Data Structure and Algorithms

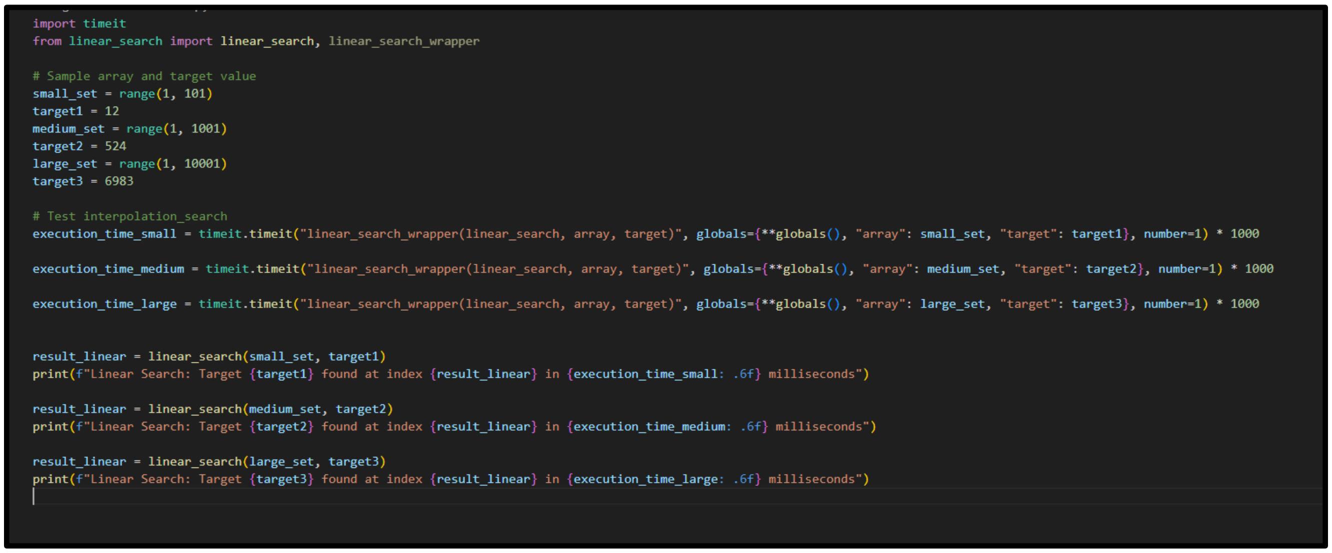
Lab 5 Group Exercises

Group # 2
Members:
Anoya, Alexie Ysabel B.
Albon, Charl B.
Bernabe, Aleckxis Kate
Dela Cruz, Lailanie E.
Marquez, Lianna L.
Morco, Catelyn Joy

RESULTS

I. *The Python code on how to generate the dataset and executing the data set:*

• *Linear search*



```python
import timeit
from linear_search import linear_search, linear_search_wrapper

# Sample array and target value
small_set = range(1, 101)
target1 = 12
medium_set = range(1, 1001)
target2 = 524
large_set = range(1, 10001)
target3 = 6983

# Test interpolation_search
execution_time_small = timeit.timeit("linear_search_wrapper(linear_search, array, target)", globals={**globals(), "array": small_set, "target": target1}, number=1) * 1000

execution_time_medium = timeit.timeit("linear_search_wrapper(linear_search, array, target)", globals={**globals(), "array": medium_set, "target": target2}, number=1) * 1000

execution_time_large = timeit.timeit("linear_search_wrapper(linear_search, array, target)", globals={**globals(), "array": large_set, "target": target3}, number=1) * 1000


result_linear = linear_search(small_set, target1)
print(f"Linear Search: Target {target1} found at index {result_linear} in {execution_time_small: .6f} milliseconds")

result_linear = linear_search(medium_set, target2)
print(f"Linear Search: Target {target2} found at index {result_linear} in {execution_time_medium: .6f} milliseconds")

result_linear = linear_search(large_set, target3)
print(f"Linear Search: Target {target3} found at index {result_linear} in {execution_time_large: .6f} milliseconds")
```

Code in text

```python
import timeit
from linear_search import linear_search, linear_search_wrapper

# Sample array and target value
small_set = range(1, 101)
target1 = 12
medium_set = range(1, 1001)
target2 = 524
large_set = range(1, 10001)
target3 = 6983

# Test interpolation_search
execution_time_small = timeit.timeit("linear_search_wrapper(linear_search, array, target)",
globals={**globals(), "array": small_set, "target": target1}, number=1) * 1000

execution_time_medium = timeit.timeit("linear_search_wrapper(linear_search, array, target)",
globals={**globals(), "array": medium_set, "target": target2}, number=1) * 1000
```

*execution_time_large = timeit.timeit("linear_search_wrapper(linear_search, array, target)", globals={**globals(), "array": large_set, "target": target3}, number=1) * 1000*

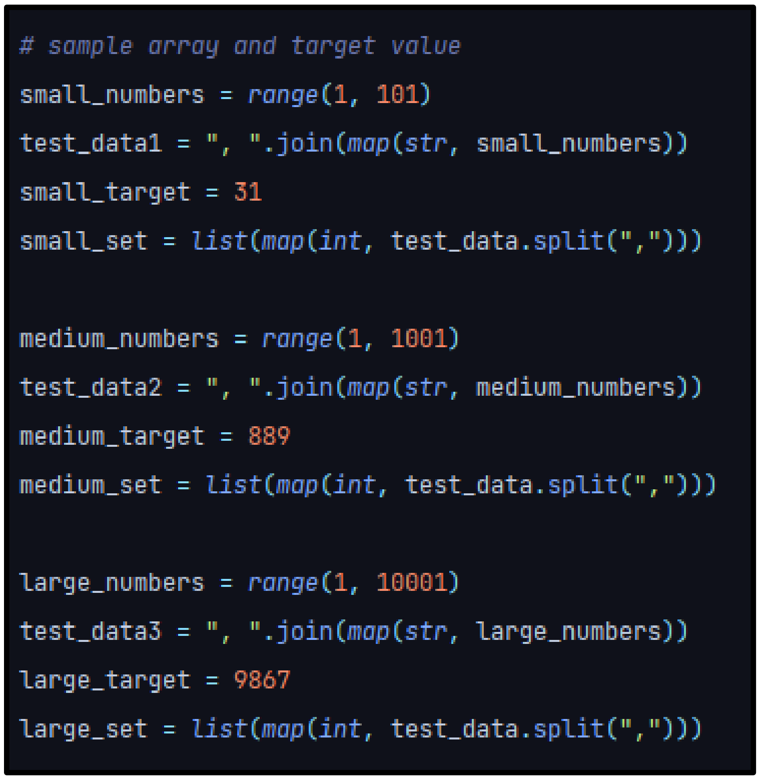
result_linear = linear_search(small_set, target1)
print(f"Linear Search: Target {target1} found at index {result_linear} in {execution_time_small: .6f} milliseconds")

