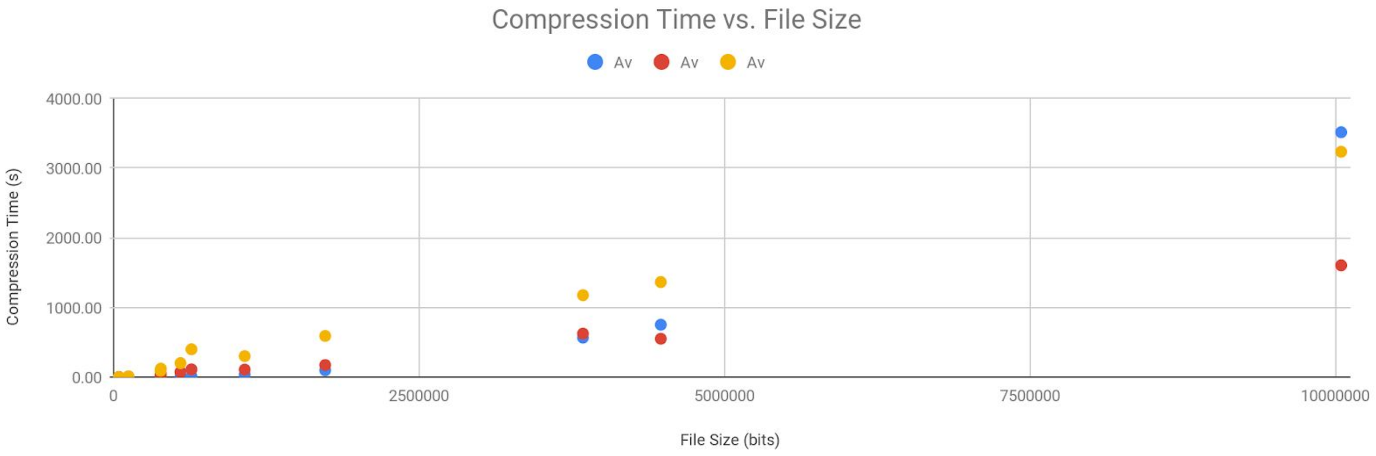
Table of Results:

File Type		File Size (bits)	Compression Time (s)											
			255				65535				262143			
			Av	PC2	Min	Max	Av	PC2	Min	Max	Av	PC2	Min	Max
jpg	1	126800	0.22	0.32	0.21	0.26	8.97	10.66	8.78	9.10	11.33	13.54	11.10	11.57
	2	48144	0.12	0.08	0.11	0.13	1.75	1.86	1.72	1.76	1.79	1.75	1.76	1.91
	3	552496	3.44	2.09	2.86	4.56	74.77	75.59	73.66	77.72	201.20	222.93	193.69	225.65
png	1	3845736	567.85	350.37	444.93	670.27	624.84	563.17	578.26	655.59	1176.10	1259.55	1169.42	1188.38
	2	389320	1.87	0.99	1.80	1.95	35.39	35.13	33.25	36.18	92.22	95.89	91.08	93.68
txt	1	1077768	37.28	23.65	34.61	40.29	108.45	103.42	107.76	109.60	301.79	305.83	300.37	302.77
	2	4482552	752.34	438.58	740.18	802.73	551.05	576.44	524.26	567.03	1364.65	1564.95	1360.19	1369.50
	3	10049336	3514.69	2084.31	3118.30	3946.22	1604.49	1525.59	1536.80	1645.61	3234.87		2982.17	3559.92
html	1	1736480	98.86	57.77	85.51	109.16	173.70	198.51	166.56	178.06	591.43	641.74	587.72	600.79
	2	391824	0.80	1.07	0.72	1.02	41.74	49.18	39.63	43.15	121.58	151.49	118.69	123.39
	3	642216	6.51	5.07	6.15	7.02	111.33	116.80	107.89	115.22	399.81	446.16	372.32	426.13

