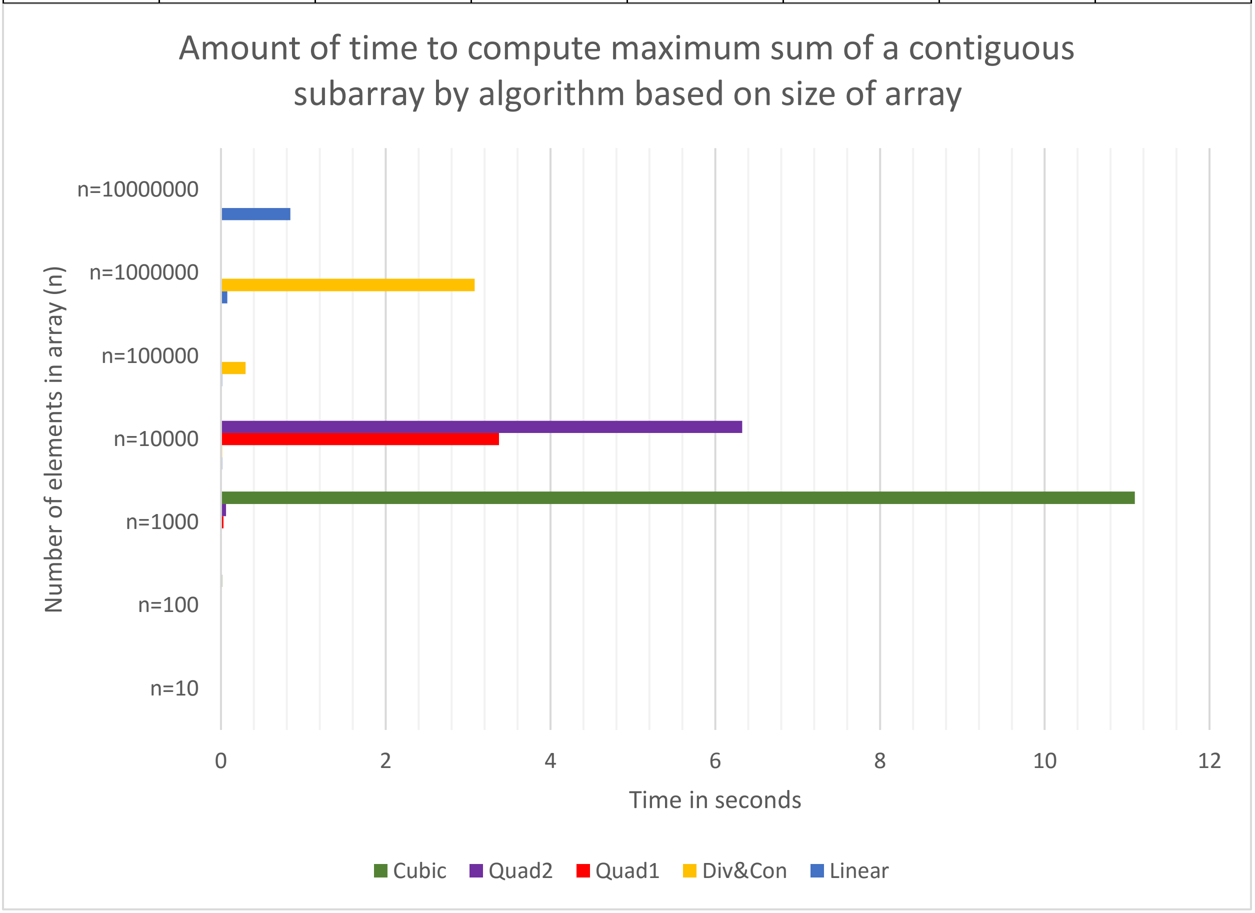
Richard Burke

CSC 440 Programming Assignment

	10	100	1000	10000	100000	1000000	10000000
Linear	<10e-9 s	<10e-9 s	<10e-9 s	0.015625s	0.015625s	0.078125s	0.843750s
Div&Con	<10e-9 s	<10e-9 s	<10e-9 s	0.015625s	0.296875s	3.078125s	N/A
Quad1	<10e-9 s	<10e-9 s	0.031250s	3.375000s	N/A	N/A	N/A
Quad2	<10e-9 s	<10e-9 s	0.062500s	6.328125s	N/A	N/A	N/A
Cubic	<10e-9 s	0.015625s	11.09375s	N/A	N/A	N/A	N/A



The file named, msinfo32.txt, contains every single detail about the machine I used to run these algorithms. This also includes every process running and resources being used which of course play a role in the CPU performance. The command in Windows operating systems, msinfo32, gives this information. An abstract description of the machine:

- Microsoft Windows 10 Home Version 10.0.10586 Build 10586
- Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz, 2592 Mhz, 4 Core(s), 8 Logical Processor(s)
- Installed Physical Memory (RAM): 12.0 GB
- Total Physical Memory: 11.9 GB
    - Available Physical Memory: 6.75 GB
- Total Virtual Memory: 14.2 GB
    - Available Virtual Memory: 7.90 GB
- System Manufacturer: ASUSTeK COMPUTER INC.
    - System Model: GL502VT

• What were the relative performances of each algorithm? Did you get the results you expected? Were the results in accordance with the theory (Did the run time of the linear algorithm increase linearly? What about the quadratic, cubic, and divide and conquer algorithms?). If not, why not?

