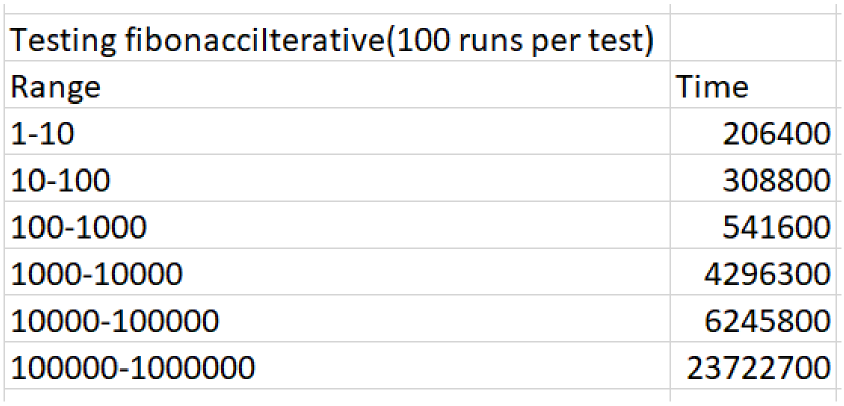
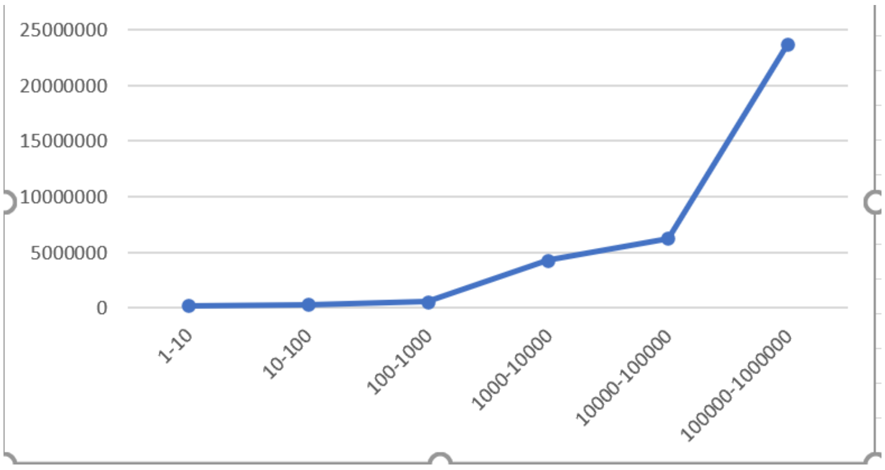
Analysis

Fibonacci



Testing fibonacciIterative(100 runs per test)	
Range	Time
1-10	206400
10-100	308800
100-1000	541600
1000-10000	4296300
10000-100000	6245800
100000-1000000	23722700

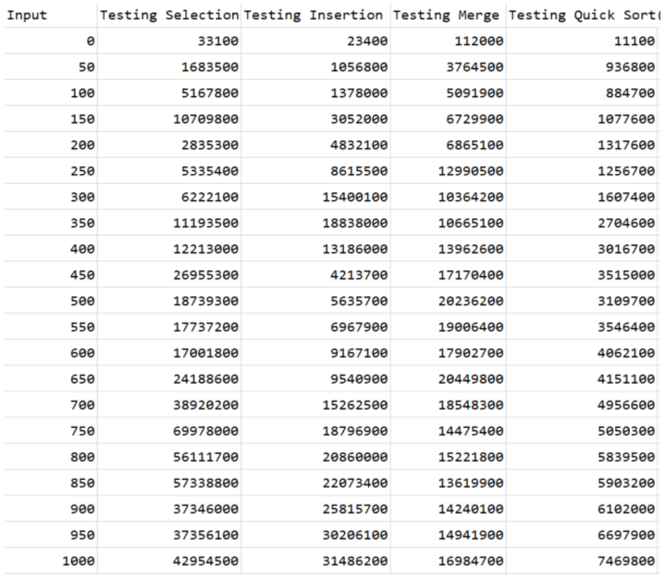


Here we analyse the performance of the Fibonacci algorithm.