result_linear = linear_search(medium_set, target2)
print(f"Linear Search: Target {target2} found at index {result_linear} in {execution_time_medium: .6f} milliseconds")

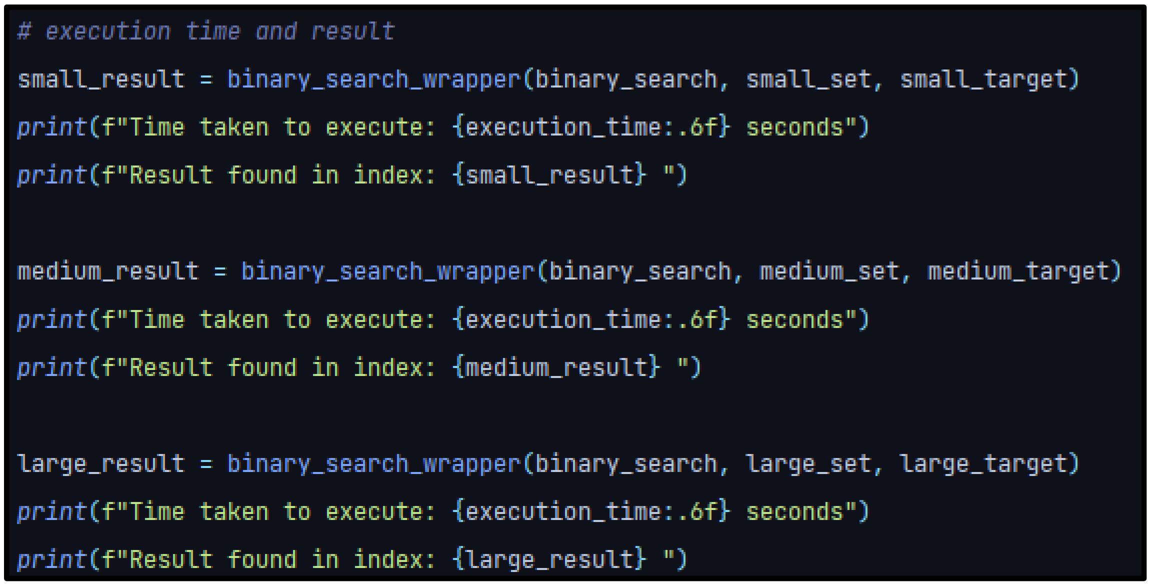
result_linear = linear_search(large_set, target3)
print(f"Linear Search: Target {target3} found at index {result_linear} in {execution_time_large: .6f} milliseconds"

• ***Binary search***

Code in picture:



```
# sample array and target value
small_numbers = range(1, 101)
test_data1 = ", ".join(map(str, small_numbers))
small_target = 31
small_set = list(map(int, test_data.split(",")))

medium_numbers = range(1, 1001)
test_data2 = ", ".join(map(str, medium_numbers))
medium_target = 889
medium_set = list(map(int, test_data.split(",")))

large_numbers = range(1, 10001)
test_data3 = ", ".join(map(str, large_numbers))
large_target = 9867
large_set = list(map(int, test_data.split(",")))
```



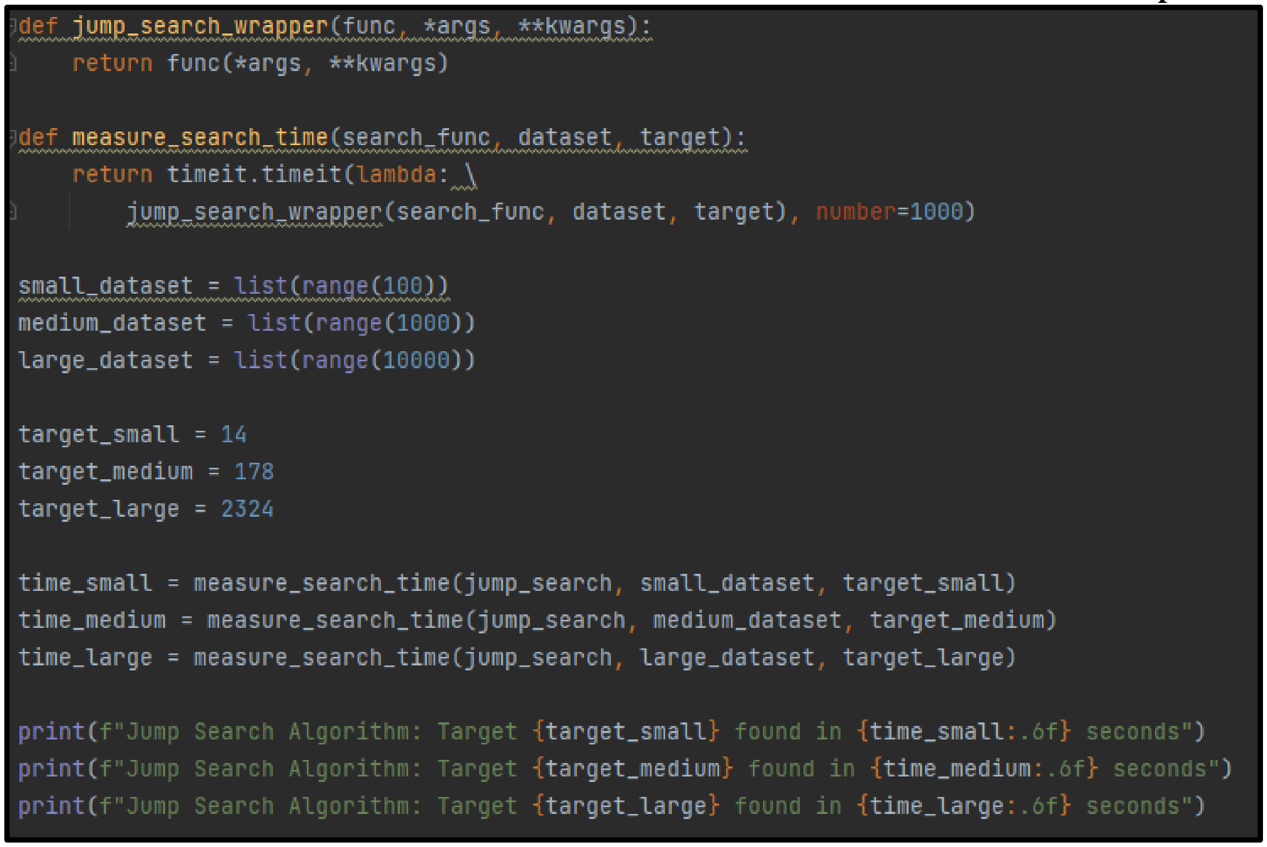
```
# execution time and result
small_result = binary_search_wrapper(binary_search, small_set, small_target)
print(f"Time taken to execute: {execution_time:.6f} seconds")
print(f"Result found in index: {small_result} ")

medium_result = binary_search_wrapper(binary_search, medium_set, medium_target)
print(f"Time taken to execute: {execution_time:.6f} seconds")
print(f"Result found in index: {medium_result} ")

large_result = binary_search_wrapper(binary_search, large_set, large_target)
print(f"Time taken to execute: {execution_time:.6f} seconds")
print(f"Result found in index: {large_result} ")
```

