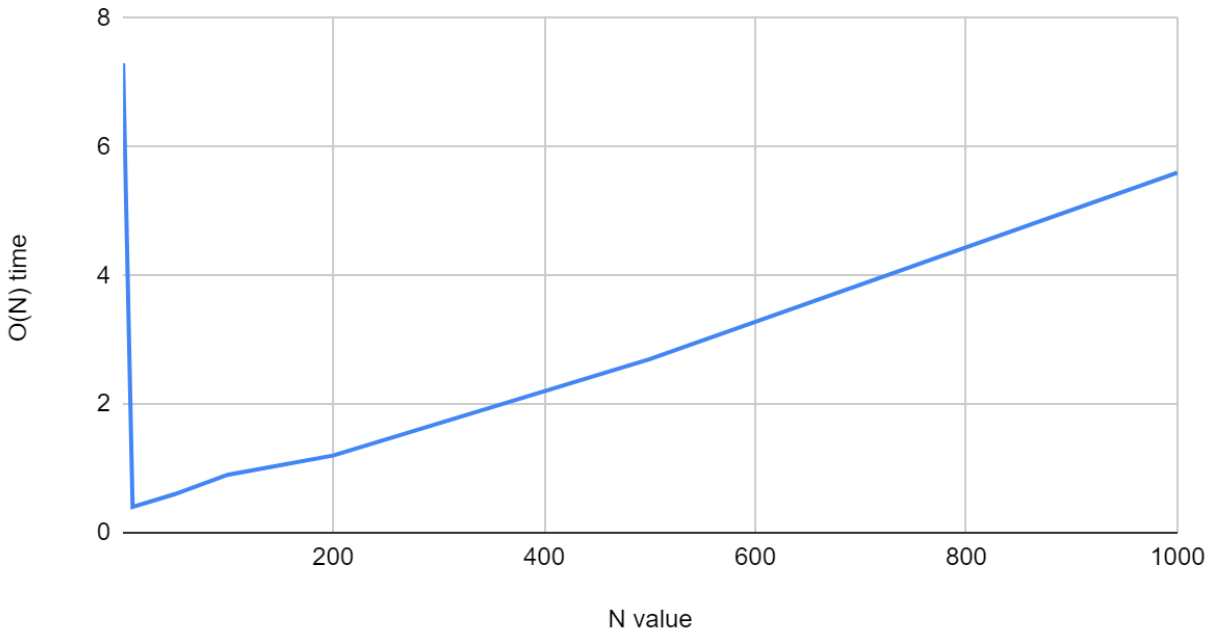
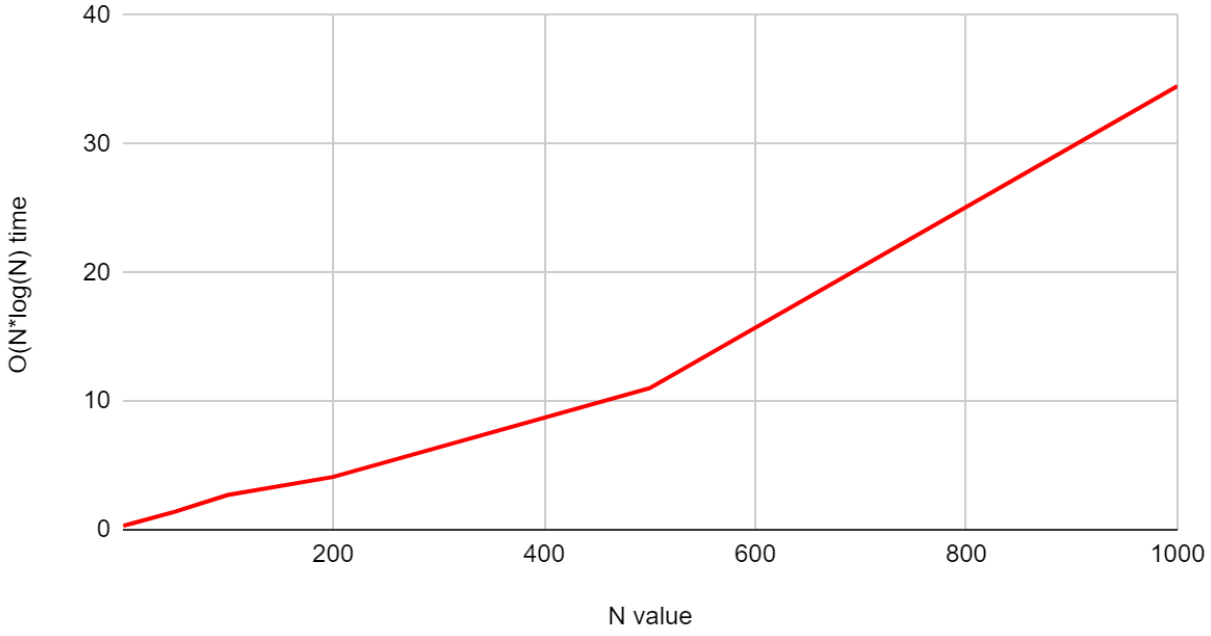
From times.txt

