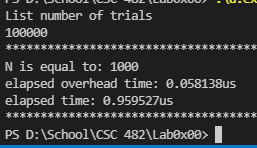
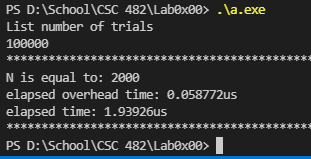
Austin Stau

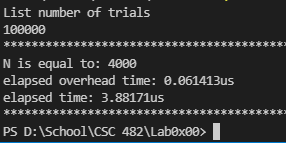
Lab0x00

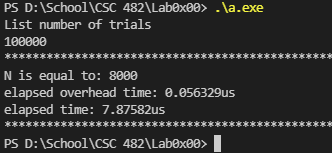
**Linear Search:** t(n) ~ C \* N is expected to take this time because it needs to check every part of the array to do this and once, we add more to N it will take more time or a product of that time. Because we have a single for loop searching for the correct number this is linear.

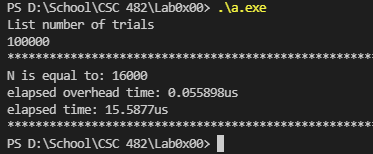
I used 100000 trials for this

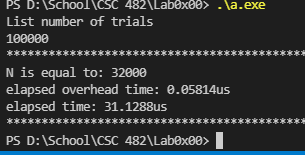
N = 1000 

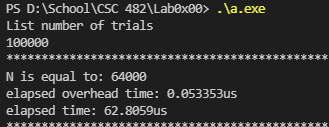
N = 2000 

N = 4000 

N = 8000 

N = 16000 

N = 32000 

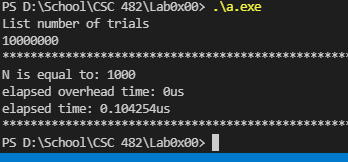
N = 64000 

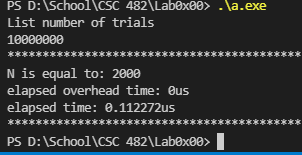
|  |  |  |  |
| --- | --- | --- | --- |
| List of size N | Avg Experimental Time\*\* | Experimental  Doubling ratio | Theoretical  Doubling Ratio |
| 1000 | 0.959527us | n/a | n/a |
| 2000 | 1.93926us | 2.021 | 2 |
| 4000 | 3.88171us | 2.002 | 2 |
| 8000 | 7.897582us | 2.035 | 2 |
| 16000 | 15.5877us | 1.974 | 2 |
| 32000 | 31.1288us | 1.997 | 2 |
| 64000 | 62.8059us | 2.018 | 2 |

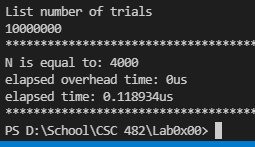
The results seem to show that even after the first doubling it aligns with the theoretical doubling ratio and continues to show that it is close to 2 doubling ratio. This shows that this takes t(N) ~C \* N time.

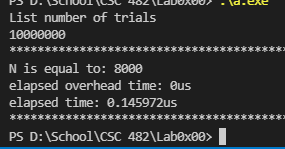
**Binary Search**: This takes t(n) ~ C \* log(n) because instead of looking all the parts of the array, this algorithm splits this in half, so it halves the linear search time by halving the array. Still as N grows so does the time, but not as fast as linear time.

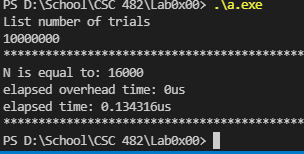
Used 10,000,000 trials for this.

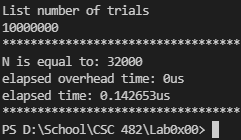
N = 1000 

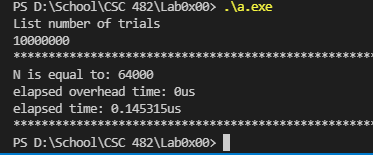
N = 2000 

N = 4000 

N = 8000 

N = 16000 

N = 32000 

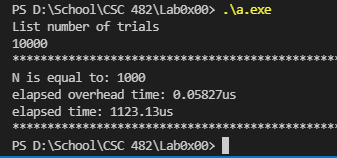
N = 64000 

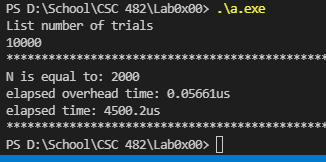
|  |  |  |  |
| --- | --- | --- | --- |
| List of size N | Avg Experimental Time\*\* | Experimental  Doubling ratio | Theoretical  Doubling Ratio |
| 1000 | 0.104254us | n/a | n/a |
| 2000 | 0.112272us | 1.077 | 1 |
| 4000 | 0.118934us | 1.059 | 1 |
| 8000 | 0.145972us | 1.227 | 1 |
| 16000 | 0.134316us | 0.920 | 1 |
| 32000 | 0.142653us | 1.091 | 1 |
| 64000 | 0.145315us | 1.019 | 1 |

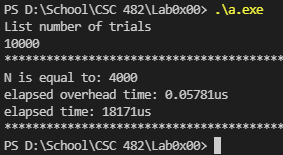
The size was around 1 from the start, which makes sense if we think about what this algorithm is doing. It a linear search algorithm, but it is halving the array and just working with that section and then works its way through it. So, it would make sense that the doubling ratio would also be half of what a linear search would be.

