

Super User is a question and answer site for computer enthusiasts and power users. Join them; it only takes a minute:

Join

Here's how it works:



Anybody can ask a question



Anybody can answer



The best answers are voted up and rise to the top

## Why is a 7zipped file larger than the raw file? [duplicate]

**Possible Duplicate:**

[Why doesn't ZIP Compression compress anything?](#)

I tried 7zipping an .exe file but it actually became larger.



Is this the expected result?

[compression](#) [zip](#) [7-zip](#)

edited Mar 20 at 10:17

Community ♦  
1

asked Aug 21 '12 at 10:48

IMB  
2,368 16 51 83

marked as duplicate by [Oliver Salzburg](#) Aug 21 '12 at 17:20

This question has been asked before and already has an answer. If those answers do not fully address your question, please [ask a new question](#).

- 3 Yes, it's the expected result. Why? Because when something is already compressed (=using the smaller possible space), it can't be compressed further. – [woliveirajr](#) Aug 21 '12 at 14:21
- 4 Just to add to everyone else's - since this exe file specifically is an installer, most of its content is probably a zip or cab archive. You would not get the same results from a normal exe file (but most normal exe files won't be 145 megabytes) – [Random832](#) Aug 21 '12 at 14:23
- 1 Explanation using basic logic only: Compression finds for a raw file a UNIQUE zipped file, and for zipped file UNIQUE raw (uncompressed) original file. Imagine you have 8-bit files and want to compress them into 5-bit files. There are 256 unique 8-bit files, but only 32 unique 5-bit files(!) So some 8-bit files must be compressed into the same 5-bit file(!). And if 2 different raw files compressed into same ZIP file, which one do you want to get after decompression? For any zipping method, if there exist files that become smaller after zipping, there must exist files, that become larger(!) – [Ivan Kuckir](#) Jan 19 '15 at 22:44

## 5 Answers

It comes down to a concept called **entropy**. See [Wikipedia](#).

The basic idea is that, if there existed a compression operation that could *always* make a file smaller, then logic dictates that said compression operation would be able to reduce any file to 0 bytes and still retain all the data. But this is *absurd*, because we know that 0 bytes can not convey any information at all. So we have just proven that there **can not exist** a compression algorithm that always makes its input smaller, because if that were the case, any information could be stored in 0 bytes -- but 0 bytes implies the *absence* of information, so you can't simultaneously have *no* information and *all* information. Hence, it's absurd.

Due to this theoretical concept, every compression program you ever use is going to *increase* the size of (or at best, maintain the same size of) **some** input. That is, for any compression algorithm you design or use, there will be certain inputs that will come out smaller, and some that will not.

Already-compressed data is generally a terrible candidate for further compression, because most lossless compression algorithms are based on the same theoretical principles. It *is*

possible to compress poorly-compressed data even further; but this is less efficient than simply compressing it with the best-available algorithm from the original data to begin with.

For example, if you had a 100 MB text file and compress it using the regular Zip algorithm, it might get compressed down to 50 MB. If you then compress the Zip file with LZMA2, you might get it down to 40 or 45 MB, because LZMA has a *higher compression ratio* for most compressible data than Zip does. So it stands to reason that it can also compress Zip data, because Zip doesn't completely suck all the entropy out of it. But if you eliminate the Zip container entirely, you may be able to get it even smaller by compressing the raw text with LZMA2, potentially yielding something on the order of 30 - 35 MB (these are just "air numbers" to illustrate the concept).

In the case of that binary you're trying to compress, it's *larger* because the 7-Zip file format has to create its own internal structure and pack the already-compressed executable's data into the 7-Zip format. This contains things like a dictionary, a file header, and so on. These extra data are usually more than offset by the savings of compressing the data itself, but it appears that the executable you're trying to compress is already compressed with some form of LZMA; otherwise, it would likely shrink the size of the executable or very slightly increase it, rather than increasing it by 2 MB (which is a lot).

answered Aug 21 '12 at 13:30



allquixotic

28.7k ● 6 ● 87 ● 115

btw the most important part for answering this question is right at the end: "This contains things like a dictionary, a file header, and so on. These extra data are usually more than offset by the savings of compressing the data itself, but it appears that the executable you're trying to compress is already compressed with some form of LZMA" – [jhocking](#) Aug 21 '12 at 15:24

6 @jhocking: No, the most important part is towards the middle: *"Every compression program you ever use is going to increase the size of ... some input."* 7zip's file-format has a dictionary/file-header/etc, but even if 7zip used an algorithm that didn't have any of those things, we're still guaranteed that some (in fact, most) inputs will have outputs that are as-large-or-larger than the inputs themselves. This is a basic fact of information-theory, and has nothing to do with file-headers. – [BlueRaja - Danny Pflughoeft](#) Aug 21 '12 at 15:44