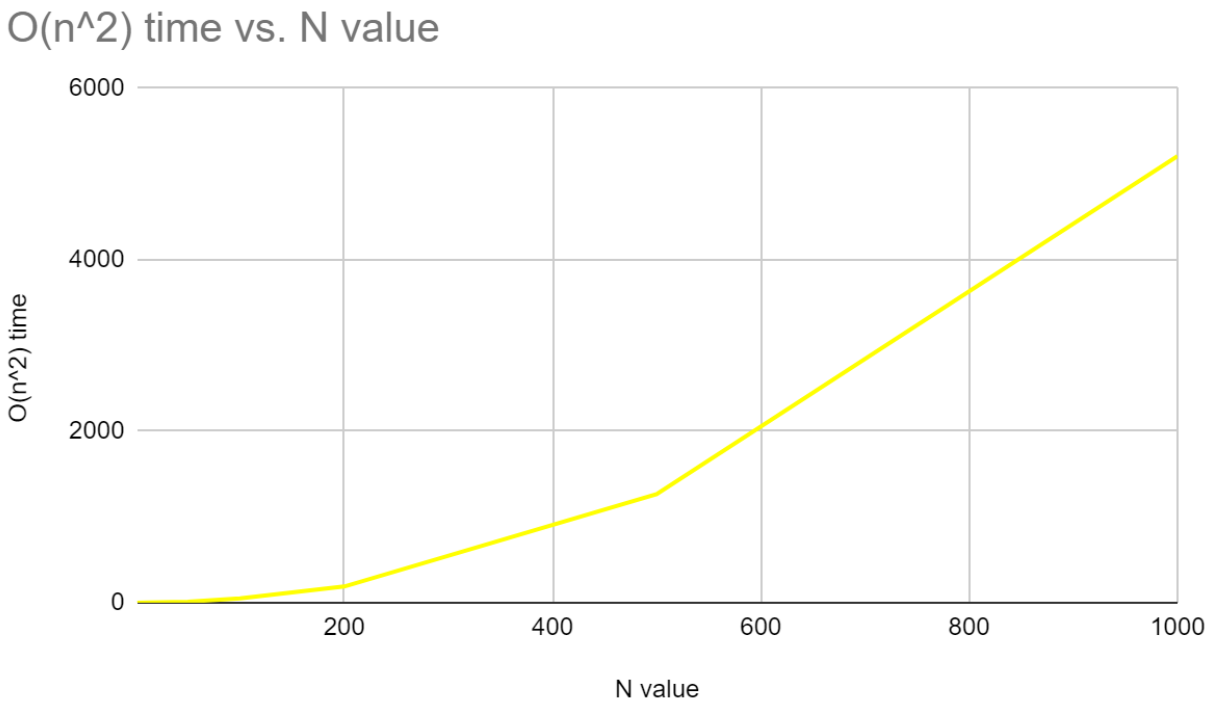
N value lowest, O(N) time, O(N*log(N)) time, O(N^2) time, O(N^3) time
1, 7.3, 0.3, 0.2, 0.3
10, 0.4, 0.5, 0.7, 3.6
50, 0.6, 1.4, 10.4, 533.1
100, 0.9, 2.7, 51.6, 4893.8
200, 1.2, 4.1, 188.6, 38962.1
500, 2.7, 11, 1265.9, 634802
1000, 5.6, 34.5, 5208.1, 4.87768e+06
N value highest, O(N) time, O(N*log(N)) time, O(N^2) time, O(N^3) time