			Average Decompression Time (s)											
			255				65535				262143			
			Av	PC 2	Min	Max	Av	PC 2	Min	Max	Av	PC 2	Min	Max
jpg	1	126800	0.035	0.037	0.032	0.042	0.024	0.036	0.020	0.031	0.023	0.031	0.020	0.027
	2	48144	0.023	0.016	0.021	0.026	0.016	0.016	0.014	0.018	0.018	0.014	0.015	0.022
	3	552496	0.205	0.157	0.187	0.240	0.135	0.099	0.130	0.146	0.118	0.083	0.112	0.136
png	1	3845736	2.860	2.063	2.209	3.356	2.243	1.683	2.097	2.359	1.478	1.313	1.444	1.514
	2	389320	0.199	0.137	0.188	0.214	0.114	0.098	0.104	0.125	0.106	0.073	0.101	0.112
txt	1	1077768	0.576	0.403	0.548	0.594	0.379	0.542	0.369	0.400	0.359	0.362	0.341	0.406
	2	4482552	3.696	2.530	3.615	3.814	1.934	1.819	1.747	2.144	1.407	1.349	1.363	1.493
	3	10049336	12.176	7.982	11.358	13.241	7.859	6.647	7.657	8.012	5.475		5.225	5.932
html	1	1736480	0.916	0.601	0.912	0.970	0.411	0.308	0.380	0.445	0.483	0.475	0.473	0.517
	2	391824	0.086	0.109	0.084	0.092	0.031	0.045	0.027	0.037	0.026	0.035	0.024	0.031
	3	642216	0.382	0.389	0.357	0.427	0.295	0.236	0.281	0.328	0.250	0.251	0.202	0.296

			Compressed File Size (bits)					Compression Ratio					Triples		
			255	256	65535	65536	262143	255	256	65535	65536	262143	255	65535	262143
jpg	1	126800	219461	322533	183233	241865	239225	0.578	0.393	0.692	0.524	0.530	12909	7329	7249
	2	48144	84294	123933	75308	99404	99404	0.571	0.388	0.639	0.484	0.484	4958	3012	3012
	3	552496	825154	1208858	626208	826592	780260	0.670	0.457	0.882	0.668	0.708	48538	25048	23644
png	1	3845736	6959757	10228133	5504058	7265354	6535922	0.553	0.376	0.699	0.529	0.588	409397	220162	198058
	2	389320	701921	1031633	552308	552308	683141	0.555	0.377	0.705	0.534	0.570	41289	22092	20701
txt	1	1077768	1874139	2749333	1400258	1848338	1662185	0.575	0.392	0.770	0.583	0.648	110243	56010	50369
	2	4482552	6514068	9505983	2849233	3760985	3054224	0.688	0.472	1.573	1.192	1.468	383180	113969	92552
	3	10049336	14333184		6697664		7300664	0.701		1.500		1.376	843128	267906	221232
html	1	1736480	2485884	3627483	1045233	1379639	1122239	0.699	0.479	1.661	1.259	1.547	146228	41809	34007
	2	391824	464074	678883	126508	166988	146297	0.844	0.577	3.097	2.346	2.678	27298	5060	4433
	3	642216	962718	1408083	270633	357233	317402	0.667	0.456	2.373	1.798	2.023	56630	10825	9618

Compressor Analysis:



As expected, compression time increases with file size and window size. This can be seen in the graph above, with the windows size of 255b being blue, 65535b being red, and 262144b being yellow. However, looking at the 255b window, the compression time increases past that of the 65535b and 262144b windows. This could mean compression time increases exponentially with file size. On the other hand, window size seems to change how quickly the gradient of the line increases.

Also, looking at the table, we see that the maximum results are mostly similar in extremity to the minimums, with a few instances where the maximum was clearly more extreme the the minimum. Perhaps the sample size is too small, as it's expected that the maximums are more extreme than the minimums - the CPU will constantly try to run the program as fast as possible, but often background processes might cause the program to run slower periodically.