Code in text:

```
# sample array and target value
small_numbers = range(1, 101)
test_data1 = ", ".join(map(str, small_numbers))
small_target = 31
small_set = list(map(int, test_data.split(",")))
medium_numbers = range(1, 1001)
test_data2 = ", ".join(map(str, medium_numbers))
medium_target = 889
medium_set = list(map(int, test_data.split(",")))

large_numbers = range(1, 10001)
test_data3 = ", ".join(map(str, large_numbers))
large_target = 9867
large_set = list(map(int, test_data.split(",")))

# execution time and result
small_result = binary_search_wrapper(binary_search, small_set, small_target)
print(f"Time taken to execute: {execution_time:.6f} seconds")
print(f"Result found in index: {small_result} ")

medium_result = binary_search_wrapper(binary_search, medium_set, medium_target)
print(f"Time taken to execute: {execution_time:.6f} seconds")
print(f"Result found in index: {medium_result} ")
```

large_result = binary_search_wrapper(binary_search, large_set, large_target)
print(f"Time taken to execute: {execution_time:.6f} seconds")
print(f"Result found in index: {large_result} ")

• ***Jump search Code in picture:***



```
def jump_search_wrapper(func, *args, **kwargs):
    return func(*args, **kwargs)

def measure_search_time(search_func, dataset, target):
    return timeit.timeit(lambda: \
        jump_search_wrapper(search_func, dataset, target), number=1000)

small_dataset = list(range(100))
medium_dataset = list(range(1000))
large_dataset = list(range(10000))

target_small = 14
target_medium = 178
target_large = 2324

time_small = measure_search_time(jump_search, small_dataset, target_small)
time_medium = measure_search_time(jump_search, medium_dataset, target_medium)
time_large = measure_search_time(jump_search, large_dataset, target_large)

print(f"Jump Search Algorithm: Target {target_small} found in {time_small:.6f} seconds")
print(f"Jump Search Algorithm: Target {target_medium} found in {time_medium:.6f} seconds")
print(f"Jump Search Algorithm: Target {target_large} found in {time_large:.6f} seconds")
```

Code in text:

```
def jump_search_wrapper(func, *args, **kwargs):
    return func(*args, **kwargs)

def measure_search_time(search_func, dataset, target):
    return timeit.timeit(lambda: \
        jump_search_wrapper(search_func, dataset, target), number=1000)

small_dataset = list(range(100))
medium_dataset = list(range(1000))
large_dataset = list(range(10000))

target_small = 14
```