2 @Mehrdad Sure: Just write a "compression" algorithm that always returns the original input. There; done. :P ... Aside from that, no – any compression algorithm that's an algorithm at all is going to have *some* metadata, even if it's just one bit at the start of the file that indicates whether or not the file is compressed (0 == uncompressed, 1 == compressed). If you're going to modify the contents of the file *AT ALL*, you need *some* metadata. And if you're modifying the contents, you're going to make *some* inputs larger. – [allquixotic](#) Jul 23 '14 at 13:36

1 However, if your question was "Is there any compression algorithm that does not increase the length of the input beyond a fixed amount of metadata", the answer is: I don't know, but it should be theoretically possible to do it. Easy, in fact. All you have to do is develop a container format that can *either* contain the original file, or a compressed data stream. Then, when you create the archive, try to compress: if the compressed size is larger than the input, just store the original input and pack your metadata in front. The file size will increase, but if the metadata is small (cont'd) – [allquixotic](#) Jul 23 '14 at 13:39

2 @Mehrdad: "Is there any compression algorithm (however poor) that does not increase the length of any input?" – The answer is no. There are  $2^{n+1} - 1$  possible messages of size n-bits or less. Our algorithm must map each one of these to a **unique** output. If even one of these gets mapped to a value with fewer bits, another value must necessarily get mapped to one with more. – [BlueRaja - Danny Pflughoeft](#) Jan 27 '15 at 23:30

The underlying compression algorithms used in 7z are *lossless*. Which means you can iteratively compress-decompress a file many times. Furthermore, after each iteration the file will remain *exactly* the same.

Unfortunately, you cannot expect a *lossless* compression algorithm be applied many times with always a positive result. There is a strict boundary which it cannot jump over. Roughly, this boundary depends on how closely an input sequence ensembles random data. Above all, lossless algorithms are used for files compression, Internet HTML data transfers, backups and other operations that expect an output file to be decompressed into exactly the same original input file.

In contrast to *lossless* compression, you may always expect a file size decrease after compression with *lossful (or lossy) compression algorithms*. The down side is that you cannot *exactly* restore an original file after a single compress-decompress iteration. These algorithms are most famous for audio/video/image transmissions and storage.

*bzip2*, *LZMA*, *LZMA2* and other algorithms used by 7z format are all *lossless*. Therefore there will be a limit after which it can no longer compress. On top of that, executable images (.exe) are usually highly compressed files. 7zip as many others compression tool embeds some metadata, which in fact can make the output file larger.

**Brain teaser: what if we did have a lossless algorithm that can always decrease a file's size?**


In this case, you shall always see that the compressed file is smaller than the input file. See a comment below why it's not possible.

edited Aug 21 '12 at 15:50

answered Aug 21 '12 at 15:32



oleksii

- 5 Proof by contadiction. **Hypothesis:** Suppose it is always possible to compress a file with a lossless algorithm. **Step1.** Single compression makes an output file smaller at least by one bit. If so, after a number of iterations we will end up with a file that has only two bits. **Step 2** Next iteration makes a file of a size of 1 bit. **Step 3** But the compression algorithms is lossless, which means there is only one valid decompression allowed. Clearly you cannot restore 2 original bits from 1 compressed bit - you will have to make a guess. **The last point violates the hypothesis.** – [oleksii](#) Aug 21 '12 at 15:33 

You can't guarantee an algorithm that makes the file smaller but you can guarantee one that won't increase the size by applying no "compression" in those cases. In order to really have no file size increase though, you would have to indicate this out of band (e.g. in the file name). – [jeteon](#) Aug 20 '15 at 7:39

@jeteon I am not sure what you trying to say. – [oleksii](#) Aug 20 '15 at 10:49

I was just adding that since you always have the option of not compressing the input, you can have a compression program that will not compress the file at all at worst. Basically, if you determine that the compressed version is larger than the uncompressed version, then you just leave it. You would also then have to indicate somehow that this is the case without adding to the size of the output so the decompressor knows the file wasn't compressed. The only way to do this without increasing the file size, is doing something like changing the file name. – [jeteon](#) Aug 20 '15 at 12:31

@jeteon oh, I see. Yep, make sense. – [oleksii](#) Aug 20 '15 at 14:19

If the original executable was already compressed (or contained heavily compressed data or noncompressible data) then compressing it will increase the size.

answered Aug 21 '12 at 11:01



[PhonicUK](#)

2,796 ● 1 ● 13 ● 15

Most compression algorithms use whats called a symbol table, basicly just peices of the file it uses as elements it CAN compress. This, of course, creates some overhead in the file but usually results a much smaller file.

In already compressed files, it still creates a set of symbols, but there's very little that can be reduce the size on. In your case, the the symbol table of the already compressed file is probably in the neighborhood of 2 MB or probably more if it did manage to do some compressing.

edited Aug 21 '12 at 16:36

answered Aug 21 '12 at 13:28



[Chad Harrison](#)

3,340 ● 10 ● 28 ● 55

the compressing ideaa:

the compression software creates a list of files and eliminates the duplicate content.

when compressing already compressed files, you may get your compressed files bigger than the original.

answered Aug 21 '12 at 13:20



[fromnaboo](#)

109 ● 1