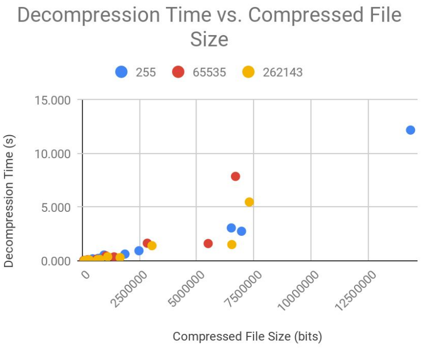
Comparing by the different inputs, it does seem like different inputs have an effect on the compression time - though file size is important, if we take a look at html2 and png2, we see that at the larger window sizes, html2 actually takes longer to encode than png2, despite being slightly smaller. This does not necessarily mean htmls take longer to encode than pngs in general, however, as if we take the compressed bits per second (bps) of html1 and png1, we find that they're still quite similar, with html1 not being as fast (2936bps and 3270bps respectively).

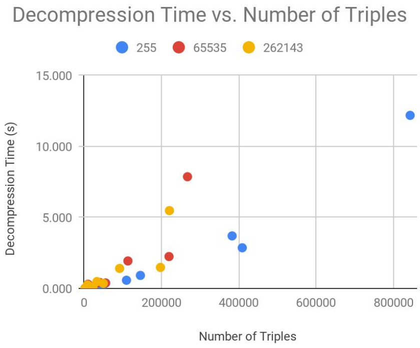
Another interesting result of mention is the way compression time reduces in the larger files, going from a 255b window to a 65535b window. Looking at txt2 and txt3, both initially see a decrease in compression time. This implies that smaller window sizes do not guarantee a faster compression time. This may be because the tradeoff of having to create a larger file offsets the benefits of a smaller window.

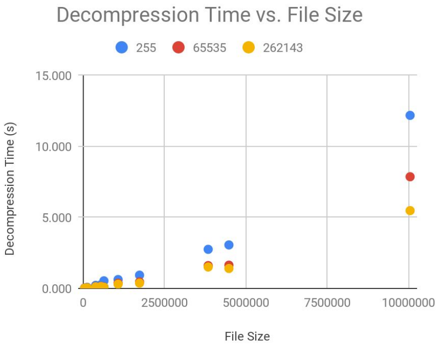
Conclusions:

- Increasing file size generally increases compression time
- Compression time at a certain size may vary for different files
- Increasing window size increases compression time up to a window size
- Minimum compression times are generally similar in extremity to maximums
- Minimising window size does not necessarily minimise compression time, especially with larger files

Decompressor Analysis:







Decompression time increase as the size of the file being decompressed increases, as shown by the graph. Because of this, anything that has an effect on the size of the compressed file has an effect on the decompression time:

- Window size generally decreases decompression time, as shown in the graphs above, with larger window sizes having a shorter decompression time. This likely has to do with how increasing window size decreases the number of triples generated during compression. Having less triples results in faster decompression.
  (The number of triples generated stops decreasing when the window size is equal to the file size, and depending on the algorithm, if it doesn't check for the file size, the compressed file may continue to grow, possibly resulting in increased decompression times. Look at the results for jpg2)
- File size generally changes the size of the compressed file, although it is far less certain than with window size/number of triples, as certain files compress better than others. As seen above, the decompression/file size graph seems exponential.

Looking at the minimum and maximum decompression times, it seems that the minimum and maximums are similarly extreme. This may be a result of how quickly the decompression occurs, meaning there are far less opportunities for delays in the program.

Here, the output file type does not seem to make a difference to the decompression time. This would make sense, as the input file to be decompressed is technically all of the same 'type'.

