Frequently Asked Questions (General)

How should I read and write the data?

You must use `BinaryStdIn` and `BinaryStdOut`, which read and write sequences of bytes.

My programs do not work properly with binary data. Why not?

Be absolutely sure that you use only `BinaryStdIn` and `BinaryStdOut` for reading and writing data. Also, be sure to call either `BinaryStdOut.flush()` or `BinaryStdOut.close()` after you are done writing.

Why use `BinaryStdIn.readChar(8)` instead of `BinaryStdIn.readByte()` to read in 8 bits?

The primitive type char is a bit simpler to use because it is an unsigned type, whereas byte is a signed type. So, for example, you can not directly use a byte as an index into an array because it might be negative. You also need to be careful when converting from byte to char. For example, the statement `char c = (char) b;` first promotes the byte b to a 32-bit signed integer, then narrows it to a 16-bit signed integer, which is rarely what you want. Typically, you would want to use char `c = (char) (b & 0xff);` instead.

How do I convert between characters and integers?

Java's char type is a 16-bit signed integer. So, for example you can iterate over the characters in the alphabet with a for loop:

```
int R = 256;
for (char c = 0; c < R; c++) {
    ...
}
```

When you apply binary arithmetic or comparison operators to two char values, Java automatically promotes them both to type int. As a result, you can compare two char values with the binary comparison operators. Similarly, you can use a char as an index into an array.

I'm curious. Which compression algorithm is used in PKZIP? In gzip? In bzip2?

PKZIP uses LZW compression followed by Shannon–Fano (an entropy encoder similar to Huffman). The Unix utility gzip uses a variation of LZ77 (similar to LZW) followed by Huffman coding. The program bzip2 combines the Burrows–Wheeler transform, Huffman coding, and a (fancier) move-to-front style rule.

How can I view the contents of a binary file and determine its size?

Use `HexDump.java`, as in the assignment. The command-line argument specifies the number of bytes per line to print; if the argument is 0, all output except for the number of bits will be suppressed.

Frequently Asked Questions (CircularSuffixArray)

Must I use `BinaryStdIn` and `BinaryStdOut` with `CircularSuffixArray`?

The constructor and methods neither write to standard output nor read from standard input, so there is no need to use either `BinaryStdIn` or `BinaryStdOut`. You are free to use `System.out.println()` when testing in `main()`.

Can I form the n circular suffixes using the `substring()` method from the `String` data type?

No. Beginning with Java 7, Update 6, the `substring()` method takes time and space proportional to the length of the substring. So, explicitly forming the n circular suffixes in this way would take both quadratic time and space.

About how many character compares will I need to make in order to compare two circular suffixes from a string of length n?

It depends on the string. If the string has lots of redundancy (e.g., all As), you will need to make n character compares. If the string is “typical English text”, you can assume that number is about $\Theta(\log n)$.

Frequently Asked Questions (MoveToFront)

How should I read the binary input in `encode()`?

The input is a sequence of extended ASCII characters (0x00 to 0xFF). You should read them one character at a time using `BinaryStdIn.readChar()` until `BinaryStdIn.isEmpty()`.

How should I read the binary input in `decode()`?

Same as `encode()`.

My encode and decode methods take $\Theta(Rn)$ time in the best case? Is that okay.

No. It is fine to take `Θ(Rn)` time in the worst case. However, they should take `O(n+R)` time on inputs that arise when compressing typical English text.

Frequently Asked Questions (BurrowsWheeler)

How should I read the binary input in `transform()`?

The input is a sequence of extended ASCII characters (0x00 to 0xFF). You can read it using `BinaryStdIn.readString()`.

How should I read the binary input in `inverseTransform()`?

The input is an integer, followed by a sequence of extended ASCII characters (0x00 to 0xFF). You can read it using `BinaryStdIn.readInt()`, followed by `BinaryStdIn.readString()`.

My `BurrowsWheeler` works except for large files. What could be the cause?

Be sure that you read and write the number first as a 32-bit int, not a char.