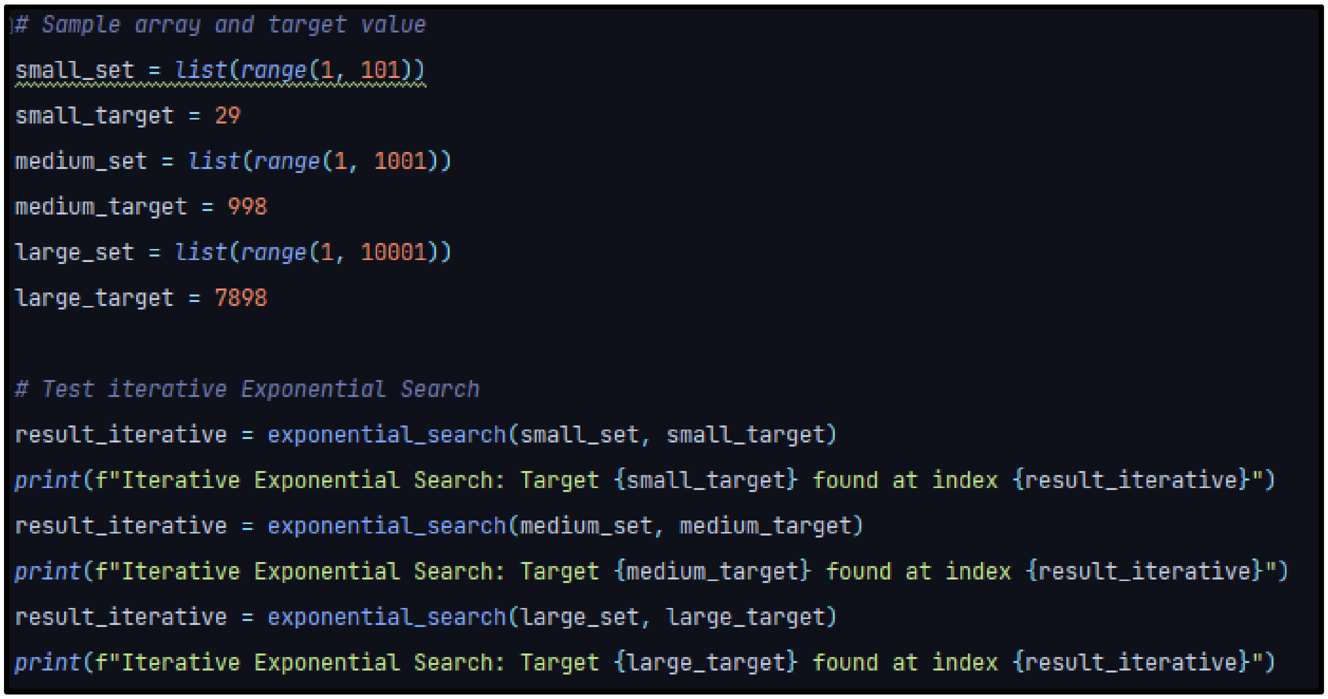
target_medium = 178
target_large = 2324

time_small = measure_search_time(jump_search, small_dataset, target_small)
time_medium = measure_search_time(jump_search, medium_dataset, target_medium)
time_large = measure_search_time(jump_search, large_dataset, target_large)

print(f"Jump Search Algorithm: Target {target_small} found in {time_small:.6f} seconds")
print(f"Jump Search Algorithm: Target {target_medium} found in {time_medium:.6f} seconds")
print(f"Jump Search Algorithm: Target {target_large} found in {time_large:.6f} seconds")

• Exponential search

Code in picture:



```python
# Sample array and target value
small_set = list(range(1, 101))
small_target = 29
medium_set = list(range(1, 1001))
medium_target = 998
large_set = list(range(1, 10001))
large_target = 7898

# Test iterative Exponential Search
result_iterative = exponential_search(small_set, small_target)
print(f"Iterative Exponential Search: Target {small_target} found at index {result_iterative}")
result_iterative = exponential_search(medium_set, medium_target)
print(f"Iterative Exponential Search: Target {medium_target} found at index {result_iterative}")
result_iterative = exponential_search(large_set, large_target)
print(f"Iterative Exponential Search: Target {large_target} found at index {result_iterative}")
```

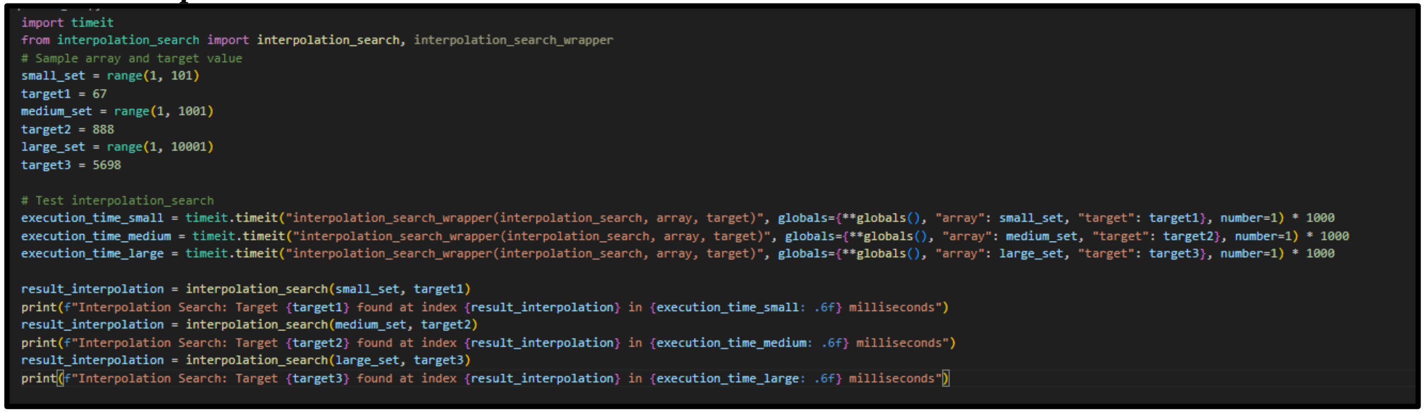
Code in text:

# Sample array and target value
small_set = list(range(1, 101))
small_target = 29
medium_set = list(range(1, 1001))
medium_target = 998
large_set = list(range(1, 10001))

large_target = 7898

# Test iterative Exponential Search
result_iterative = exponential_search(small_set, small_target)
print(f"Iterative Exponential Search: Target {small_target} found at index {result_iterative}")
result_iterative = exponential_search(medium_set, medium_target)
print(f"Iterative Exponential Search: Target {medium_target} found at index {result_iterative}")
result_iterative = exponential_search(large_set, large_target)
print(f"Iterative Exponential Search: Target {large_target} found at index {result_iterative}")

• Interpolation search
Code in picture:



```
import timeit
from interpolation_search import interpolation_search, interpolation_search_wrapper
# Sample array and target value
small_set = range(1, 101)
target1 = 67
medium_set = range(1, 1001)
target2 = 888
large_set = range(1, 10001)
target3 = 5698

# Test interpolation_search
execution_time_small = timeit.timeit("interpolation_search_wrapper(interpolation_search, array, target)", globals={**globals(), "array": small_set, "target": target1}, number=1) * 1000
execution_time_medium = timeit.timeit("interpolation_search_wrapper(interpolation_search, array, target)", globals={**globals(), "array": medium_set, "target": target2}, number=1) * 1000
execution_time_large = timeit.timeit("interpolation_search_wrapper(interpolation_search, array, target)", globals={**globals(), "array": large_set, "target": target3}, number=1) * 1000

result_interpolation = interpolation_search(small_set, target1)
print(f"Interpolation Search: Target {target1} found at index {result_interpolation} in {execution_time_small: .6f} milliseconds")
result_interpolation = interpolation_search(medium_set, target2)
print(f"Interpolation Search: Target {target2} found at index {result_interpolation} in {execution_time_medium: .6f} milliseconds")
result_interpolation = interpolation_search(large_set, target3)
print(f"Interpolation Search: Target {target3} found at index {result_interpolation} in {execution_time_large: .6f} milliseconds")
```

