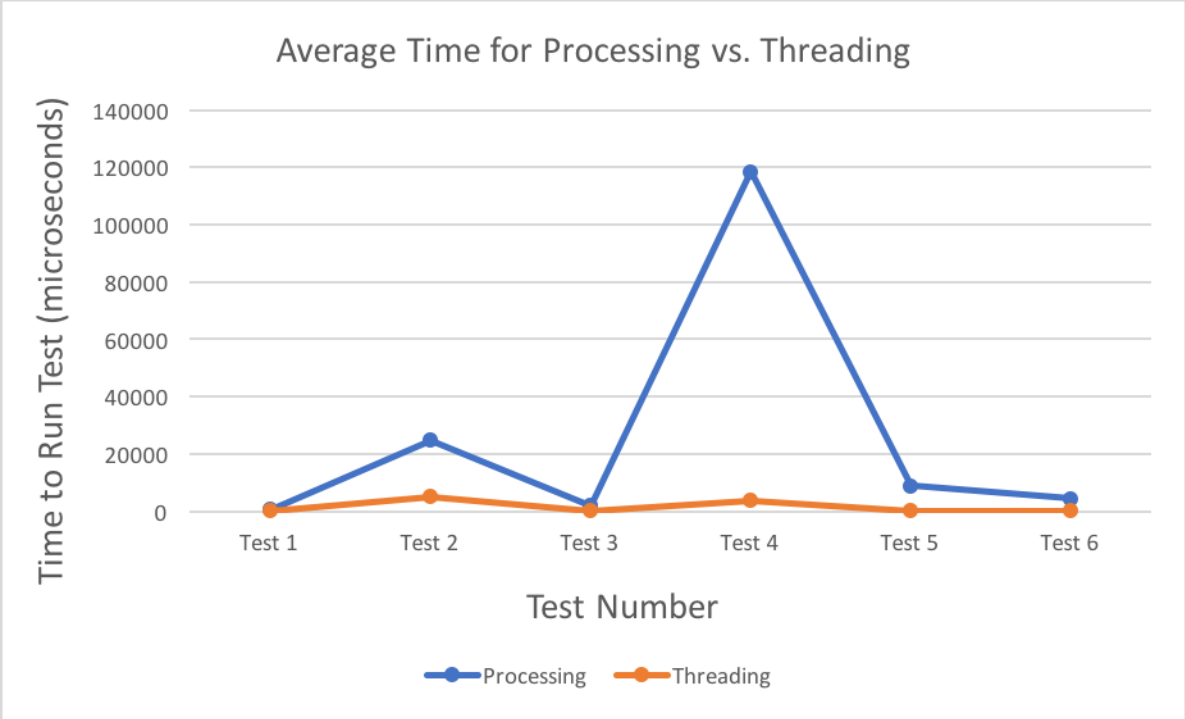
Ferris Hussein
Results Document for Assignment 2: Spooky Searching

	Test Case 1	Test Case 2	Test Case 3	Test Case 4	Test Case 5	Test Case 6

Processing	int size = 1000;
	int target = 1001;
	int block = 250;
		int size = 54321;
		int target = 12345;
		int block = 250;
			int size = 2344;
			int target = 1;
			int block = 250;
				int size = 100000;
				int target = 32982;
				int block = 250;
					int size = 2019;
					int target = 2019;
					int block = 100;
						int size = 2019;
						int target = 2019;
						int block = 250;

Threading	int size = 1000;
	int target = 1001;
	int block = 250;
		int size = 54321;
		int target = 12345;
		int block = 250;
			int size = 2344;
			int target = 1;
			int block = 250;
				int size = 100000;
				int target = 32982;
				int block = 250;
					int size = 2019;
					int target = 2019;
					int block = 100;
						int size = 2019;
						int target = 2019;
						int block = 250;

One interesting aspect to take from this first table is from test cases 5 and 6. For both multi-processing and multi-threading, having block sizes of 250 seem to be more efficient than having block sizes of 100. This follows general logic, since you would need more threads and processes to run the array is chunked into smaller sizes. More processes/threads leads to more CPU usage, along with more time to run through each one, as shown from test cases 5 and 6. The rest of the data will be analyzed below.



