Assignment 1

Part 1 A:

Algorithms:

Brute Force:

The Brute Force algorithm exhaustively checks all possible solutions and has an average runtime of O(m*n), where m is the size of the text and n is the size of the pattern. It is a simple and straightforward approach but can be computationally expensive for large problem sizes due to its exponential time complexity.

Sunday:

The Sunday pattern matching algorithm is an efficient string searching algorithm that finds occurrences of a pattern within a larger text. It uses the heuristic based on the rightmost character of the current window in the text to determine how many positions to shift the pattern.

The algorithm compares the pattern to the text character by character, starting from the leftmost position. If a mismatch occurs, it shifts the pattern to the right by a calculated amount based on the rightmost character of the current window. This allows for skipping ahead in the text, potentially reducing the number of comparisons required.

The average runtime complexity of the Sunday algorithm is O(m/n), where m is the size of the text and n is the size of the pattern. The Sunday algorithm is particularly efficient for patterns with large alphabets or in cases where the pattern has few distinct characters.

KMP:

The Knuth-Morris-Pratt (KMP) algorithm is a string matching algorithm that efficiently finds occurrences of a pattern within a larger text. It utilizes a failure function to determine the maximum length of a proper suffix that is also a prefix for each prefix of the pattern. This information is used to avoid unnecessary character comparisons during the matching process.

The runtime complexity of the KMP algorithm is O(m + n), where m is the length of the text and n is the length of the pattern. The preprocessing step to construct the failure function takes O(n) time, and the matching process takes O(m) time. This makes the KMP algorithm efficient for large texts or patterns.

Rabin-Karp:

The Rabin-Karp algorithm efficiently finds occurrences of a pattern within a larger text using hashing. It compares hash values of the pattern and text substrings and performs character-by-character comparisons for potential matches. It has a runtime complexity of O(m + n) in most cases, where m is the text length and n is the pattern length. It's suitable for large texts and multiple pattern matching.

GusfieldZ:

The Gusfield Z algorithm is a linear time string matching algorithm that efficiently finds all occurrences of a pattern within a larger text. It constructs a Zarray that stores the lengths of the longest common prefixes between the pattern and each prefix of the text. The runtime complexity of the Gusfield Z algorithm is O(m + n), where m is the length of the text and n is the length of the pattern. It is a powerful algorithm for pattern matching in large texts.