Code in text:
import timeit
from interpolation_search import interpolation_search, interpolation_search_wrapper

# Sample array and target value
small_set = range(1, 101)
target1 = 67
medium_set = range(1, 1001)
target2 = 888
large_set = range(1, 10001)
target3 = 5698

# Test interpolation_search
*execution_time_small = timeit.timeit("interpolation_search_wrapper(interpolation_search, array, target)", globals={**globals(), "array": small_set, "target": target1}, number=1) * 1000*

*execution_time_medium = timeit.timeit("interpolation_search_wrapper(interpolation_search, array, target)", globals={**globals(), "array": medium_set, "target": target2}, number=1) * 1000*

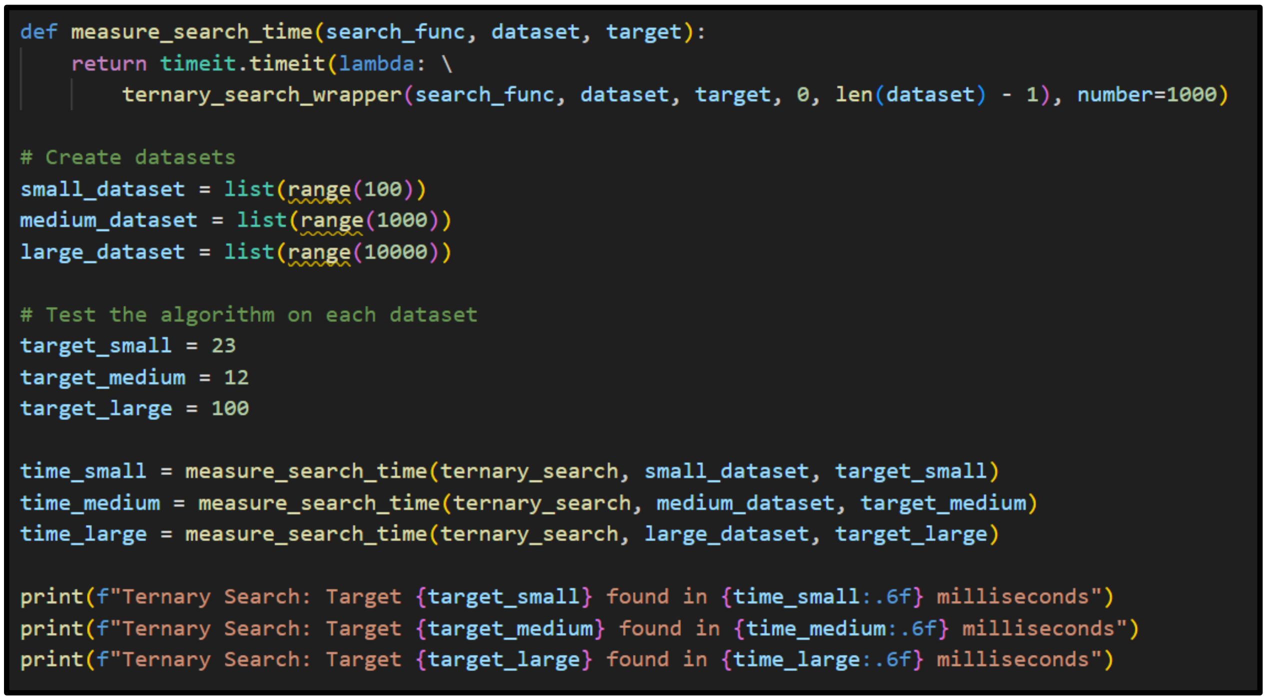
*execution_time_large = timeit.timeit("interpolation_search_wrapper(interpolation_search, array, target)", globals={**globals(), "array": large_set, "target": target3}, number=1) * 1000*

result_interpolation = interpolation_search(small_set, target1)
print(f"Interpolation Search: Target {target1} found at index {result_interpolation} in {execution_time_small: .6f} milliseconds")

result_interpolation = interpolation_search(medium_set, target2)
print(f"Interpolation Search: Target {target2} found at index {result_interpolation} in {execution_time_medium: .6f} milliseconds")

result_interpolation = interpolation_search(large_set, target3)
print(f"Interpolation Search: Target {target3} found at index {result_interpolation} in {execution_time_large: .6f} milliseconds")

• Ternary Search
Code in picture:



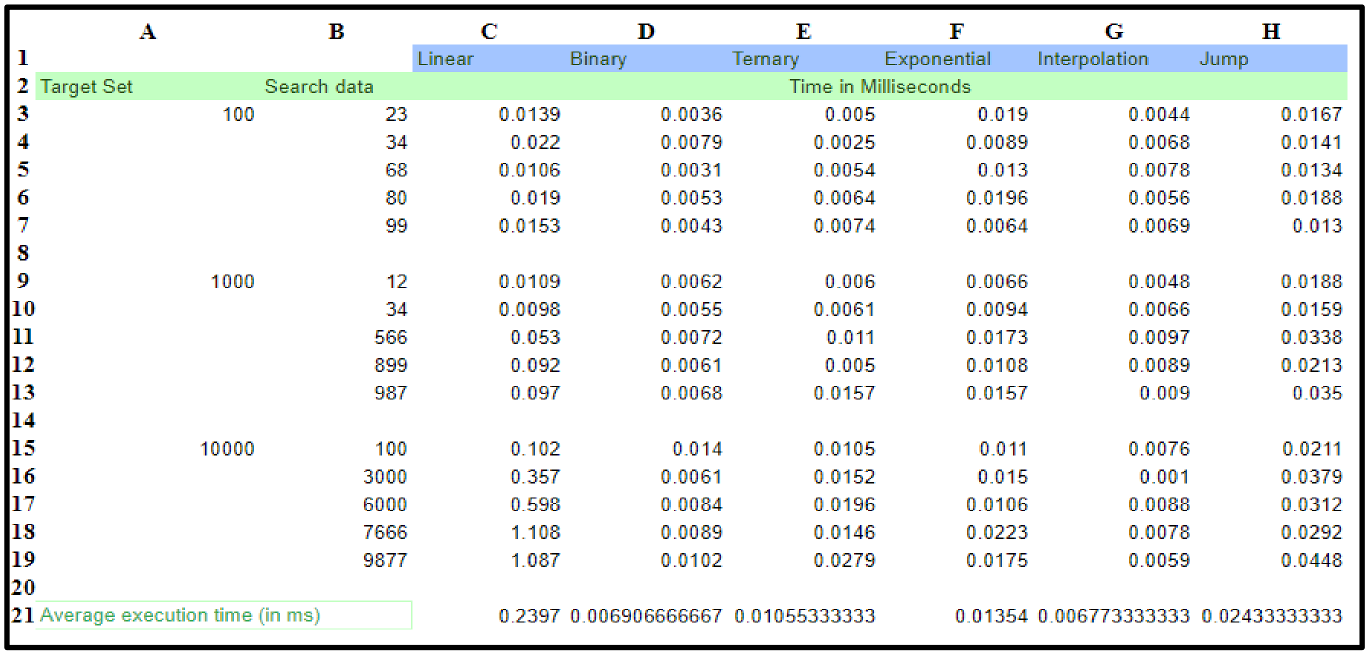
```
def measure_search_time(search_func, dataset, target):
    return timeit.timeit(lambda: \
        ternary_search_wrapper(search_func, dataset, target, 0, len(dataset) - 1), number=1000)

# Create datasets
small_dataset = list(range(100))
medium_dataset = list(range(1000))
large_dataset = list(range(10000))

# Test the algorithm on each dataset
target_small = 23
target_medium = 12
target_large = 100

time_small = measure_search_time(ternary_search, small_dataset, target_small)
time_medium = measure_search_time(ternary_search, medium_dataset, target_medium)
time_large = measure_search_time(ternary_search, large_dataset, target_large)

print(f"Ternary Search: Target {target_small} found in {time_small:.6f} milliseconds")
print(f"Ternary Search: Target {target_medium} found in {time_medium:.6f} milliseconds")
print(f"Ternary Search: Target {target_large} found in {time_large:.6f} milliseconds")
```

Code in text:

```
def measure_search_time(search_func, dataset, target):
    return timeit.timeit(lambda: \
        ternary_search_wrapper(search_func, dataset, target, 0, len(dataset) - 1), number=1000)

# Create datasets
small_dataset = list(range(100))
medium_dataset = list(range(1000))
large_dataset = list(range(10000))

# Test the algorithm on each dataset
target_small = 23
target_medium = 12
target_large = 100

time_small = measure_search_time(ternary_search, small_dataset, target_small)
time_medium = measure_search_time(ternary_search, medium_dataset, target_medium)
time_large = measure_search_time(ternary_search, large_dataset, target_large)

print(f"Ternary Search: Target {target_small} found in {time_small:.6f} milliseconds")
print(f"Ternary Search: Target {target_medium} found in {time_medium:.6f} milliseconds")
print(f"Ternary Search: Target {target_large} found in {time_large:.6f} milliseconds"
```

II. ***Table and graph comparing the performance of the search algorithms on the different data sets and analysis and conclusion of the exercise:***

Table 1. Search Algorithm Analysis Worksheet



	A	B	C	D	E	F	G	H
1			Linear	Binary	Ternary	Exponential	Interpolation	Jump
2	Target Set	Search data			Time in Milliseconds			
3	100	23	0.0139	0.0036	0.005	0.019	0.0044	0.0167
4		34	0.022	0.0079	0.0025	0.0089	0.0068	0.0141
5		68	0.0106	0.0031	0.0054	0.013	0.0078	0.0134
6		80	0.019	0.0053	0.0064	0.0196	0.0056	0.0188
7		99	0.0153	0.0043	0.0074	0.0064	0.0069	0.013
8								
9	1000	12	0.0109	0.0062	0.006	0.0066	0.0048	0.0188
10		34	0.0098	0.0055	0.0061	0.0094	0.0066	0.0159
11		566	0.053	0.0072	0.011	0.0173	0.0097	0.0338
12		899	0.092	0.0061	0.005	0.0108	0.0089	0.0213
13		987	0.097	0.0068	0.0157	0.0157	0.009	0.035
14								
15	10000	100	0.102	0.014	0.0105	0.011	0.0076	0.0211
16		3000	0.357	0.0061	0.0152	0.015	0.001	0.0379
17		6000	0.598	0.0084	0.0196	0.0106	0.0088	0.0312
18		7666	1.108	0.0089	0.0146	0.0223	0.0078	0.0292
19		9877	1.087	0.0102	0.0279	0.0175	0.0059	0.0448
20								
21	Average execution time (in ms)		0.2397	0.006906666667	0.01055333333	0.01354	0.006773333333	0.02433333333