For the first main comparison, I ran 6 test cases, each 100 times, and took the average times each method took in order to find which was faster, either multi-processing or multi-threading. As you can see in the table, each test case was the same for processing and threading in order to get an accurate representation of how long each method took. As expected, multi-threading is faster than multi-processing. However, this difference is more apparent once the size of the integer array is very large, as you can see from test cases 2 and 4. Pictures for the actual data will be provided at the end.

	Test Case 1	Test Case 2	Test Case 3	Test Case 4	Test Case 5	Test Case 6
Processing	Min: 254 Max: 1192 SD: 258	Min: 9145 Max: 38362 SD: 7968	Min:1779 Max: 2227 SD: 24263	Min: 70264 Max: 160360 SD: 94008	Min: 8412 Max: 9440 SD: 17920	Min: 3956 Max: 5534 SD: 22033
Threading	Min: 82 Max: 2777 SD: 278	Min: 4062 Max: 8624 SD: 524	Min: 194 Max: 538 SD: 4753	Min: 1900 Max: 10825 SD: 1264	Min: 100 Max: 319 SD: 4884	Min: 97 Max: 539 SD: 4882

All these pieces of data Ire retrieved from 100 runs of each test case; each test case has the same parameters as the one listed earlier in the first table. All pieces of data are measured in microseconds as well. As you can see, multi-threading takes significantly shorter time to complete than multi-processing. While the maximum value for threading may be larger than processing for test case 1, I can assume this is an outlier since all other pieces of data follow the idea that multi-threading takes a shorter amount of time.

PROCESS

```
Array Size: 250
        Looking For: 100
        Target found in index: 39
        Check: randArray[39] = 100
        Time Taken: 374.000000 microseconds

Array Size: 500
        Looking For: 100
        Target found in index: 77
        Check: randArray[77] = 100
        Time Taken: 507.000000 microseconds

Array Size: 750
        Looking For: 100
        Target found in index: 67
        Check: randArray[67] = 100
        Time Taken: 605.000000 microseconds

Array Size: 1000
        Looking For: 100
        Target found in index: 513
        Check: randArray[513] = 100
        Time Taken: 672.000000 microseconds

Array Size: 1250
        Looking For: 100
        Target found in index: 1034
        Check: randArray[1034] = 100
        Time Taken: 815.000000 microseconds

Array Size: 1500
        Looking For: 100
        Target found in index: 842
        Check: randArray[842] = 100
        Time Taken: 1042.000000 microseconds

Array Size: 1750
        Looking For: 100
        Target found in index: 558
        Check: randArray[558] = 100
        Time Taken: 1023.000000 microseconds

Array Size: 2000
        Looking For: 100
        Target found in index: 158
        Check: randArray[158] = 100
        Time Taken: 1439.000000 microseconds

Array Size: 2250
        Looking For: 100
        Target found in index: 911
        Check: randArray[911] = 100
        Time Taken: 1348.000000 microseconds

Array Size: 2500
```