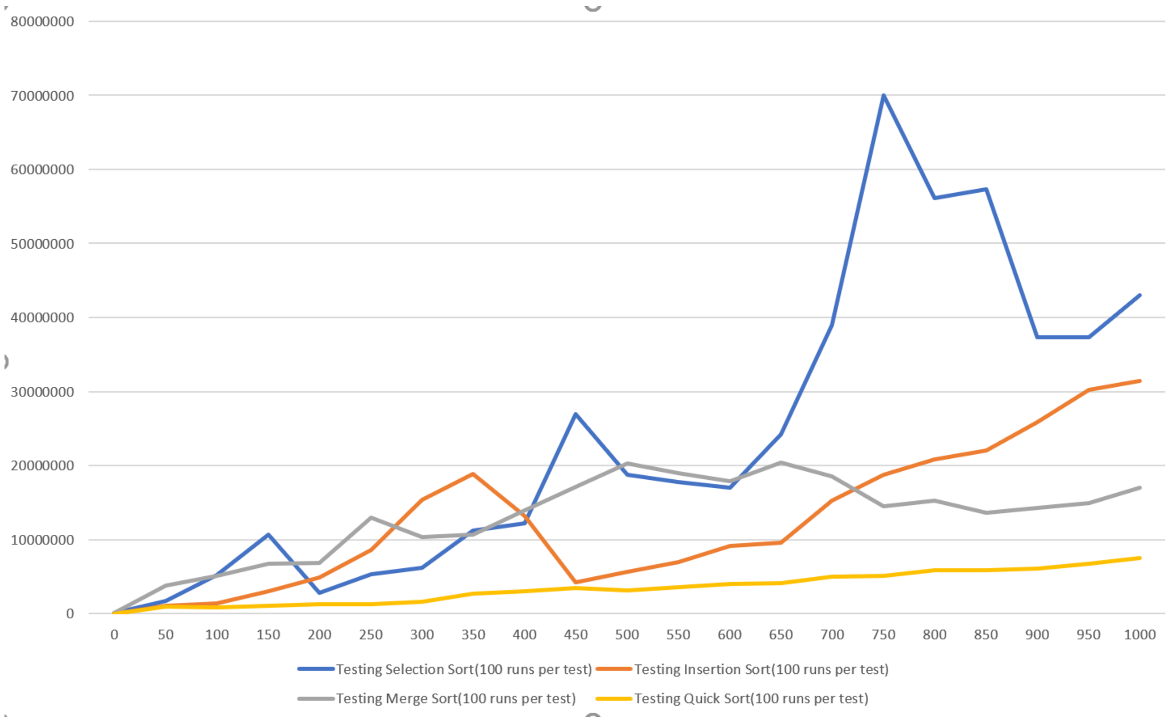
I approached this by running each range 100 times selecting a random number within the range.

The result is expected the with the Fibonacci growth being O(2^n).

Sorting



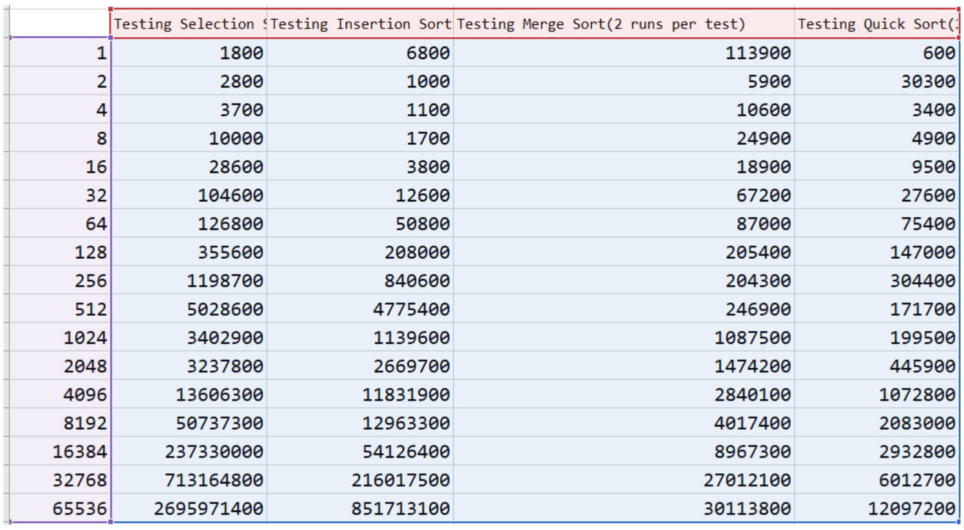
Input	Testing Selection	Testing Insertion	Testing Merge	Testing Quick Sort(
0	33100	23400	112000	11100
50	1683500	1056800	3764500	936800
100	5167800	1378000	5091900	884700
150	10709800	3052000	6729900	1077600
200	2835300	4832100	6865100	1317600
250	5335400	8615500	12990500	1256700
300	6222100	15400100	10364200	1607400
350	11193500	18838000	10665100	2704600
400	12213000	13186000	13962600	3016700
450	26955300	4213700	17170400	3515000
500	18739300	5635700	20236200	3109700
550	17737200	6967900	19006400	3546400
600	17001800	9167100	17902700	4062100
650	24188600	9540900	20449800	4151100
700	38920200	15262500	18548300	4956600
750	69978000	18796900	14475400	5050300
800	56111700	20860000	15221800	5839500
850	57338800	22073400	13619900	5903200
900	37346000	25815700	14240100	6102000
950	37356100	30206100	14941900	6697900
1000	42954500	31486200	16984700	7469800