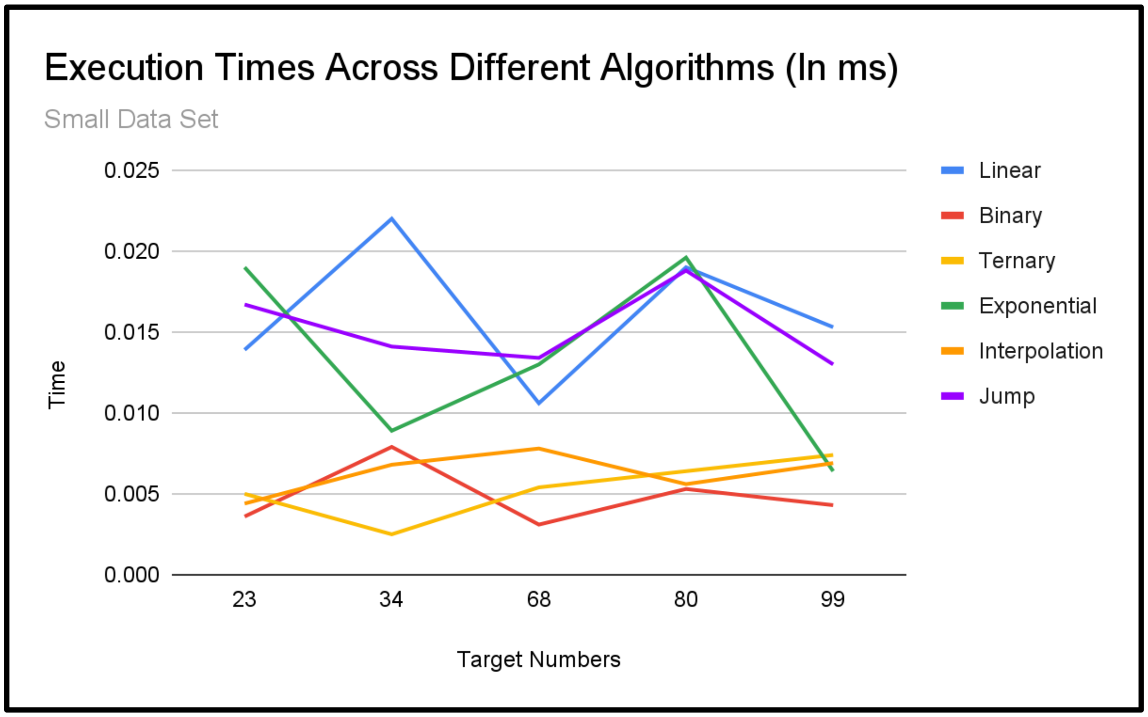


Figure 1.

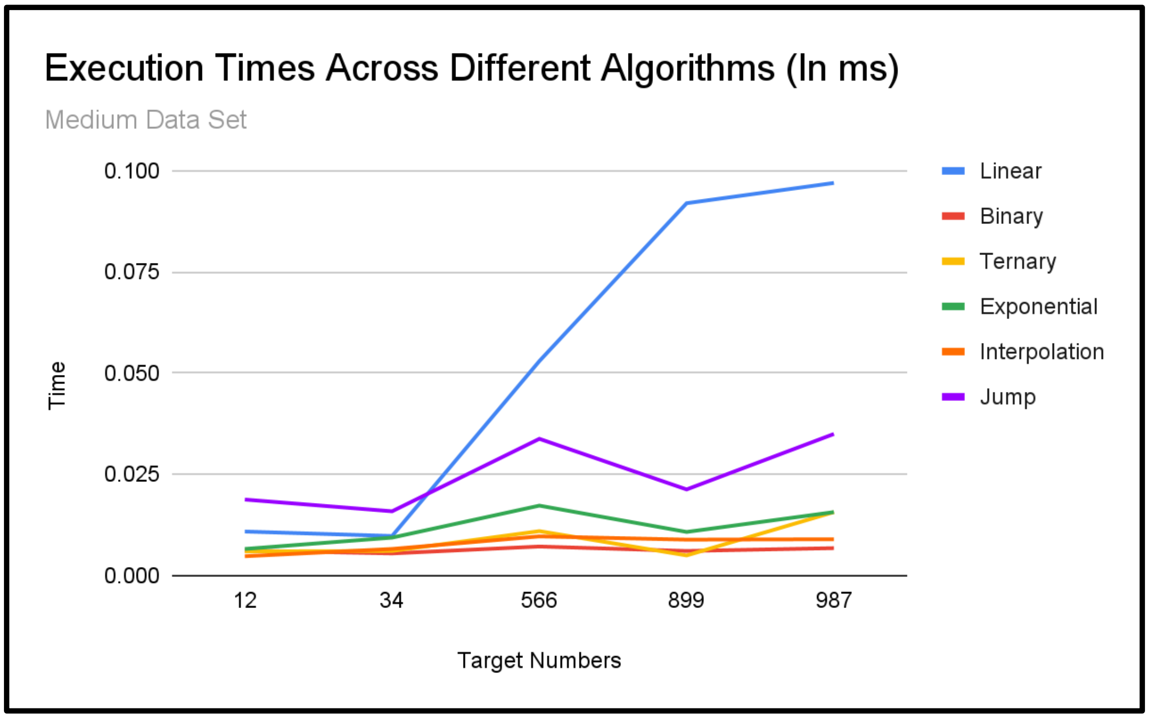


Figure 2.

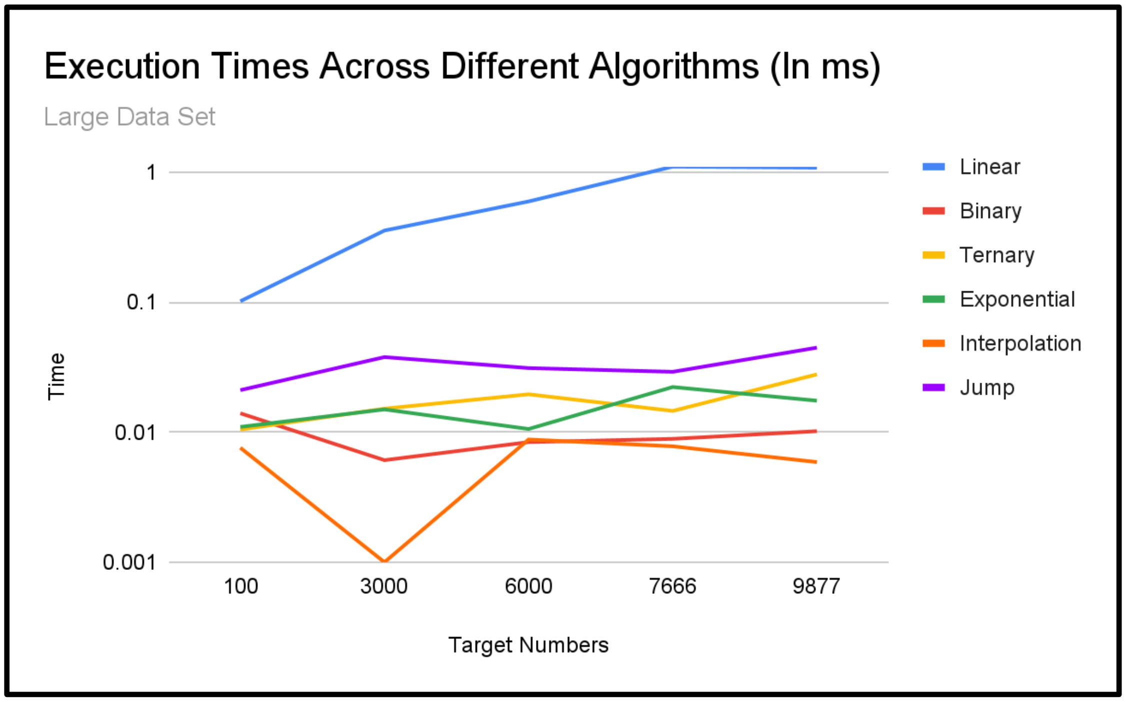


Figure 3.