Can I assume that `inverseTransform()` receives only valid inputs (e.g., that correspond to the output of `transform()`)?

Yes.

My `transform/inverseTranform` pipeline produces incorrect results for *periodic string* such as *abababab* and *aaa*. What could be the cause?

A string is *periodic* if it consists of two or more repeated copies of another string. The circular suffixes of a periodic string are not all distinct (e.g., the string *abababab* has 8 circular suffices but only two are distinct: *abababab* and *babababa*). For periodic strings, there is ambiguity in the definition of the next[] array because each circular suffix appears in multiple positions. Your implementation is free to resolve the ambiguity in any way that you choose: the sequence s[next[first]], s[next[next[first]]], s[next[next[next[first]]]], ... is always the same. However, you must take care to write the correct number of characters.

For the `Burrows–Wheeler` transform, in which order do I use to sort the suffixes?

Use lexicographic order to sort the suffixes, which is the natural order of the String data type.

For the `Burrows–Wheeler` inverse transform, does next[0] always equal first?

No. This is just a coincidence with the input string "ABRACADABRA!". Consider any two input strings that are cyclic rotations of one another, e.g., "ABRACADABRA!"and "CADABRA!ABRA". They will have the same sorted suffixes and t[] array—their only difference will be in the index first.

How much memory can my program consume?

The `Burrows-Wheeler` transform may use quite a bit, so you may need to use the – `Xmx` option when executing. You must use space linear in the input size n and alphabet size R. (Industrial strength `Burrows-Wheeler` compression algorithms typically use a fixed block size, and encode the message in these smaller chunks. This reduces the memory requirements, at the expense of some loss in compression ratio.) Therefore, depending on your operating system and configuration there may be some very large files for which your program will not have enough memory even with the `-Xmx` option.

I am running out of memory in the `transform()` method in `Burrows-Wheeler`. Any ideas? Be sure not to construct a new String object (e.g., by calling `substring()`) for each circular suffix created in `CircularSuffixArray`. It is fine to have multiple references to the same String object.

What is meant by "typical English text inputs"?

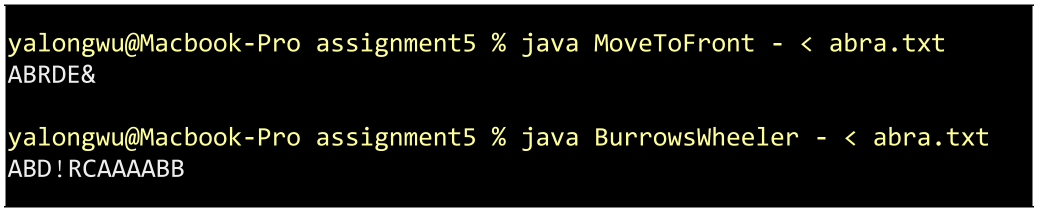
Inputs such as *Aesop's Fables*, *Moby Dick*, or your most recent essay. We do not mean inputs with very long repeated substrings (such as an input with 1 million consecutive As) or random inputs.

Testing

Input. To fully test your programs, you should use not only text files but also binary files (such as `.class` or `.jpg` files).

Debugging. Debugging MoveToFront and BurrowsWheeler present extra challenges because they produce binary output (instead of text output) on standard output.

- Viewing standard output in the terminal may produce unexpected results, as the bytes will be converted to Unicode and some of the corresponding characters may be unprintable.



```
yalongwu@Macbook-Pro assignment5 % java MoveToFront - < abra.txt
ABRDE&

yalongwu@Macbook-Pro assignment5 % java BurrowsWheeler - < abra.txt
ABD!RCAAAABB
```

In the move-to-front example, the first three bytes correspond to the ASCII characters A, B, and R but, then, some of the bytes correspond to unprintable characters and things get off track. In the Burrows–Wheeler transform example, the first four bytes (corresponding to the 32-bit integer) do not appear but the remaining bytes correspond to characters in abra.txt.

