[redacted] Case Study

[redacted]

Table of Contents

Table of Figures

Table of Tables

1. Introduction

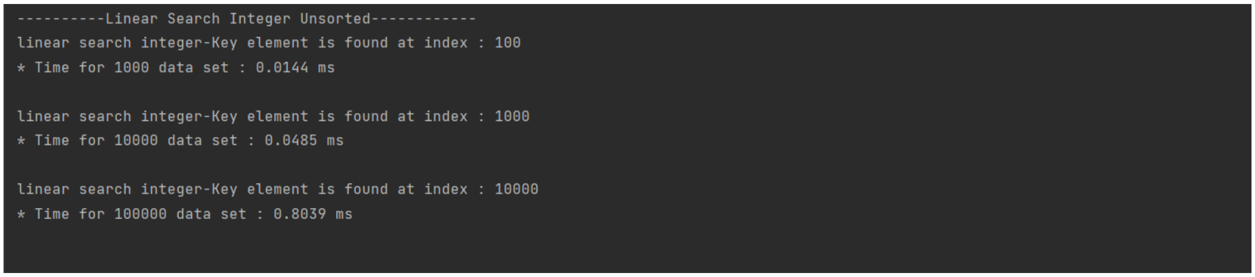
Performance modeling and analysis are critical tools for ensuring the cost-effective design and engineering of communications and computer systems, as well as services that rely on such systems. Appropriate application of performance modeling and analysis techniques can provide quantitative insight into system performance that would be difficult, expensive, or even impossible to obtain otherwise.

Whether it is done by a computer or a human, searching takes a long time in any application. There are numerous algorithms in computer science that can address this problem. In this case, our system is a search engine, and the objective is a comprehensive performance review of various finding algorithms that incorporate different system behaviors.

Algorithm	**Variation1**	**Variation 2**	**Time performance for comparison algorithm**	**Describe the results of the test**
Linear Search	1000 values	Random integer values - Unsorted	0.0144 ms	When the size of the dataset increased, so did the time.
	10000 values	Random integer values - Unsorted	0.0485 ms	
	100000 values	Random integer values - Unsorted	0.8039 ms	
	1000 values	Random integer values - Sorted	0.0144 ms	When the size of the dataset increased, so did the time.
	10000 values	Random integer values - Sorted	0.0553 ms	
	100000 values	Random integer values - Sorted	0.3924 ms	
Linear Search	1000 values	Random string values - Unsorted	0.0526 ms	When the size of the dataset increased, so did the time.
	10000 values	Random string values - Unsorted	0.4611 ms	
	100000 values	Random string values - Unsorted	0.6678 ms	
	1000 values	Random string values - Sorted	0.0639 ms	When the size of the dataset increased, so did the time.
	10000 values	Random string values - Sorted	0.5848 ms	
	100000 values	Random string values - Sorted	0.5893 ms	
Binary search	1000 values	Random integer values - Sorted	0.0159 ms	Searching time depends on the position of the searching value.
	10000 values	Random integer values - Sorted	0.0125 ms	
	100000 values	Random integer values - Sorted	0.0123 ms	
Binary search	1000 values	Random string values - Sorted	0.0176 ms	Strings require more time to run this algorithm than integers.
	10000 values	Random string values - Sorted	0.0145 ms	
	100000 values	Random string values - Sorted	0.0122 ms	
Jump search	1000 values	Random integer values - Sorted	0.037 ms	Strings require more time to run this algorithm than integers.
	10000 values	Random integer values - Sorted	0.0154 ms	