Relatively speaking, the algorithms that should theoretically perform better than their counterparts, did. The overall behaviors of the results are what I would expect: the implementation roughly reflects the theoretical time complexity. When it comes to examining the exact numbers, the machine outputs the time performance in oddly specific values especially when the given algorithm took around or faster than one one-hundredth of a second. In fact, in the original output, which is saved in the cmdOutputs.txt, the time was sometimes 0.00000000. With that said I think the exact values shouldn't be judged but rather the relative time it took when compared to the other algorithms or just between different sizes of the array. Namely, as the size of the array increased by powers of ten, one can see that the linear algorithm does roughly increase in powers of ten such as when n=1 000 000 compared to n=10 000 000.

• In Python it is possible to implement the five algorithms almost exactly as they are written using the pseudo code in your reader. In particular it is possible to use the max() method to return the maximum of two numbers. How would you get around using this max() function and how does implementing the algorithms without using the max() function alter the run times of your methods? Can you explain the difference, if any?

According to the Python wiki site, the built-in max() function runs in linear time. Since all of the tested algorithms are of linear time-complexity or slower the max() function, theoretically, shouldn't negatively affect the algorithms' performances.

• What are the storage requirements of each algorithm? One of the quadratic algorithms requires extra store; explain which one, why and how much. The divide-and-conquer algorithm also requires extra storage, explain why and how much. Finally, suppose you were reading input from a file, describe how it would be possible for the linear algorithm to use less storage than loading the full data into an array. How would this change in loading the data affect the run time of the algorithm?

The linear algorithm, one of the quadratic algorithms, and the cubic algorithm have a storage of size n (given the array of size n) because they each load the array into memory and use iteration over the array. The second quadratic function, which does the preprocessing, requires extra storage because it creates an array of the same size as the already loaded array, therefore it requires roughly 2*n storage. The divide and conquer algorithm requires extra storage because it needs to call recursively on subarrays of the input array as well as recursive calls for subarrays of other subarrays. Each recursive call requires a copy of a subarray. Given that there are n elements in the original array and the depth of the recursive calls would be log(n) (log base 2 because we're creating subarrays by dividing by 2) the storage required would be roughly n*log(n) + n. In the case of reducing the linear algorithm's storage one could simply read the file line by line and store only a few values while doing the computation along the way. This approach shouldn't improve overall performance of the algorithm because it still reads each element as well as temporarily loads the element into memory at least once.

• Research the Python library timeit (docs.python.org/3/library/timeit. html). The documentation states that the timeit library "avoids a number of common traps for measuring execution times". Discuss what these common traps are. What system effects might cause the measurement of the run time of an algorithm to be inaccurate. How does the timeit library overcome these traps and system effects? How does the timeit library fail to control for system effects?

I found a post on stack overflow that describes a common trap that timeit takes into account:

"The default timer function is platform dependent. On Windows, time.clock() has microsecond granularity but time.time()'s granularity is 1/60th of a second; on Unix, time.clock() has 1/100th of a second granularity and time.time() is much more precise. On either platform, the default timer functions measure wall clock time, not the CPU time. This means that other processes running on the same computer may interfere with the timing ... On Unix, you can use time.clock() to measure CPU time."

```
if sys.platform == "win32":
  # On Windows, the best timer is time.clock()
  default_timer = time.clock
else:
  # On most other platforms the best timer is time.time()
  default_timer = time.time
```

Also from another source I found:

"The timeit module provides a simple interface for determining the execution time of small bits of Python code. It uses a platform-specific time function to provide the most accurate time calculation

possible. It reduces the impact of startup or shutdown costs on the time calculation by executing the code repeatedly."

Another possible issue, that I can think of, that may cause inaccuracy in time recorded is the provided hardware timeit is run on. If the CPU is running at 100% and is falling behind calls then the CPU schedule might interrupt the timeit calls which could theoretically cause latency.

Works Cited

https://docs.python.org/2/library/timeit.html. n.d.

https://pymotw.com/2/timeit/. n.d.

https://stackoverflow.com/questions/1685221/accurately-measure-time-python-function-takes. n.d. Stack Overflow.

https://wiki.python.org/moin/TimeComplexity. n.d.

https://www.geeksforgeeks.org/largest-sum-contiguous-subarray/?fbclid=IwAR1EeUJvHS0MAImzqIJdm-M3PXDu6pGcqpshz9yQjMpVLT_3yjA-1sXhHHc. n.d. GeeksforGeeks.

https://www.geeksforgeeks.org/maximum-subarray-sum-using-divide-and-conquer-algorithm/. n.d. GeeksforGeeks.