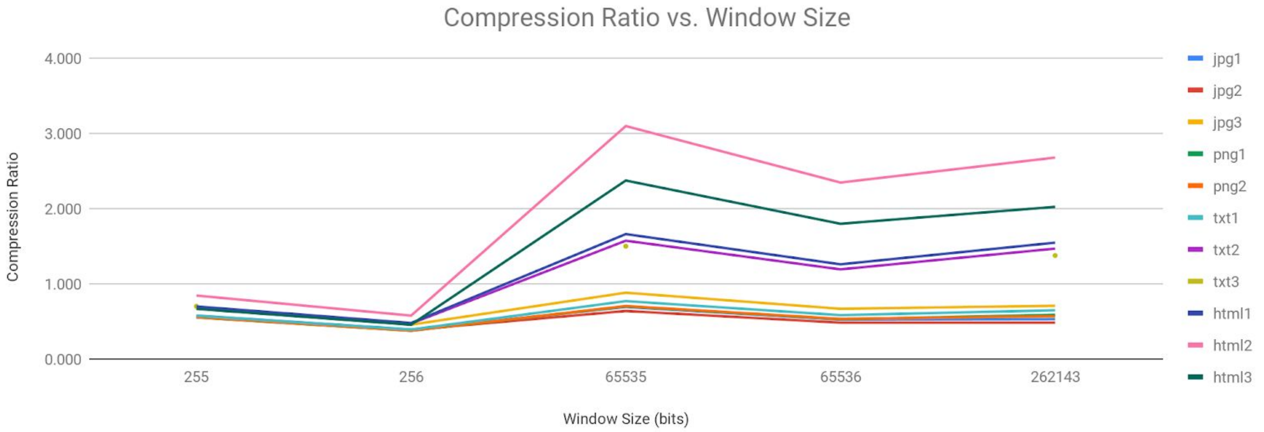
Conclusions:

- Decompression time increases with the size of the compressed file
- Increasing window size decreases the decompression time up to a certain point
- File type has little no effect on decompression time

Note on (De)Compression Times in Results Table:
Av represents the average of a computer running a desktop AMD Ryzen 5 processor. PC 2 represents the average of another computer running a mobile Intel i7 processor. From the results gathered, the Intel processor generally outperforms the AMD processor, with several outliers. This is unexpected, and likely to be an experimental error, as a desktop processor is generally far more powerful than a mobile one. If I had more time, I would only compress/decompress one file at a time - at the moment, several files are processed at any given moment using the multiprocessing module in python, in order to save time. However, this may have led to the reduced CPU performance for the individual cases. Proof of underperformance can be found in the results of jpg1 and html2 - these were run separately from the rest and yielded better results than that of PC 2.

All data from the AMD system are taken from the average of 10 results. For the Intel system, the data is taken from the average of 3 results.

Compression Ratio:



In these tests, we use the window sizes of 255b (stored as 1B), 256b (stored as 2B) 65535b (stored as 2B), 65536b (stored as 3B), and 262143b (stored as 3B).

The compression ratio for a given file seems to change in such a way that there is a single window size that gives the maximum compression ratio. In the case of the results gathered, for all the files, the greatest compression occurs when the window size is 65535b, and is less when the window size is either 255b or 262143b. This implies that there is a point at which increasing the window size does not result in better compression.
Moreover, it is trivial to see that going from a 255b window size to a 256b window size will result in a sudden large decrease in compression ratio. This is because a distance of 255 can be stored using 1B/8b, whereas a distance of 256 requires 9b to store, hence 2B. This is made even clearer in the graph above.

File size may have a significant effect on where the optimal window size is - for example, for files up to 10000000b as tested here, a window size of 65535b seems to be the most optimal. However, for much larger file sizes, say in the range of 1000000000b, a larger window size may be more optimal - the extra byte(s) required to store the triples are counteracted by the reduced number of triples generated.

However, we can be certain that the a window size of (2^8n)-1 will always produce the optimal number of triples, with n being the number of bytes to store the distance. This is trivial - with a larger window size, less triples are generated. Hence, the largest window size that can be stored in n bytes will always generate the least amount of triples that can be stored with an n-byte distance.

Overall, taking a look at the results, we see that very few files actually end up being compressed. The ones that are compressed are the text based ones (htmls, one txt file), which is understandable as the text based files have far more regularity than others. Also, it seems that quite often, files of the same type yield similar compression ratios. To explain the difference in compression ratio between txt1 and txt2/txt3, txt1 is a random string, hence its content differs from txt2/txt3, which are books. Similarly, html1 is a webpage transcribing a book, whereas html2/html3 are more standard websites. Ultimately, this shows that the content is a main factor for compression ratio, although certain file types generally compress better.