- To inspect the raw bytes (or bits), pipe the output through HexDump, as in the assignment specification.
- By default, any text output that your program produces via calls to System.out.println() will be intermixed with the binary output that your program produces via calls to BinaryStdOut.write(), which is probably not helpful. As an alternative, consider printing debugging information to standard error via calls to System.err.println(). By default, both standard output and standard error are sent to the terminal window. However, if you redirect standard output to a file, you'll see only the debugging information from standard error in the terminal.

```
yalongwu@Macbook-Pro assignment5 % java MoveToFront - < abra.txt > output.mtf
Whatever you write with BinaryStdOut.write() will go to the file output.mtf;
Whatever text you print with System.err.println() will appear here.
```

Reference solutions. For reference, we have provided the output of compressing aesop.txt and us.gif. We have also provided the results of applying each of the three encoding algorithms in isolation. Note that the binary file us.gif is already compressed.

To compare the contents of two files, you can use the following Bash command:

```
yalongwu@Macbook-Pro assignment5 % cmp aesop.txt us.gif
Aesop.txt us.gif differ: byte 1, line 1

yalongwu@Macbook-Pro assignment5 % cmp us.gif us.copy.gif
```

Compression ratio. You can use the ls command to determine the size of a file (in bytes).

```
yalongwu@Macbook-Pro assignment5 % ls -l
total 600
-rw-rw-r--@ 1 yalongwu staff      1 May  4 2012 a.txt
-rw-rw-r--@ 1 yalongwu staff     12 Mar 22 2009 abra.txt
-rw-rw-r--@ 1 yalongwu staff     19 Mar 22 2009 abra.txt.bwt.mtf.huf
```

```
-rw-rw-r--@ 1 yalongwu staff 191943 Nov 10 2005 aesop.txt
-rw-rw-r--@ 1 yalongwu staff  66026 Mar 22 2009 aesop.txt.bwt.mtf.huf
-rw-rw-r--@ 1 yalongwu staff  12400 Nov 10 2005 us.gif
-rw-rw-r--@ 1 yalongwu staff  12726 Mar 22 2009 us.gif.bwt.mtf.huf
……………………
```

For example, `aesop.txt` uses 191,943 bytes; after compression, it (`aesop.txt.bwt.mtf.huf`) uses only 66,026 bytes; the compression ratio is 66026/191943 = 0.344.

Timing your program. Use the following Bash commands for compression and expansion, respectively:

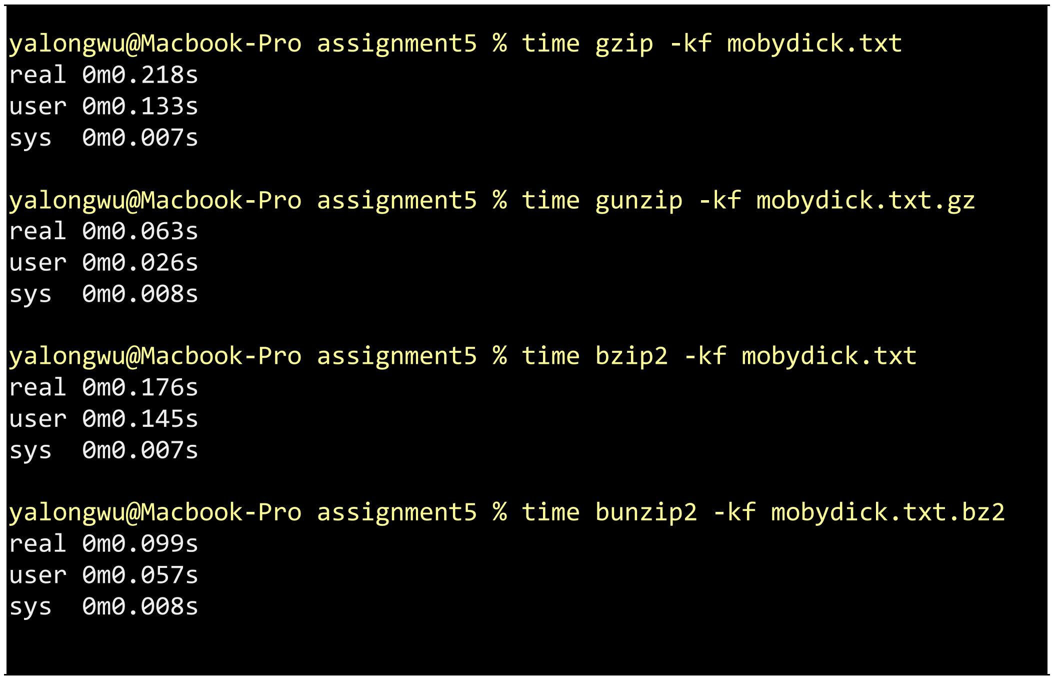
```
yalongwu@Macbook-Pro assignment5 % time java Huffman - < mobydick.txt - >|
mobyDickOutputFileName1

yalongwu@Macbook-Pro assignment5 % time java Huffman + < mobyDickOutputFileName1
- >| moby-copy.txt

yalongwu@Macbook-Pro assignment5 % time java BurrowsWheeler - < mobydick.txt |
java MoveToFront - | java Huffman - >| mobyDickOutputFileName2
real 0m1.341s
user 0m1.447s
sys  0m0.477s

yalongwu@Macbook-Pro assignment5 % time java Huffman + < mobyDickOutputFileName2
| java MoveToFront + | java BurrowsWheeler + >| moby-copy.txt
real 0m0.419s
user 0m0.750s
sys  0m0.372s
```