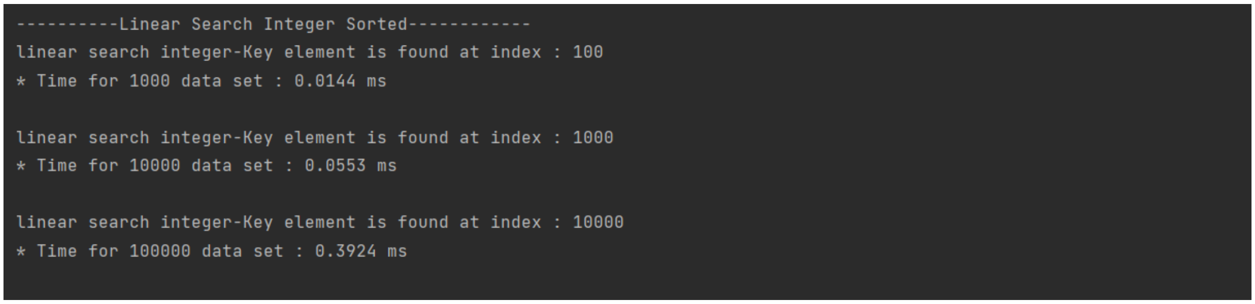
	100000 values	Random integer values - Sorted	0.0132 ms	
Jump search	1000 values	Random string values - Sorted	0.0441 ms	Strings require more time to run this algorithm than integers.
	10000 values	Random string values - Sorted	0.0111 ms	
	100000 values	Random string values - Sorted	2.1786 ms	
Exponential search	1000 values	Random integer values - Sorted	0.0127 ms	Strings require more time to run this algorithm than integers and when the size of the dataset increased, so did the time.
	10000 values	Random integer values - Sorted	0.0078 ms	
	100000 values	Random integer values - Sorted	0.0063 ms	
Exponential search	1000 values	Random string values - Sorted	0.028 ms	Strings require more time to run this algorithm than integers and when the size of the dataset increased, so did the time.
	10000 values	Random string values - Sorted	0.0094 ms	
	100000 values	Random string values - Sorted	0.0074 ms	
Interpolation search	1000 values	Random integer values - Sorted	0.1098 ms	The size of the dataset has no effect on the time.
	10000 values	Random integer values - Sorted	0.044 ms	
	100000 values	Random integer values - Sorted	0.0088 ms	

Table 1 - Variation Category & Description



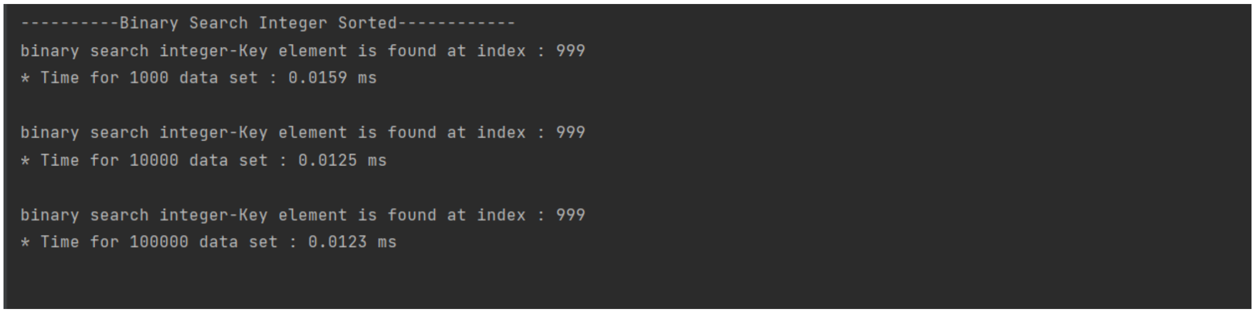
```
----------Linear Search Integer Unsorted-----------
linear search integer-Key element is found at index : 100
* Time for 1000 data set : 0.0144 ms

linear search integer-Key element is found at index : 1000
* Time for 10000 data set : 0.0485 ms

linear search integer-Key element is found at index : 10000
* Time for 100000 data set : 0.8039 ms
```