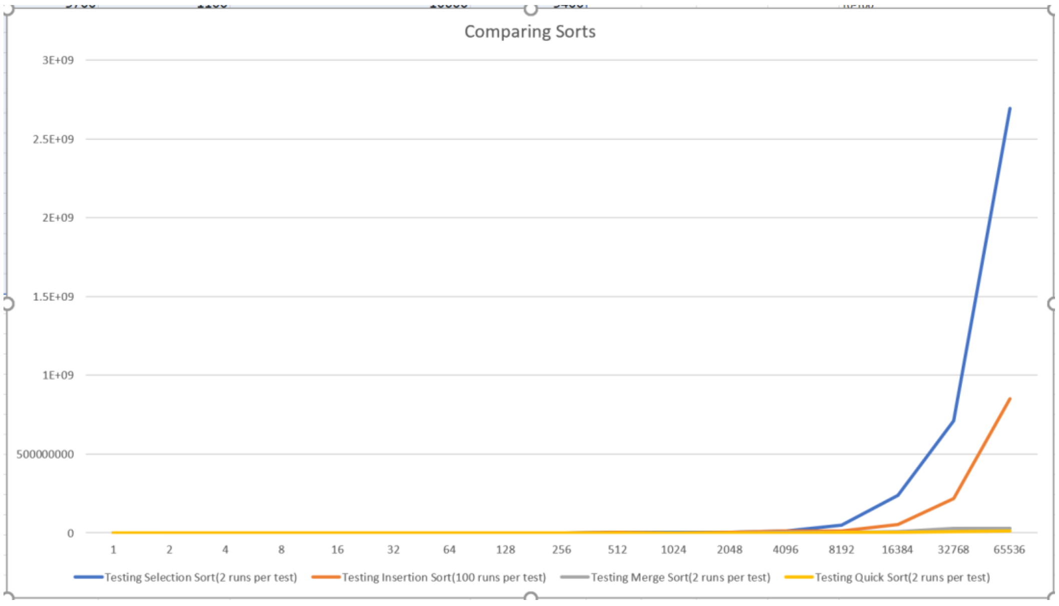
Next, I analysed the sorting algorithms, I done this by running each input range 100 times and increasing the input size by 50.

we can see spikes in our graph where background tasks clearly interrupted the analysis.

For this we get a rough idea of how each sort preforms but its not as clear as I was hoping so I increased the input size and lowers the runs.



	Testing Selection	Testing Insertion Sort	Testing Merge Sort(2 runs per test)	Testing Quick Sort(
1	1800	6800	113900	600
2	2800	1000	5900	30300
4	3700	1100	10600	3400
8	10000	1700	24900	4900
16	28600	3800	18900	9500
32	104600	12600	67200	27600
64	126800	50800	87000	75400
128	355600	208000	205400	147000
256	1198700	840600	204300	304400
512	5028600	4775400	246900	171700
1024	3402900	1139600	1087500	199500
2048	3237800	2669700	1474200	445900
4096	13606300	11831900	2840100	1072800
8192	50737300	12963300	4017400	2083000
16384	237330000	54126400	8967300	2932800
32768	713164800	216017500	27012100	6012700
65536	2695971400	851713100	30113800	12097200