The “real” value is the wall clock time; the “user” value is the amount of CPU time spent in user-mode; the “system” value is the amount of CPU time spent in kernel mode. The CPU time may exceed the real time if your computer is using multiple CPUs.

Note that the internal Bash command time measures the time of the pipeline. If you use an external version of the command time (or an internal version associated with a different shell), it may measure only the first command in the pipeline.

We use >| instead of > to redirect standard output to a file and force overwriting (in case a file with that name already exists).

Timing using gzip or bzip2. If you are using Bash, you should have access to the following data compression and expansion commands: gzip, gunzip, bzip2, or bunzip2. You can time them using the following Bash commands:



```
yalongwu@Macbook-Pro assignment5 % time gzip -kf mobydick.txt
real 0m0.218s
user 0m0.133s
sys  0m0.007s

yalongwu@Macbook-Pro assignment5 % time gunzip -kf mobydick.txt.gz
real 0m0.063s
user 0m0.026s
sys  0m0.008s

yalongwu@Macbook-Pro assignment5 % time bzip2 -kf mobydick.txt
real 0m0.176s
user 0m0.145s
sys  0m0.007s

yalongwu@Macbook-Pro assignment5 % time bunzip2 -kf mobydick.txt.bz2
real 0m0.099s
user 0m0.057s
sys  0m0.008s
```

Possible Progress Steps

These are purely suggestions for how you might make progress. You do not have to follow these steps.

- Implement the `CircularSuffixArray`. Be sure not to create new String objects when you sort the suffixes. That would take quadratic space. A natural approach is to define a nested class `CircularSuffix` that represents a circular suffix implicitly (via a reference to the input string and a pointer to the first character in the circular suffix). The constructor of `CircularSuffix` should take constant time and use constant space. You might also consider making `CircularSuffix` implement the `Comparable<CircularSuffix>` interface. Note, that while this is, perhaps, the cleanest solution, it is not the fastest.
- Implement the `Burrows-Wheeler` transform, using the `CircularSuffixArray` class.
- The `Burrows-Wheeler` decoding is the trickiest part conceptually, but it is very little code once you understand how it works. (Not including declarations and input, our solution is about 10 lines of code.) You should find the key-indexed counting algorithm from the string sorting lecture to be especially useful.
- Implement the move-to-front encoding and decoding algorithms. Not including comments and declarations, our solutions are about 10 lines of code each. If yours is significantly longer, try to simplify it. Do not worry about optimizing the worst-case performance—the goal is good performance on typical English text inputs.