Figure 1-Linear search unsorted integer



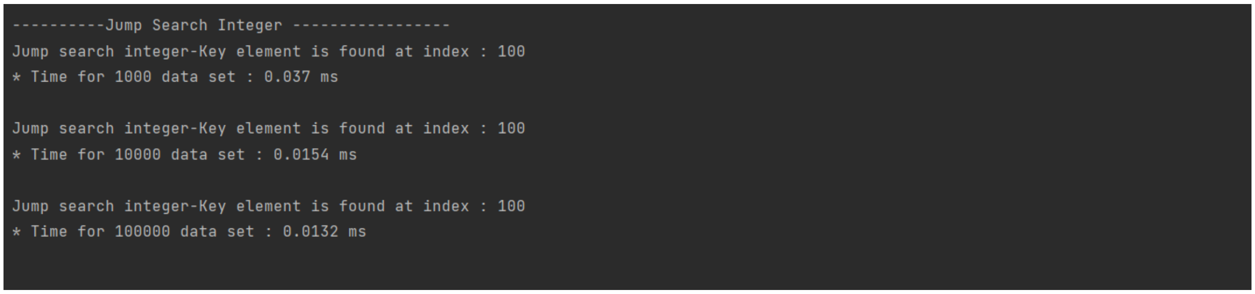
```
----------Linear Search Integer Sorted------------
linear search integer-Key element is found at index : 100
* Time for 1000 data set : 0.0144 ms

linear search integer-Key element is found at index : 1000
* Time for 10000 data set : 0.0553 ms

linear search integer-Key element is found at index : 10000
* Time for 100000 data set : 0.3924 ms
```

Figure 2-Linear search sorted integer



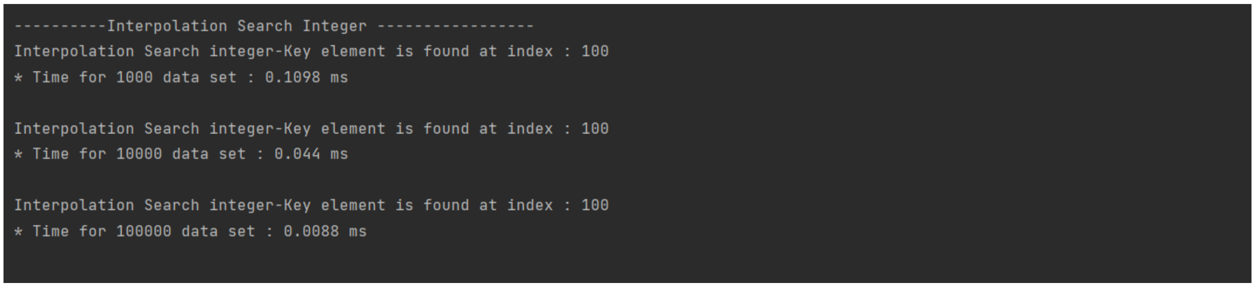
```
----------Binary Search Integer Sorted------------
binary search integer-Key element is found at index : 999
* Time for 1000 data set : 0.0159 ms

binary search integer-Key element is found at index : 999
* Time for 10000 data set : 0.0125 ms

binary search integer-Key element is found at index : 999
* Time for 100000 data set : 0.0123 ms
```

Figure 3- Binary search integer sorted



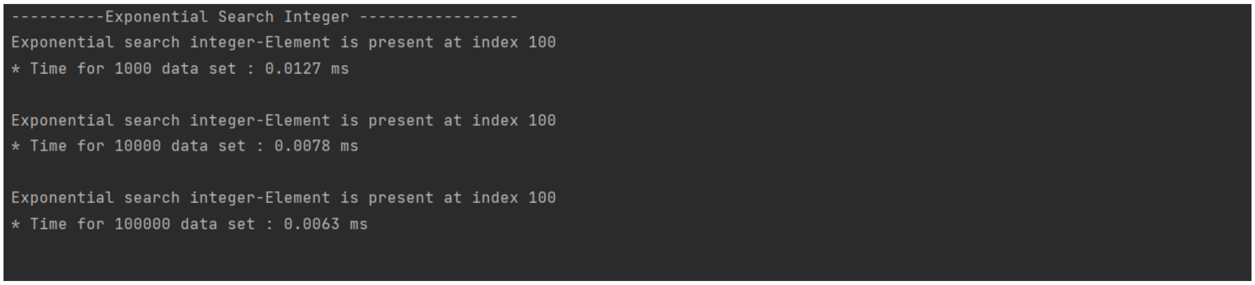
```
----------Jump Search Integer -----------------
Jump search integer-Key element is found at index : 100
* Time for 1000 data set : 0.037 ms

Jump search integer-Key element is found at index : 100
* Time for 10000 data set : 0.0154 ms

Jump search integer-Key element is found at index : 100
* Time for 100000 data set : 0.0132 ms
```