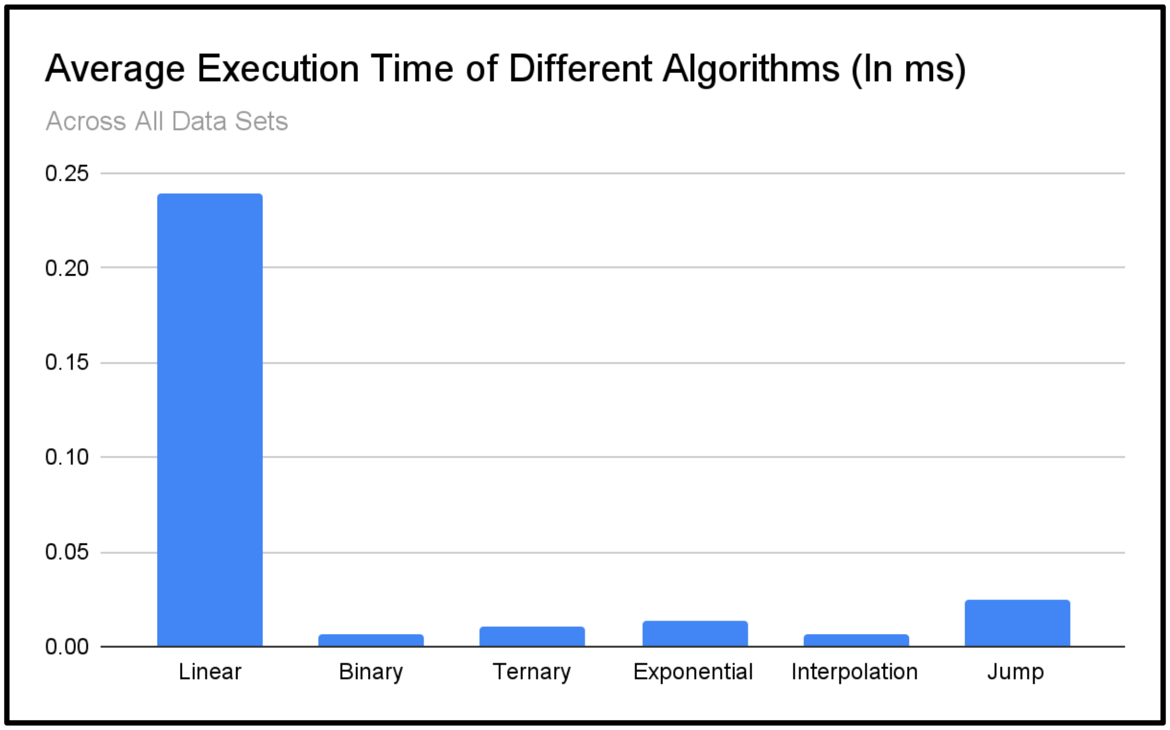


Figure 4.

Analysis and Conclusion
Create a table or graph to visualize the performance of each search algorithm on the different data sets.

Analyze the results and answer the following questions:

a. Which search algorithm performed the best overall?

As evident in Table 1 and the fourth figure shown above, the interpolation search algorithm performed the best overall. Across all the trials in the given target sets 100, 1000, and 10000, this type of search algorithm garnered the least average execution time. Therefore, this result only serves as evidence that among all the different algorithms, the interpolation search algorithm is the fastest.

b. Did any search algorithms perform better on specific data sets?

As shown in table 1, the average execution time of Interpolation is 0.0067ms making it the fastest search algorithm. Interpolation search works well on uniformly distributed and sorted data. It's particularly efficient when the data set has a linear distribution. Binary search is effective on sorted data sets. In cases where the data is not uniformly distributed, or there are repeated elements, it might perform better than interpolation search. Ternary search is useful when the data is sorted but not as effective as binary search in many cases. It divides the search space into three parts instead of two. Exponential search is efficient when there is a known upper bound on the size of the data set. It works particularly well when the target element is likely to be close to the beginning of the array. Jump search is effective for large, uniformly distributed data sets. It combines aspects of linear search and binary search, making it suitable for cases where binary search might be impractical. Linear search is generally less efficient than binary search on large, sorted datasets because it has only a linear time complexity. In all the given data sets (100,1000,10000), there is no better search algorithm to use than Interpolation.

c. How did the size of the data set affect the performance of the search algorithms?

Throughout the laboratory exercise, our close collaboration focused on analyzing and optimizing search algorithms, revealing a clear correlation between the size of the data and algorithm performance. As the size of the data set increased, so did the time complexity of the various search algorithms. This pattern demonstrated that larger data sets generally created difficulties for search operations, necessitating a review of algorithm choices and observation into optimization strategies to preserve efficiency when dealing with significant volumes of data.

d. Write a brief conclusion summarizing your findings

In conclusion, the average execution time of Interpolation is 0.0067ms making it the fastest search algorithm, while Binary has an average of 0.0069ms, Ternary has 0.0105ms, Exponential has 0.0135ms, Jump has 0.0143ms, and lastly is the Linear which has 0.2397ms that implies the slowest among all search algorithm. Moreover, the study highlighted the impact of data set size on algorithm performance. As the size of the data set increased, the time complexity of the search algorithms also escalated. This observation underscores the need for careful consideration and optimization of algorithm choices, especially when dealing with larger volumes of data. The efficiency of the interpolation search algorithm positions it as a favorable choice for scenarios where speed is crucial, while the study provides valuable insights into the nuanced relationship between algorithm performance and data set size.