contents.at(i));
    }
    assert(contents_cstring[contents.length()] == '\0');
}
```



```cpp
#include <cstdio>
#include <iostream>
#include <fstream>
#include <string>
#include <cassert>

using namespace std;

void read_file(string filename, string& contents, int& num_lines)
{
    ifstream f;
    f.open(filename.c_str());
    string line;

    contents = "";
    num_lines = 0;
    while(getline(f, line)){
        contents.append(line.substr(0, line.length()));
        num_lines++;
    }

    f.close();
}

int main(int argc, const char* argv[])
{
    string fn = argv[1];
    string contents;
    int num_lines;

    read_file(fn, contents, num_lines);

    cout << "Read: " << fn << "\n";
    cout << "  * " << num_lines << " lines\n";
    cout << "  * " << contents.length() << " characters (excluding
        newlines)\n";

    char* contents_cstring = (char*)contents.c_str();
    for (int i = 0; i < contents.length(); i++){
        assert(contents_cstring[i] == *(contents_cstring + i));
        assert(contents_cstring[i] == contents.at(i));
    }
    assert(contents_cstring[contents.length()] == '\0');
}
```

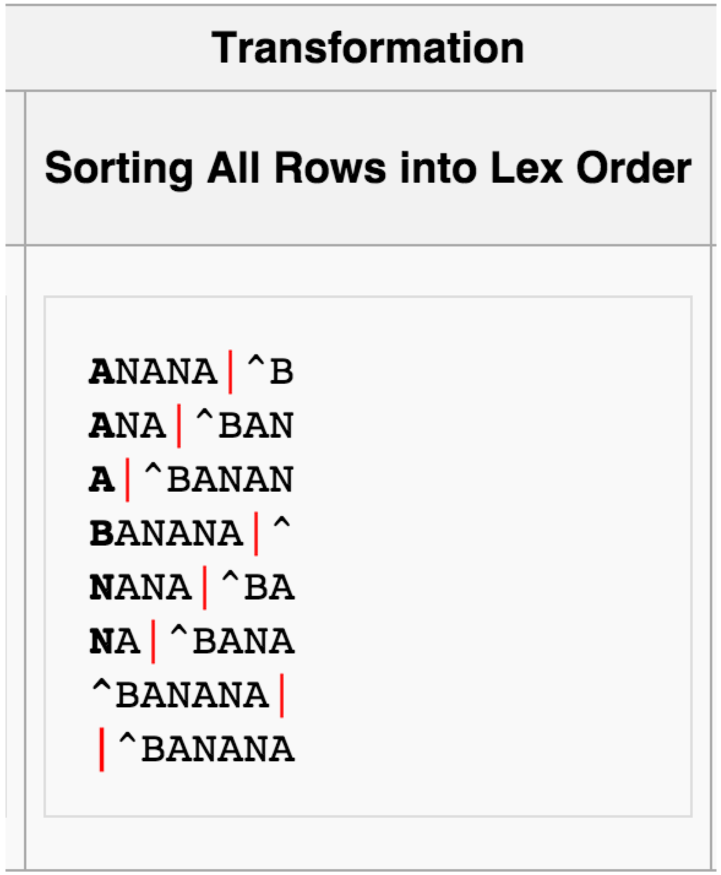