Figure 4 - Jump search integer sorted



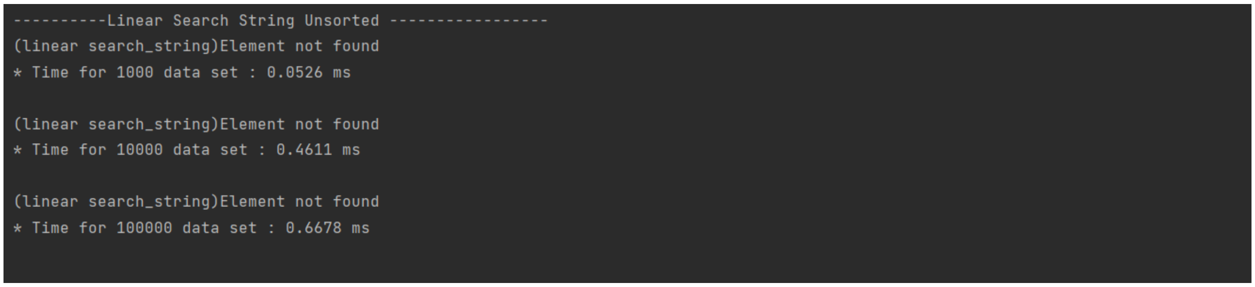
```
----------Interpolation Search Integer -----------------
Interpolation Search integer-Key element is found at index : 100
* Time for 1000 data set : 0.1098 ms

Interpolation Search integer-Key element is found at index : 100
* Time for 10000 data set : 0.044 ms

Interpolation Search integer-Key element is found at index : 100
* Time for 100000 data set : 0.0088 ms
```

Figure 5- Interpolation search Integer sorted



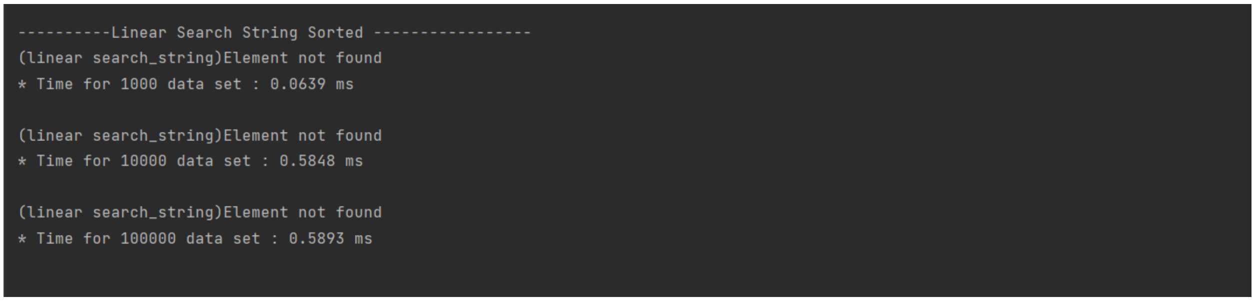
```
----------Exponential Search Integer -----------------
Exponential search integer-Element is present at index 100
* Time for 1000 data set : 0.0127 ms

Exponential search integer-Element is present at index 100
* Time for 10000 data set : 0.0078 ms

Exponential search integer-Element is present at index 100
* Time for 100000 data set : 0.0063 ms
```