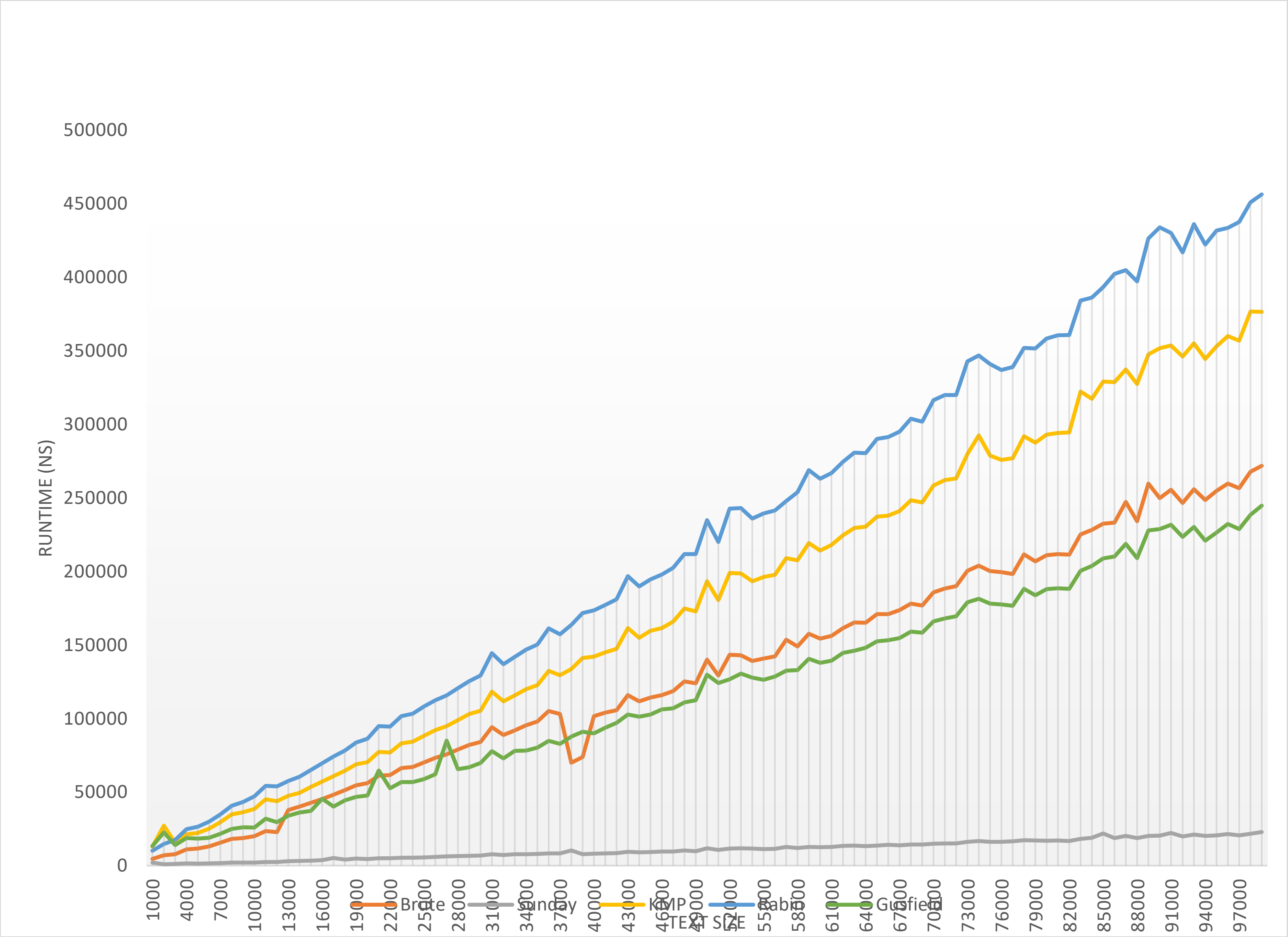
Output:

Size:	Brute	Sunday	KMP	Rabin	Gusfield
1000	18447	4471	32288	17641	32211
2000	4561	2069	12876	10127	13335
3000	7004	908	27131	14769	22573
4000	7823	1116	15451	17370	13999
5000	11035	1534	21339	24708	18784
6000	11524	1389	22147	26335	18300
7000	13024	1518	25054	29980	19004
8000	15675	1691	29581	34858	21600
9000	18234	2045	34870	40777	24885
10000	18763	2108	36228	43248	26069
11000	20072	2140	38511	47195	25848
12000	23458	2476	45078	54184	31906
13000	22774	2382	43783	53935	29602
14000	37737	3013	47491	57501	34011
15000	40112	3160	49512	60516	36201
16000	42641	3457	53577	65108	37314
17000	45316	3797	57094	69534	45460
18000	48276	5277	60780	74241	40112
19000	51332	4146	64544	78273	44431
20000	54596	4856	68817	83675	46678
21000	56086	4433	70410	86239	47702
22000	61274	5076	77309	94782	64755
23000	61506	4991	77008	94546	52514
24000	66248	5417	83230	101675	56727
25000	67128	5460	84252	103313	56798
26000	70181	5633	88205	108206	58736
27000	73211	5973	92078	112418	62172
28000	75580	6206	94883	115707	85063
29000	78855	6454	98957	120578	65660
30000	82115	6735	103103	125416	66848
31000	84053	6813	105267	129342	69727
32000	94140	7719	118236	144437	77798
33000	88792	7209	111616	136868	72991
34000	92018	7770	115773	141935	78072
35000	95481	7766	119920	146893	78229
36000	97896	7931	122603	150256	80232
37000	105177	8374	132333	161253	84821
38000	103137	8363	129461	157310	82821
39000	70044	10319	133599	163608	87794
40000	73901	7816	141241	171696	90958