THREAD

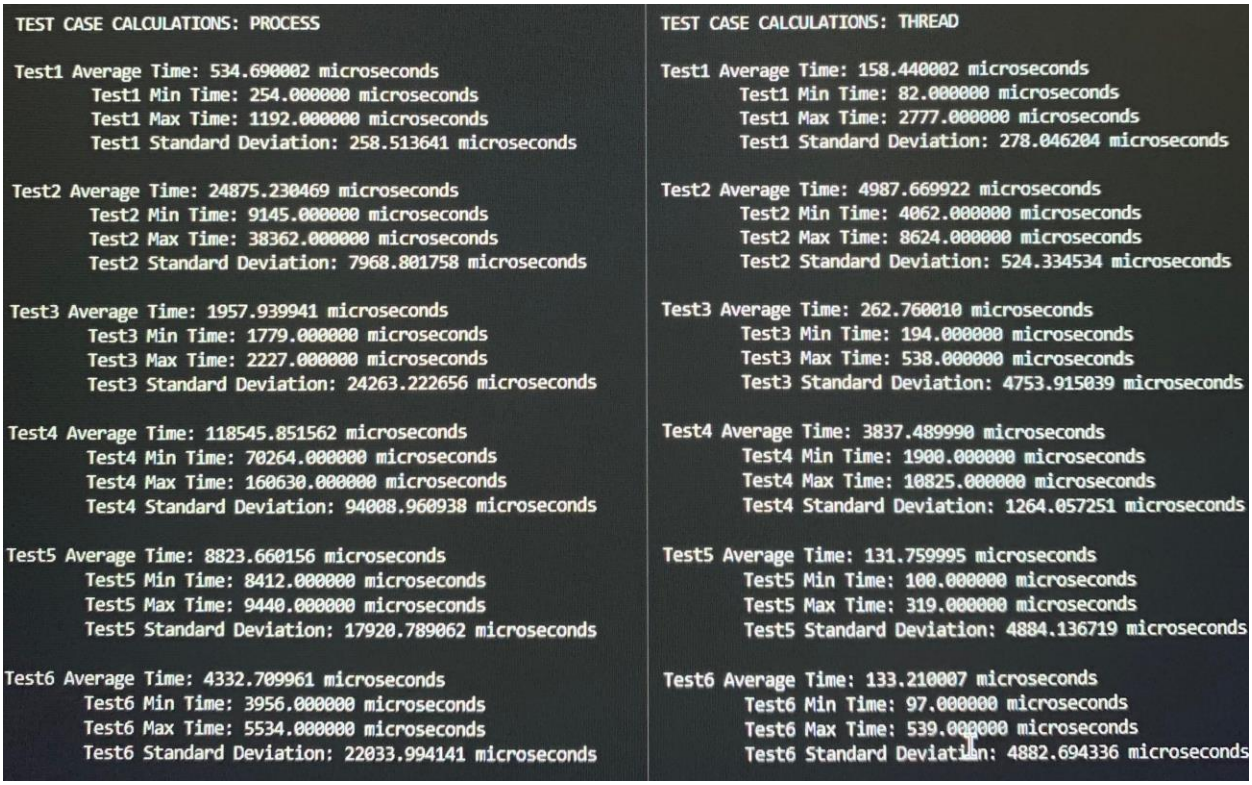
```
Array Size: 250
        Looking For: 100
        Target found in index: 39
        Check: randArray[39] = 100
        Time Taken: 652.000000 microseconds

Array Size: 500
        Looking For: 100
        Target found in index: 77
        Check: randArray[77] = 100
        Time Taken: 354.000000 microseconds

Array Size: 750
        Looking For: 100
        Target found in index: 67
        Check: randArray[67] = 100
        Time Taken: 568.000000 microseconds

Array Size: 1000
        Looking For: 100
        Target found in index: 513
        Check: randArray[513] = 100
        Time Taken: 106.000000 microseconds

Array Size: 1250
        Looking For: 100
        Target found in index: 1034
        Check: randArray[1034] = 100
        Time Taken: 294.000000 microseconds

Array Size: 1500
        Looking For: 100
        Target found in index: 842
        Check: randArray[842] = 100
        Time Taken: 250.000000 microseconds

Array Size: 1750
        Looking For: 100
        Target found in index: 558
        Check: randArray[558] = 100
        Time Taken: 268.000000 microseconds

Array Size: 2000
        Looking For: 100
        Target found in index: 158
        Check: randArray[158] = 100
        Time Taken: 295.000000 microseconds

Array Size: 2250
        Looking For: 100
        Target found in index: 911
        Check: randArray[911] = 100
        Time Taken: 319.000000 microseconds

Array Size: 2500
```

```
        Check: randArray[158] = 100
        Time Taken: 1439.000000 microseconds

Array Size: 2250
        Looking For: 100
        Target found in index: 911
        Check: randArray[911] = 100
        Time Taken: 1348.000000 microseconds

Array Size: 2500
        Looking For: 100
        Target found in index: 1296
        Check: randArray[1296] = 100
        Time Taken: 1686.000000 microseconds

Array Size: 2750
        Looking For: 100
        Target found in index: 789
        Check: randArray[789] = 100
        Time Taken: 1468.000000 microseconds

Array Size: 3000
        Looking For: 100
        Target found in index: 1866
        Check: randArray[1866] = 100
        Time Taken: 1750.000000 microseconds

Array Size: 3250
        Looking For: 100
        Target found in index: 660
        Check: randArray[660] = 100
        Time Taken: 1728.000000 microseconds

Array Size: 3500
        Looking For: 100
        Target found in index: 29
        Check: randArray[29] = 100
        Time Taken: 2000.000000 microseconds

Array Size: 3750
        Looking For: 100
        Target found in index: 326
        Check: randArray[326] = 100
        Time Taken: 2053.000000 microseconds

Array Size: 4000
        Looking For: 100
        Target found in index: 2006
        Check: randArray[2006] = 100
        Time Taken: 2210.000000 microseconds

Array Size: 4250
        Looking For: 100
        Target found in index: 2773
