DS210 HW2
Name: Yebin Song
BUid: U76395343
Collaborators: None

1. Report: What are the computation times for different *k* for each of the options? How do the times to compute Fibonacci numbers compare for large *k* between these two options? Are they roughly the same or are they very different? If they are different, what is the multiplicative difference?

   Cargo run absolutely took more time compared to using cargo –release. The time w/o release was approx. 2.4 times the time w release

k	time1	time2	fib	time1/time2
1	0.000000265	1.11E-07	1	2.387387387
2	0.000000154	8.30E-08	1	1.855421687
3	0.000000124	8.30E-08	2	1.493975904
4	0.000000137	9.20E-08	3	1.489130435
5	0.000000195	7.90E-08	5	2.46835443
6	0.00000021	9.60E-08	8	2.1875
7	0.000000284	1.47E-07	13	1.931972789
8	0.000000396	2.26E-07	21	1.752212389
9	0.000000599	2.58E-07	34	2.321705426
10	0.000000931	4.15E-07	55	2.243373494
11	0.000001427	5.63E-07	89	2.534635879
12	0.000002285	8.77E-07	144	2.605473204
13	0.000003438	1.40E-06	233	2.457469621
14	0.000005562	2.26E-06	377	2.459973463
15	0.000008843	3.64E-06	610	2.428728371
16	0.000014197	5.84E-06	987	2.429329227
17	0.000022986	9.43E-06	1597	2.437281306
18	0.000037201	1.53E-05	2584	2.437971033
19	0.000060017	2.47E-05	4181	2.43368071
20	0.000097076	3.98E-05	6765	2.437380737
21	0.000156927	6.44E-05	10946	2.435279877
22	0.000253761	0.00010418	17711	2.435793818

23	0.000416615	0.05143252	28657	0.008100225
24	0.000664022	0.00036257	46368	1.831421069
25	0.001082963	0.00046326	75025	2.337690119
26	0.001741659	0.00072424	121393	2.404802538
27	0.002826384	0.0011655	196418	2.425039897
28	0.004570956	0.00189358	317811	2.413926658
29	0.007408126	0.00304797	514229	2.430515442
30	0.011977778	0.00492606	832040	2.431514292
31	0.0193753	0.00796435	1346269	2.432754374
32	0.041533806	0.01288043	2178309	3.224566476
33	0.050640118	0.0209231	3524578	2.420297204
34	0.084260417	0.06004866	5702887	1.40320238
35	0.138122461	0.05467831	9227465	2.526092219
36	0.218234205	0.09266354	14930352	2.355124944
37	0.381165279	0.1434275	24157817	2.65754679
38	0.614152445	0.23172975	39088169	2.650296124
39	0.923910796	0.40050601	63245986	2.306858744
40	1.525992081	0.64671691	102334155	2.359598234
41	2.468554523	1.0557069	165580141	2.338295334
42	4.096549344	1.62990167	267914296	2.513372071
43	6.472697672	2.64891869	433494437	2.443524487
44	10.43644812	4.36413918	701408733	2.391410468
45	16.93890994	6.90367131	1134903170	2.453608983
46	27.42601304	11.3321511	1836311903	2.420194781
47	44.12178338	18.2847774	2971215073	2.413033663
48	71.65408178	29.3379259	4807526976	2.442370398
49	116.0191603	47.534388	7778742049	2.440741646

2. Report: Now conduct the following experiment. Replace the array entry type with u8 and adjust any other types accordingly so your program still compiles. Try running the modified code with both cargo run and cargo run --release. Are there any differences in the behavior of the program? If so, what are they?

When running with cargo run, it said “thread 'main' panicked at src/main.rs:17:16: attempt to add with overflow note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace” and didn’t output anything.
When running with cargo run –release, it overflowed when the number went over 256 (∵2^8 == 256), but didn’t give me any warnings or error messages and let the code burn.

3. Report: Explain why the situation described above is not happening, i.e., why the range of integers you use is sufficiently large. This kind of problem is known as *integer overflow*, i.e., you want to explain why integer overflow is not a problem in your code.

   I used u32 for sum and cubed (I wasn’t sure about how inefficient I was instructed to code, so I did a nested for loop to cube and sum everything). For u8, the largest number it can hold is 255 (∵2^8 == 256). 255^3 = 16581375. 2^16=65536, smaller than 255^3, thus cubed needed to be u32. Now, 2^32=4294967296, and sum(i**3, 0<i<256) is 1065369600, within 2^32. Thus, u32 is enough to store enough information for this ta sk.

   Another (useless) explanation would be: $\sum_{1}^{255} i^3 = (\frac{255(256)}{2})^2 = 1065369600$, and this is greater than 2^16, smaller than 2^32, so it works.

4. Took me about 3 hours. (I was doing a bunch of random things)