41000	101711	8181	142055	173466	89898
42000	104066	8388	145020	177206	93702
43000	105680	8581	147464	181089	97091
44000	115905	9390	161508	196704	102670
45000	111754	8976	154956	189852	101357
46000	114221	9137	159669	194615	102729
47000	115875	9520	161538	197864	106160
48000	118590	9650	165836	202502	106911
49000	125278	10294	174847	211714	110975
50000	124010	9796	172905	211718	112499
51000	140084	11713	193265	234848	129893
52000	129266	10685	180477	219949	124163
53000	143385	11645	199082	242797	126757
54000	142962	11738	198713	243033	130540
55000	139124	11535	193331	235869	127877
56000	140813	11262	196246	239454	126316
57000	142194	11382	197723	241495	128619
58000	153592	12763	209070	247901	132603
59000	149022	11890	207565	253802	132923
60000	157724	12616	219287	268911	140555
61000	154316	12534	214208	263070	137921
62000	156210	12732	218002	266887	139266
63000	161462	13421	224701	274478	144710
64000	165327	13568	229494	280816	146034
65000	165060	13333	230537	280463	148072
66000	171054	13628	237231	290198	152571
67000	171010	14191	237986	291353	153247
68000	173798	13751	241171	295016	154720
69000	178168	14276	248316	303827	159153
70000	176787	14259	246865	301815	158360
71000	185852	14866	258463	316485	166130
72000	188316	14995	262085	319969	168007
73000	190010	15010	263206	319949	169446
74000	200497	16237	279940	342850	179127
75000	203901	16723	292510	346880	181473
76000	200325	16234	278794	340956	178094
77000	199495	16195	275940	337025	177503
78000	198216	16584	276997	338964	176574
79000	211543	17222	291978	352053	188277
80000	206805	17145	287622	351634	183764
81000	211124	16977	293090	358427	188044
82000	211709	17003	294111	360490	188544
83000	211524	16687	294464	360832	188247
84000	225095	18130	322399	384107	200390
85000	228356	18851	317336	386244	203808
86000	232488	21835	329046	393408	208872
87000	233139	18680	328800	402262	210154
88000	247326	20143	337257	404852	218844

89000	234135	18694	327454	397239	209021
90000	259774	20142	347560	426533	227929
91000	249949	20313	351726	433915	228903
92000	255454	22272	353640	430180	231761
93000	246548	19778	346210	416974	223467
94000	255914	21147	355160	436132	230203
95000	248574	20236	344489	422286	220947
96000	254776	20538	353112	431746	226423
97000	259731	21506	360080	433509	232323
98000	256626	20651	356933	437573	228827
99000	267873	21685	376740	451038	238522
100000	271827	22844	376510	456346	244729

Chart:



Findings:

In this example I have checked each algorithm on text sizes from 1000 to 100000 with 1000 steps and tested each algorithm 1000 times and got average result as runtime to get more stable results.

In the end, Sunday algorithm outshone any other algorithm in both short and long text sizes.

Brute Force algorithm was, as expected, fast for short text sizes, but gradually gave second place to Gusfield as text size became bigger.