        Check: randArray[2773] = 100
        Time Taken: 2288.000000 microseconds

Array Size: 4500
```

```
        Check: randArray[158] = 100
        Time Taken: 295.000000 microseconds

Array Size: 2250
        Looking For: 100
        Target found in index: 911
        Check: randArray[911] = 100
        Time Taken: 319.000000 microseconds

Array Size: 2500
        Looking For: 100
        Target found in index: 1296
        Check: randArray[1296] = 100
        Time Taken: 382.000000 microseconds

Array Size: 2750
        Looking For: 100
        Target found in index: 789
        Check: randArray[789] = 100
        Time Taken: 397.000000 microseconds

Array Size: 3000
        Looking For: 100
        Target found in index: 1866
        Check: randArray[1866] = 100
        Time Taken: 511.000000 microseconds

Array Size: 3250
        Looking For: 100
        Target found in index: 660
        Check: randArray[660] = 100
        Time Taken: 616.000000 microseconds

Array Size: 3500
        Looking For: 100
        Target found in index: 29
        Check: randArray[29] = 100
        Time Taken: 639.000000 microseconds

Array Size: 3750
        Looking For: 100
        Target found in index: 326
        Check: randArray[326] = 100
        Time Taken: 558.000000 microseconds

Array Size: 4000
        Looking For: 100
        Target found in index: 2006
        Check: randArray[2006] = 100
        Time Taken: 577.000000 microseconds

Array Size: 4250
        Looking For: 100
        Target found in index: 2773
        Check: randArray[2773] = 100
        Time Taken: 592.000000 microseconds

Array Size: 4500
```

The two screenshots posted above give you an idea of how long each method takes based on the size of the array. These values Ire obtained from the tradeOff() function in my program. These screenshots further support my hypothesis that multi-threading is faster than multi-processing. What is interesting to notice, as mentioned further, is that for very small array sizes, processing is slightly faster than threading. However, as the array size grows, the time for running multi-processing grows in an almost linear fashion while multi-threading stays somewhat consistent. This fact allows us to assume that for large sets of data, it is more efficient to run multi-threading, granted that the user has enough memory available to handle multi-threading. The

crossover where multi-threading becomes faster than multi-processing happens after the array size grows from 250 integers to 500 integers. Since the time to complete multi-threading stays rather similar up to array sizes of 4250 integers, you can assume that it would take a significantly large array (size of 10,000 integers+) for multi-threading to take the same time as multi-processing would take for an array size of say, 1,000 integers.



This final picture is the data I collected and inserted into tables, but condensed into 1 picture for ease of view.