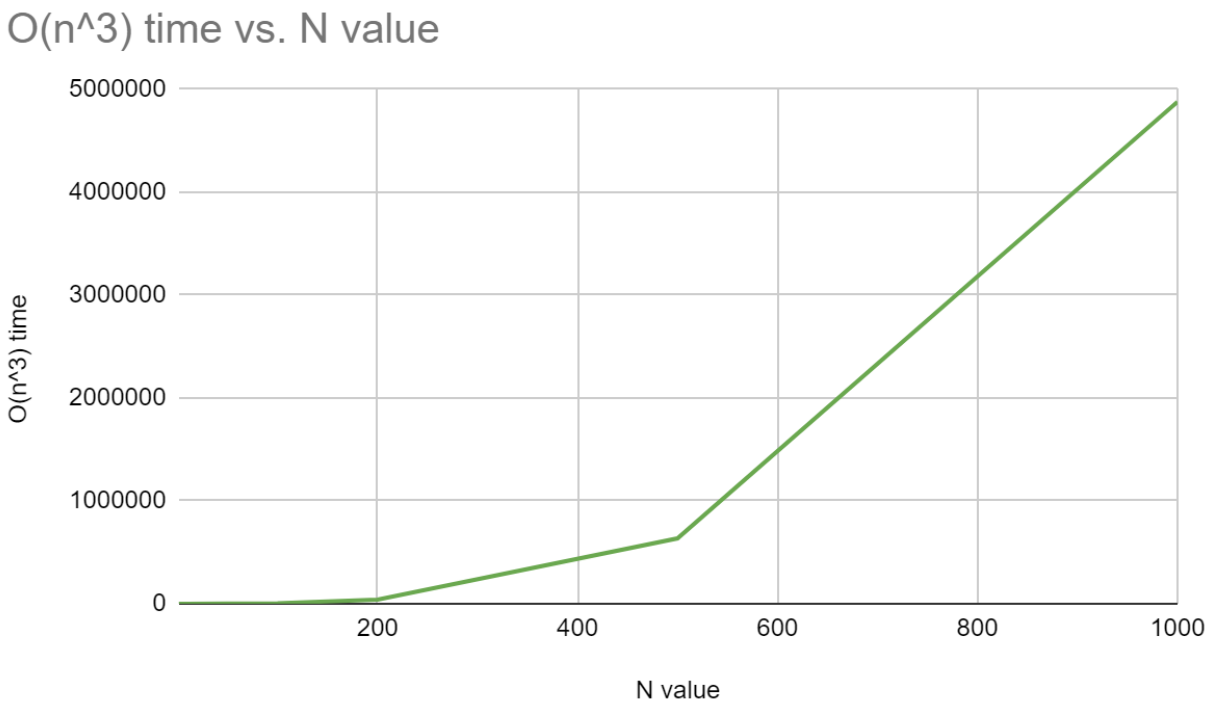
O(N) time vs. N value

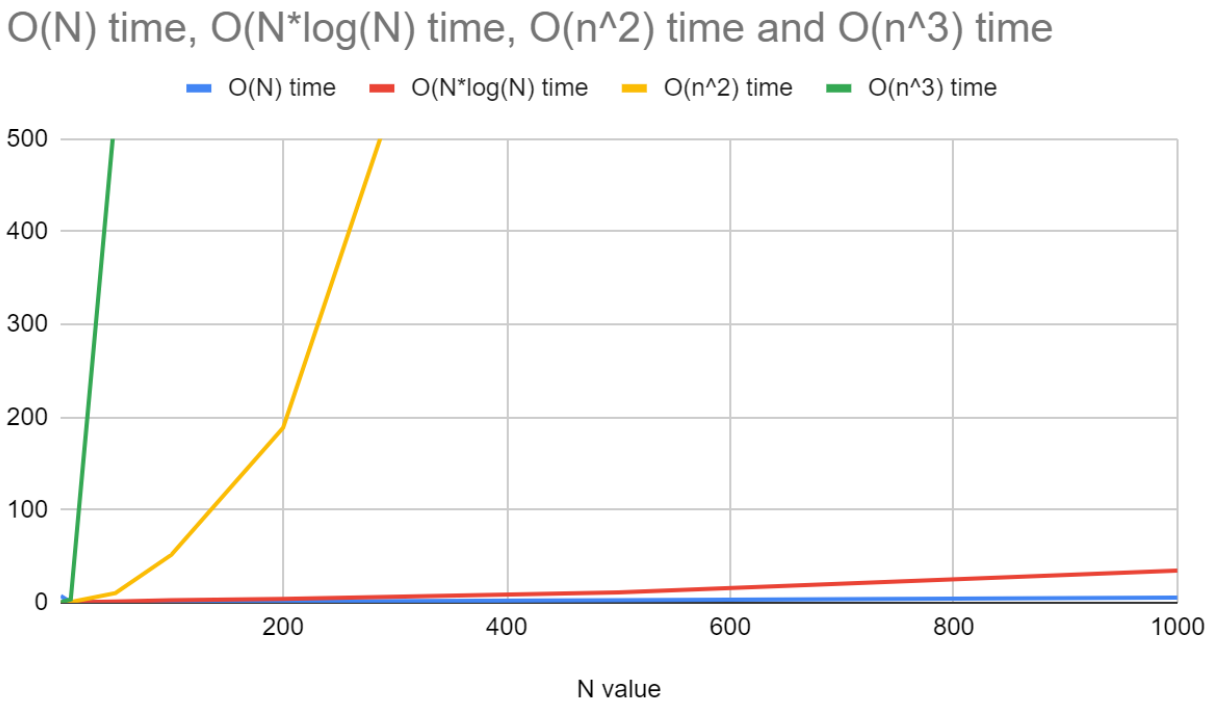


O(N*log(N) time vs. N value




O(n^2) time vs. N value
6000
4000
2000
0
O(n^2) time
200
400
600
800
1000
N value


O(n^3) time vs. N value
5000000
4000000
3000000
2000000
1000000
0
O(n^3) time
200
400
600
800
1000
N value



The O(N) function is the fastest because it increases linearly. Plugging 1000 into all of these O(N)s, we get O(1000), O(1000*log(1000)), O(1000^2), O(1000^3). As one can see, the largest value is 1000^3, and that is what grows the fastest. From the graph, we can see that it increases exponentially the fastest. O(N^2) is right behind it, as it is an exponential function as well. O(N*log(N)) increases at a much slower rate, as it is almost an exponential, but the logarithm keeps the value low. O(N) seems to be the best one for sorting, as it is the fastest at computing all the results. The difficulty is designing an algorithm that can do that.