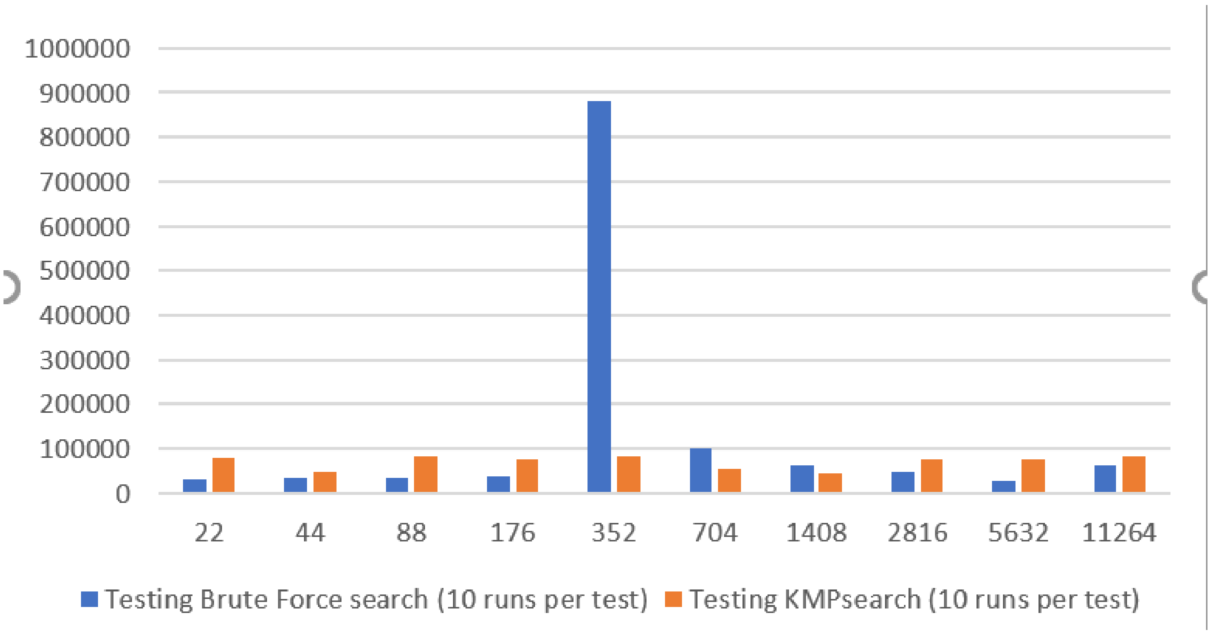
Now we can clearly see quick sort preforms the best and selection sort the worst, we see how increasing the input size greatly helps.

	Best	Average	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)
Merge Sort	Ω(n log(n))	Θ(n log(n))	O(n log(n))
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)

Comparing our results to the expected time complexities we can see our graph follows a similar order.

