Lab 5

Bounds

1. This is not a reasonable bound to use because 10^{26} is a lot. We would need that many buckets, which wastes a lot of memory and causes unneccessarily slow computations, just because the numbers are big. Our hashes would need to be 64 bit integers when we only really need 10 bits to accomplish our task.
2. A tighter bound would be $N \leq 1023$. This gives plenty of room for more names in the future, since we only have 574, but doesn't add any more bits than necessary.
3. $B = 1023/0.75 = 1364$

Collisions With My Name

Here are the people with the same hashes as me with $B = 100$, for the unicode hash:

Student Name	Grade	Unicode Hash (B=inf)	Python Hash (B=inf)
75	Chen, Ethan	6	954
143	Dye, Reid	12	754
246	Kani, Adrian	7	1054
257	Kelly, Tristyn	12	1354

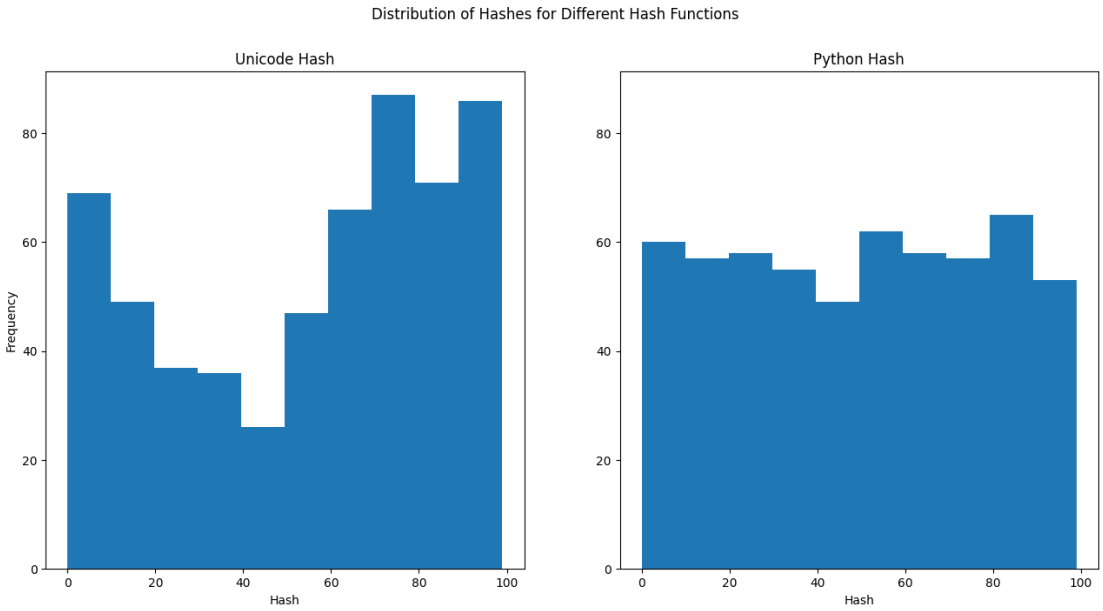
And here's the same thing, for the python hash:

Row	Student Name	Grade	Unicode Hash (B=inf)	Python Hash (B=inf)
20	Avant, Skyler	12	1216	2439014664632337231
38	Bisgaard, Kristian	8	1710	-8996295914056139369
55	Brooks, Kaiya	12	1195	675325292927907131
143	Dye, Reid	12	754	-2211237902951583569
221	Huh, Allison	10	1091	-6231730602966111969
258	Kent, Kaedan	8	1058	2157637754058783331
277	Kwauk, Elena	10	1076	4149948076530565231
307	Lin, Erica	9	851	-8167474085485940869
448	Shah, Rikhav	9	1077	-1651898499280626569

(this is much longer than the unicode hash, but upon further investigation, it seems that this was somewhat of a fluke - it's normally somewhere around 4-8 collisions.