Conclusions:

- Compression ratio varies with window size
- Compression ratio has a maximum
- Maximum compression ratio varies from file to file
- Optimal window size is always (2^8n)-1, n being the number of bytes to store distance
- Optimal window size is dependent on file size
- Bitwise compression not useful on most files
- Contents of the file will result in differing compression ratios within the same filetype

Comparison to Other Compression Techniques:

LZSS:

Overall, LZSS compresses marginally better than LZ77, as seen in the results. However, it also takes a bit more time to compress/decompress in general, hence it would be expected that it yields better results. These results are taken from an algorithm I've coded, building upon the original LZ77 compressor.

File Type		File Size (bits)	Compression Time (s)			Compression Ratio			Decompression Time (s)		
			255	65535	262144	255	65535	262144	255	65535	262144
jpg	1	126800	0.288	8.784	14.572	0.618	0.675	0.570	0.062	0.031	0.047
	2	48144	0.091	1.486	3.161	0.615	0.637	0.557	0.021	0.015	0.027
	3	552496	2.250	66.472	202.134	0.713	0.856	0.723	0.201	0.092	0.114
png	1	3845736	1215.719	513.045	1169.878	0.595	0.672	0.583	0.595	0.672	0.583
	2	389320	1.449	30.588	88.144	0.599	0.681	0.584	0.172	0.078	0.101
txt	1	1077768	14.018	82.988	254.429	0.611	0.741	0.637	0.463	0.217	0.220
	2	4482552	1074.882	545.366	1439.557	0.728	1.512	1.431	3.014	1.911	1.544
	3	10049336	5658.806	1963.985	3275.718	0.740	1.446	1.342	12.118	10.253	5.613
html	1	1736480	90.022	179.182	547.799	0.744	1.604	1.514	0.671	0.322	0.320
	2	391824	1.140	40.973	133.390	0.922	3.043	2.733	0.123	0.035	0.037
	3	642216	3.212	96.022	354.215	0.712	2.313	2.046	0.389	0.186	0.193

We can build off of LZSS to create DEFLATE, which compresses the LZSS output even further through Huffman-encoding.

Bitwise Compression vs Bytewise Compression:

I've reimplemented the LZ77 compressor such that it reads byte by byte instead of bit by bit. The results are as follows:

File Type		File Size (bits)	Compression Time (s)			Compression Ratio			Decompression Time (s)		
			255	65535	262144	255	65535	262144	255	65535	262144
jpg	1	126800	0.054	0.107	0.108	0.567	0.543	0.434	0.032	0.027	0.027
	2	48144	0.022	0.297	0.351	0.566	0.520	0.416	0.013	0.009	0.010
	3	552496	0.238	1.397	1.373	0.688	0.737	0.589	0.096	0.085	0.094
png	1	3845736	6.889	15.953	34.531	0.549	0.659	0.573	1.547	1.090	1.029
	2	389320	0.174	0.705	0.726	0.557	0.601	0.481	0.081	0.067	0.058
txt	1	1077768	0.627	2.746	2.536	0.683	0.787	0.836	0.216	0.146	0.112
	2	4482552	4.285	14.022	34.661	1.061	1.826	1.641	1.652	1.176	1.084
	3	10049336	45.204	47.142	105.816	1.030	1.695	1.550	5.513	4.821	4.858
html	1	1736480	0.866	4.754	8.177	1.098	1.902	1.605	0.311	0.235	0.206
	2	391824	0.099	0.588	0.600	1.835	3.847	3.077	0.046	0.029	0.030
	3	642216	0.182	1.332	1.391	1.455	2.691	2.168	0.230	0.229	0.203

Comparing the compression ratios, some files see improvements from the bitwise LZ77, namely the text based files (txt and html). Both image type files saw reduced compression ratios. However, the compression and decompression times are also significantly reduced, as expected, since there are far less bytes in the file than bits, and we're iterating through each byte instead of bit.