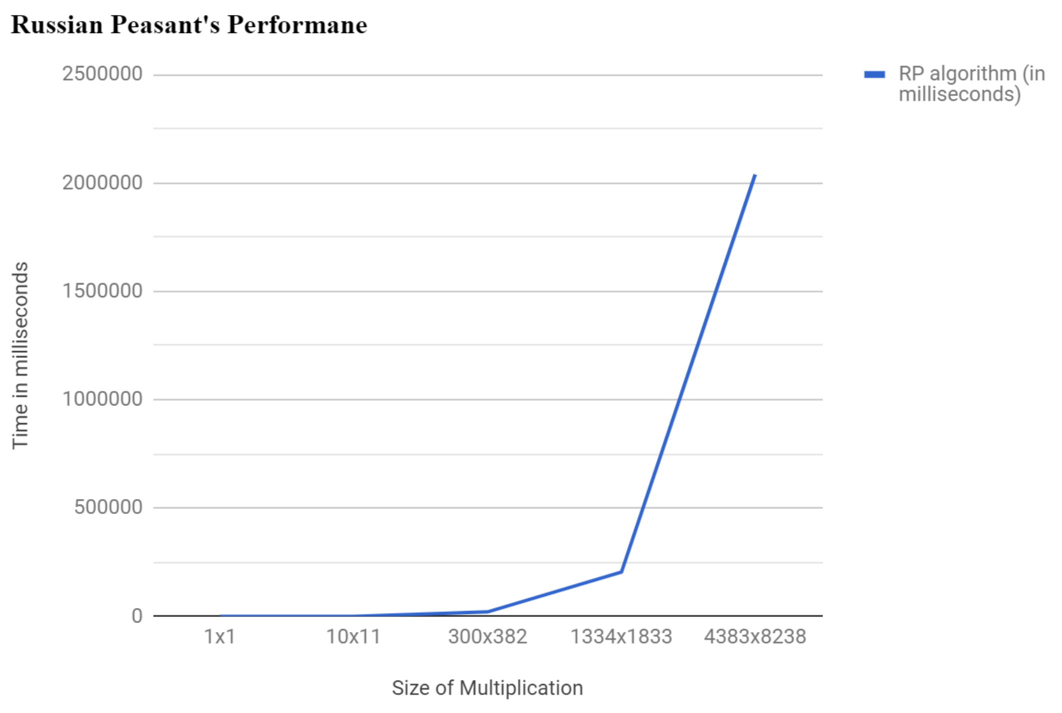
Searches

Leght	Testing Brute Force search (10 runs per test)	Testing KMPsearch (10 runs per test)
22	30500	78400
44	35300	48800
88	32700	81600
176	36200	76200
352	882100	82800
704	100700	53600
1408	60800	44500
2816	48600	75600
5632	26800	75900
11264	60900	81600



Here again we see background tasks disrupting my analysis. Here the brute force string search outperformed the KMP search. This test was done by running each length 10 times and doubling the length of the string each iteration. The size of the string being found was not changed.

Russian Algorithm



Log of Test and Performance.

```
Russian Multiplication Working
Fibonacci is working
Selection Sort Working
Insertion Sort Working
Merge Sort Working
Quick Sort Working
Found brute force search working
Not Found brute force search working
Found KMP search working
Not Found KMP search working
Present in trie, working
Not present in trie, working

Testing RussianAlgorithm(1 runs per test)
Range   Time
1-10         3700
10-100       2700
100-1000     3800
1000-10000   2100
10000-100000 2500
100000-1000000      2300

Testing fibonacciIterative(100 runs per test)
Range        Time
1-10         21300
10-100       290000
100-1000     2270300
1000-10000   3313100
10000-100000 5799000
100000-1000000      27884300
```

```
Testing Selection Sort(100 runs per test)
Input  Time
0      30200
50     3079000
100    8550000
150    4443800
200    4002000
250    5516000
300    13097700
350    34444000
400    79539900
450    18226200
500    16141700
550    17819200
600    20737400
650    18570800
700    29009700
750    26780100
800    32341000
850    28448100
900    31722800
950    39005400
1000   48487900
```

```
Testing Insertion Sort(100 runs per test)
Input  Time
0      12300
50     793100
100    1964200
150    4631900
200    8744200
250    11343200
300    14288200
350    16526300
400    16979200
450    9933800
500    3665300
550    4085900
600    5315500
650    5451700
700    6095700
750    6576400
800    7536900
850    8135700
900    9460400
950    14526200
1000   23596300
```

```
Testing Merge Sort(100 runs per test)
Input  Time
0      167900
50     2233200
100    2661100
150    4507400
200    6022100
250    11171800
```

```
300    8373000
350    11359600
400    11796300
450    11776600
500    9334000
550    8352200
600    9484000
650    12498700
700    13506600
750    12171600
800    16815800
850    14314700
900    13492800
950    14643500
1000   16334700
```

```
Testing Quick Sort(100 runs per test)
Input  Time
0      17500
50     974200
100    1177900
150    1855100
200    1464600
250    1159300
300    1399900
350    1812400
400    1955100
450    2233200
500    2915400
550    2946000
600    3279300
650    3560200
700    4627900
750    4550000
800    4528900
850    6593900
900    5088900
950    5865200
1000   5796700
```

```
Testing Advanced Quick Sort(100 runs per test)
Input  Time
0      18700
50     814900
100    1607700
150    6184800
200    10559400
250    17123700
```

```
300    18764900
350    27458300
400    36697300
450    47481700
500    59205200
550    71372400
600    83248400
650    97779600
700    116678300
750    136185100
800    143821100
850    165816800
900    186142400
950    201821200
1000   228678600
```

```
Testing Brute Force search (10 runs per test)
Leght  Time
22     64800
44     99600
88     88300
176    65300
352    121500
704    64400
1408   62500
2816   68800
5632   62000
11264  36600

Testing KMPsearch (10 runs per test)
Leght  Time
22     50700
44     45000
88     44700
176    69300
352    51800
704    19500
1408   18600
2816   81400
5632   31300
11264  40500
```