Histograms

Here are the histograms of the hashes, for $B = 100$



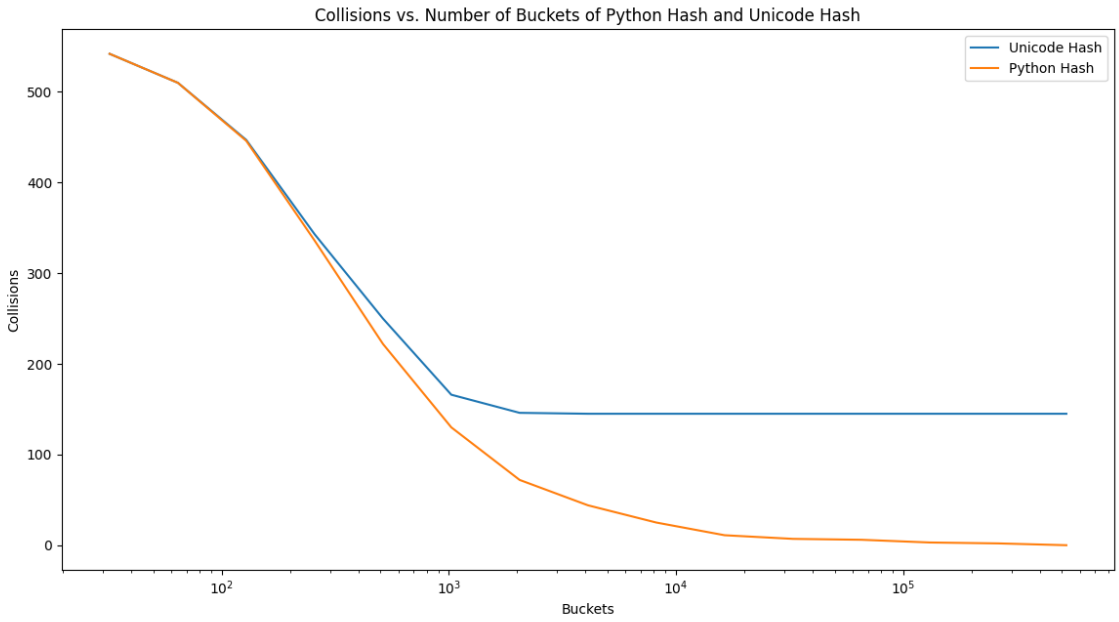
The left histogram shows that the hashes are hardly distributed evenly. Unicode hashes tend to be concentrated around the extremes, with the least hashes between 40 and 50.
This is probably caused by the mod we're applying. The hashes will tend to be centered around the same number, since all the names are approximately the same length, and all letters have similar unicode values. The specific value we chose for the number of buckets probably chopped the distribution in half, creating this weird looking profile.
The Python hash is far more uniform, which makes sense because it's not based around any clearly ordered, string-related thing. I believe it was designed to be maximally dependent on every bit of the input, making a very uniform distribution.

Plots of Collisions vs. Buckets

Here are the collision counts for a couple values of B:

Algorithm	B	Collisions
Unicode	100	475
Unicode	1364	192
Python	100	475
Python	1364	104

And here's a graph of the number of collisions at different numbers of buckets. Measurements were taken at $B = 2^5, 2^6, 2^7, \cdots, 2^{19}$.



For the Unicode hash, it looks like increasing the bucket size helps reduce collisions a lot, but only up to around ~1300, which seems to be exactly the number I predicted as the optimal number of buckets! After that, it doesn't decrease any more. It seems that the Unicode hash function just sucks, because the minimum number of collisions is 145. It doesn't decrease when we add more buckets after that because the mod doesn't do anything - the output of our hash function just doesn't get that big to begin with.

Python's `hash()` is a completely different story. While the two have very similar numbers of collisions for very small numbers of buckets, Python's hash function doesn't have the high lower bound of 145, and just continues to have less and less collisions as we add more buckets. This is likely due to the fact that its range is just higher, and it uses all parts of that range approximately uniformly.