```
[2017-01-05 01:18:32 alex@Rincewind week_1]$ g++ example.cpp -o example
[2017-01-05 01:18:43 alex@Rincewind week_1]$ ./example example.cpp
Read: example.cpp
  * 43 lines
  * 894 characters (excluding newlines)
```

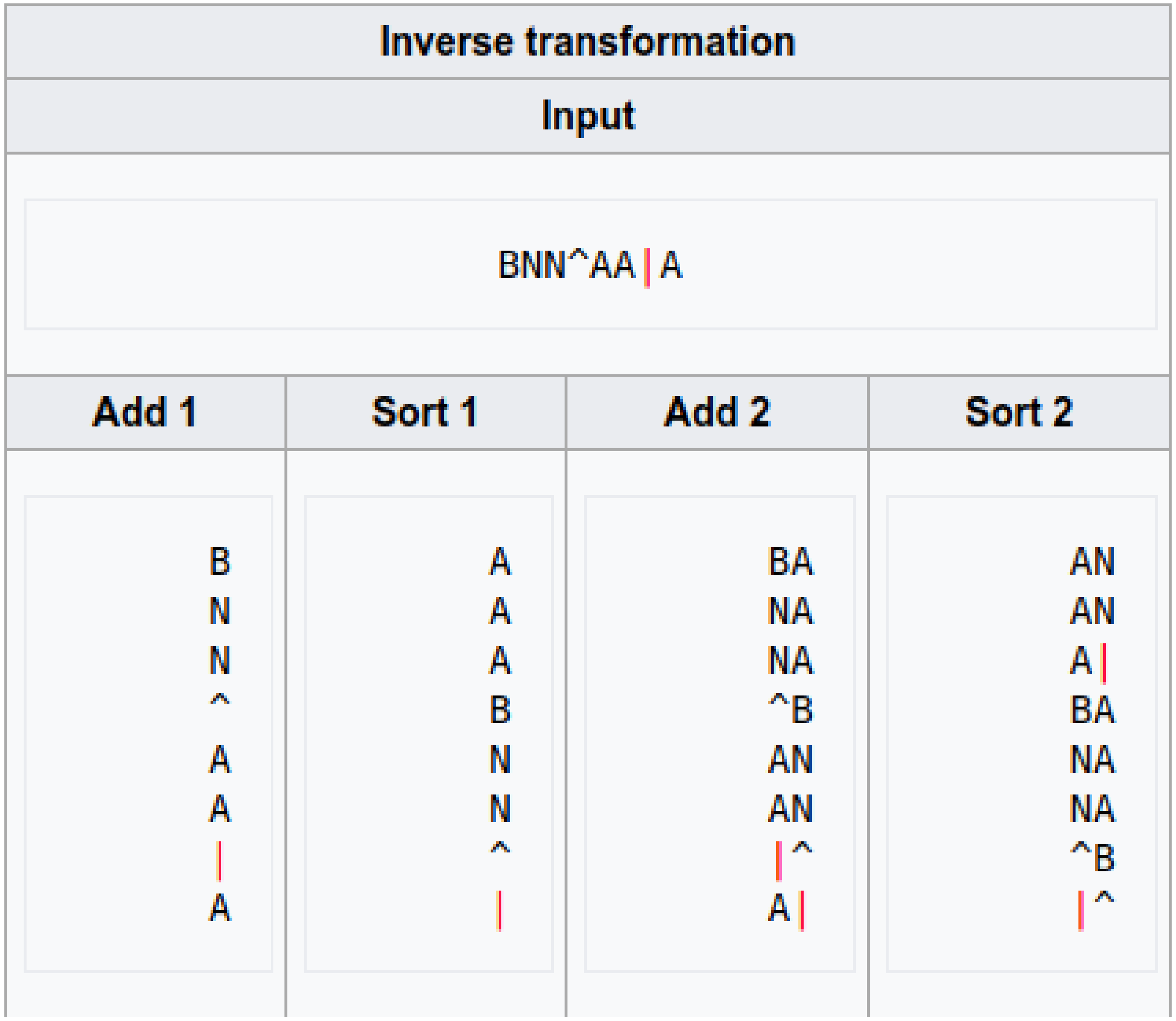
Suggestions for discussion topics?

- BLAST/multiple alignment
- Additional applications of HMMs (GENSCAN)
- Dynamic Bayesian Networks
- Frequentist vs. Bayesian statistics, probabilities vs. likelihoods
- Dynamic programming
- More programming tips
- More language/tool specifics: C++, R, Unix tools
- Machine learning
- Other suggestions?

Inverting the Burrows-Wheeler transform



Transformation
Sorting All Rows into Lex Order
ANANA|^B
ANA|^BAN
A|^BANAN
BANANA|^
NANA|^BA
NA|^BANA
^BANANA|
|^BANANA



Inverse transformation			
Input			
BNN^AA|A			
Add 1	**Sort 1**	**Add 2**	**Sort 2**
B	A	BA	AN
N	A	NA	AN
N	A	NA	A|
^	B	^B	BA
A	N	AN	NA
A	N	AN	NA
|	^	|^	^B
A	|	A|	|^