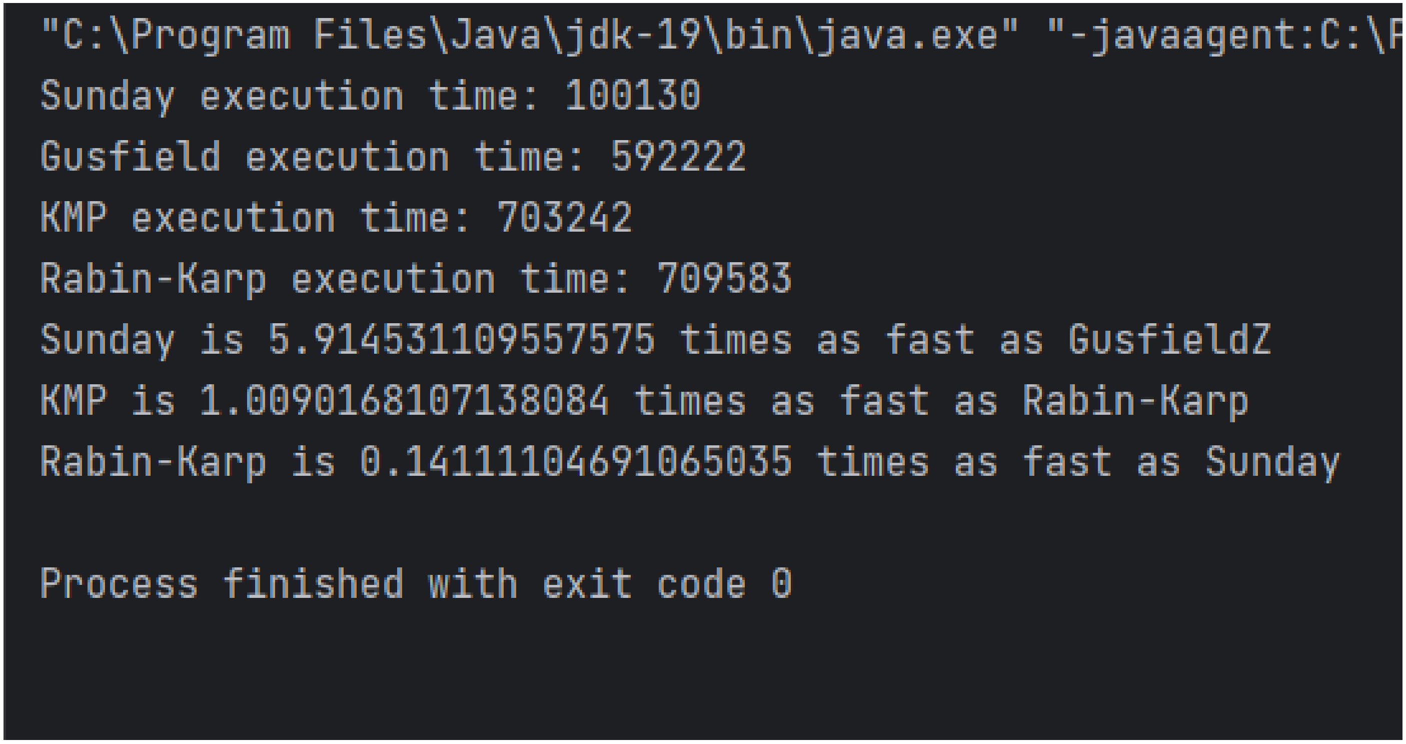
KMP and Rabin-Karp algorithms were slowest ones, taking 4th and 5th places respectively.

Almost all algorithms’ runtime grow linearly with the text size, but Sunday algorithm has the lowest rate of growth, making it the best Pattern Matching Algorithm.

Part 1 B:

In the next example, I took “text.txt” file, which is 164 kB, and copied it’s content to String text object. Tested Sunday, Gusfield, Rabin-Karp and KMP algorithms 1000 times each and got average result.

Output:



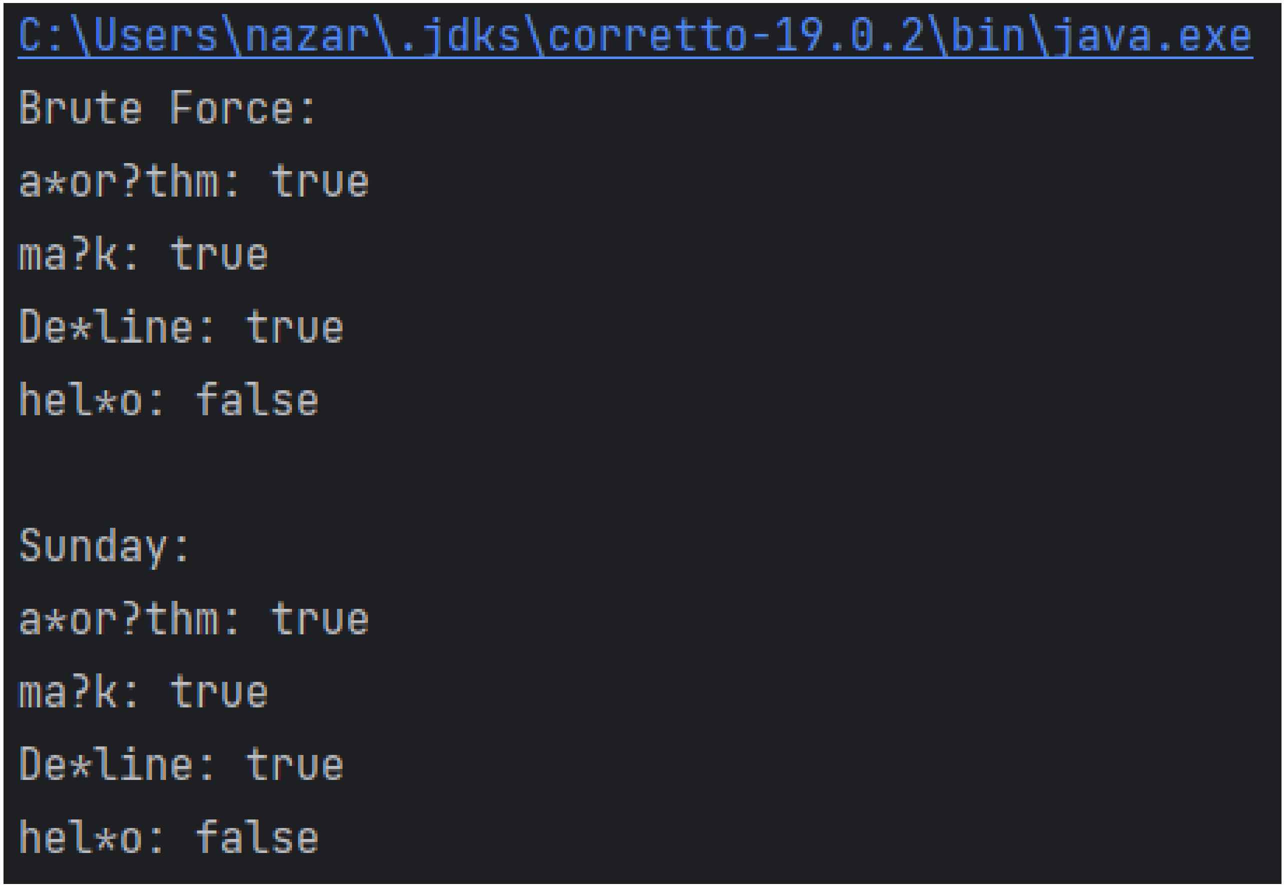
```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\
Sunday execution time: 100130
Gusfield execution time: 592222
KMP execution time: 703242
Rabin-Karp execution time: 709583
Sunday is 5.914531109557575 times as fast as GusfieldZ
KMP is 1.0090168107138084 times as fast as Rabin-Karp
Rabin-Karp is 0.14111104691065035 times as fast as Sunday

Process finished with exit code 0
```

Part 2:

This example contains implementations of Brute Force and Sunday algorithms, but instead of finding all occurrences of pattern, they both return true value as soon as they find first matching substring, finishing the execution.

This implementations of Brute Force and Sunday algorithms extend to support ‘*’ and ‘?’ wildcards, it compares characters in text and pattern. If pattern contains one of wildcards, it will automatically be equal with text character.

Output:



```
C:\Users\nazar\.jdks\corretto-19.0.2\bin\java.exe
Brute Force:
a*or?thm: true
ma?k: true
De*line: true
hel*o: false

Sunday:
a*or?thm: true
ma?k: true
De*line: true
hel*o: false
```