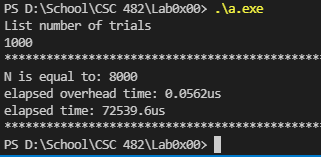
**Simple Sort (bubble sort):** t(N) ~ C \* N2 is the expected outcome for this because as we add more to the N every time the time will increase N square times. This is because we have two different linear loops running together. We know that linear time is calculated as C \*N well if we have two of these with the same C then we have C \* N2.

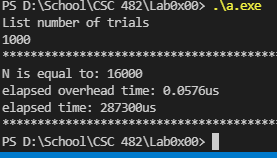
I ran this with 10,000 trials, had to change to 1000 trials at N = 8000, then down to 100 trials for N = 32000 and N = 64000

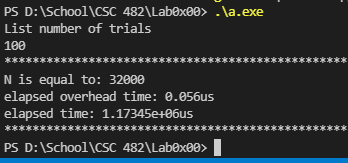
N = 1000 

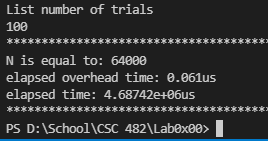
N = 2000 

N = 4000 

N = 8000 

N = 16000 

N = 32000 

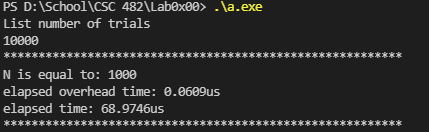
N = 64000 

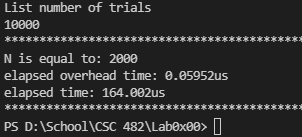
|  |  |  |  |
| --- | --- | --- | --- |
| List of size N | Avg Experimental Time\*\* | Experimental  Doubling ratio | Theoretical  Doubling Ratio |
| 1000 | 1123.13us | n/a | n/a |
| 2000 | 4500.2us | 4.007 | 4 |
| 4000 | 18171us | 4.038 | 4 |
| 8000 | 72539.6us | 3.992 | 4 |
| 16000 | 287300us | 3.961 | 4 |
| 32000 | 1173450us | 4.084 | 4 |
| 64000 | 4687420us | 3.995 | 4 |

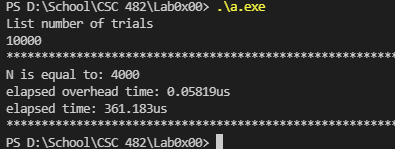
From the start we see that the experimental doubling ratio is close to the theoretical doubling ratio. It shows that with two linear loops we have N2 time when running this algorithm.

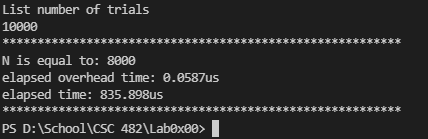
**Efficient Sort (Quick Sort):** This algorithm takes N\*log(N) because the same idea with binary sort we halve the array with the part function and then work with that section, this gives us the log(N). We also have linear time to worry about as well when the loop is checking the array values, this gives us the N\*.

Started with 10,000 trials

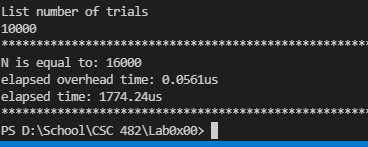
N = 1000 

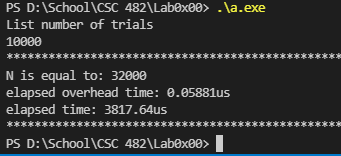
N = 2000 

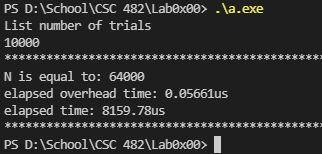
N = 4000 

N = 8000 

N = 1

N=6000 

N = 32000 

N = 64000 

|  |  |  |  |
| --- | --- | --- | --- |
| List of size N | Avg Experimental Time\*\* | Experimental  Doubling ratio | Theoretical  Doubling Ratio |
| 1000 | 68.9746us | n/a | n/a |
| 2000 | 164.002us | 2.378 | 2 |
| 4000 | 361.183us | 2.202 | 2 |
| 8000 | 835.898us | 2.314 | 2 |
| 16000 | 1774.24us | 2.122 | 2 |
| 32000 | 3817.64us | 2.151 | 2 |
| 64000 | 8159.78us | 2.137 | 2 |

As N gets larger, we see that the experimental ratio gets closer to the theoretical. This was to be expected of the result, although I never get under 2 this seems to work well enough. Although during testing I did see worst case O(N) = N2 by not shuffling the array correctly which meant this would sort a pre-sorted array, which would give this algorithm its worst case.