Figure 6- Exponential search integer sorted



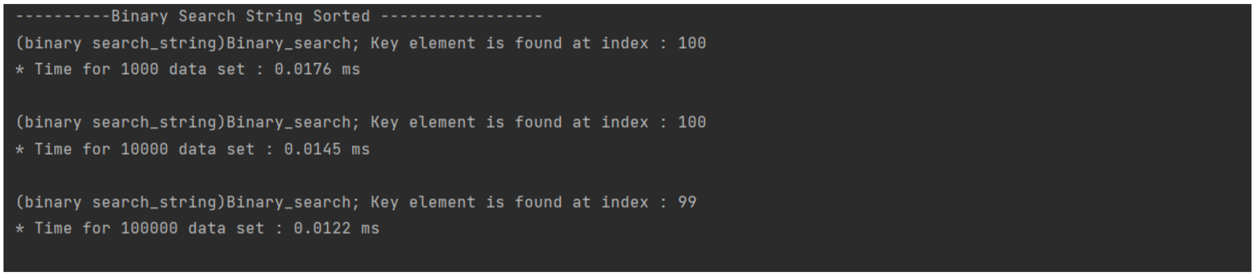
```
----------Linear Search String Unsorted -----------------
(linear search_string)Element not found
* Time for 1000 data set : 0.0526 ms

(linear search_string)Element not found
* Time for 10000 data set : 0.4611 ms

(linear search_string)Element not found
* Time for 100000 data set : 0.6678 ms
```

Figure 7-Linear search unsorted string



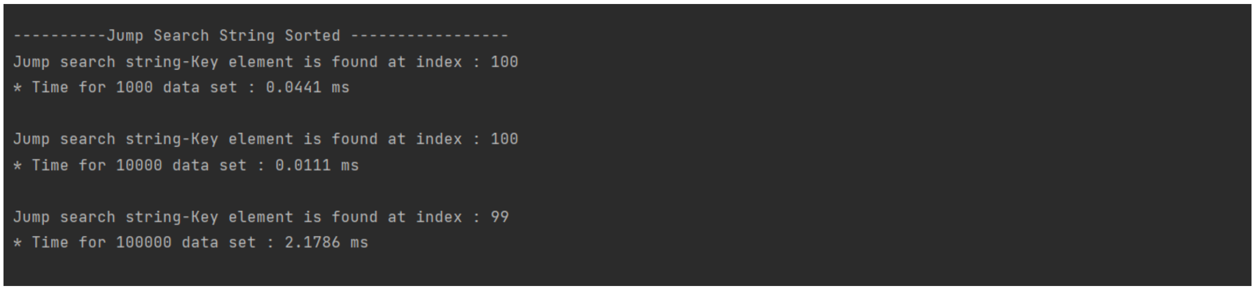
```
----------Linear Search String Sorted -----------------
(linear search_string)Element not found
* Time for 1000 data set : 0.0639 ms

(linear search_string)Element not found
* Time for 10000 data set : 0.5848 ms

(linear search_string)Element not found
* Time for 100000 data set : 0.5893 ms
```

Figure 8-Linear search sorted string



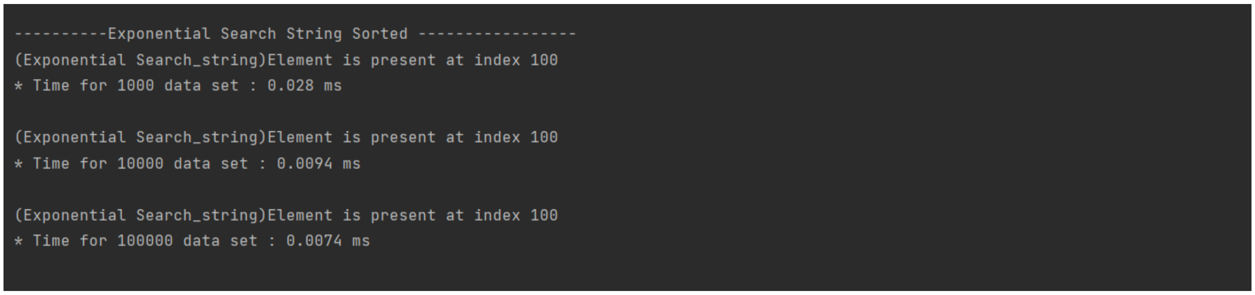
```
----------Binary Search String Sorted -----------------
(binary search_string)Binary_search; Key element is found at index : 100
* Time for 1000 data set : 0.0176 ms

(binary search_string)Binary_search; Key element is found at index : 100
* Time for 10000 data set : 0.0145 ms

(binary search_string)Binary_search; Key element is found at index : 99
* Time for 100000 data set : 0.0122 ms
```

Figure 9-Binary search unsorted string



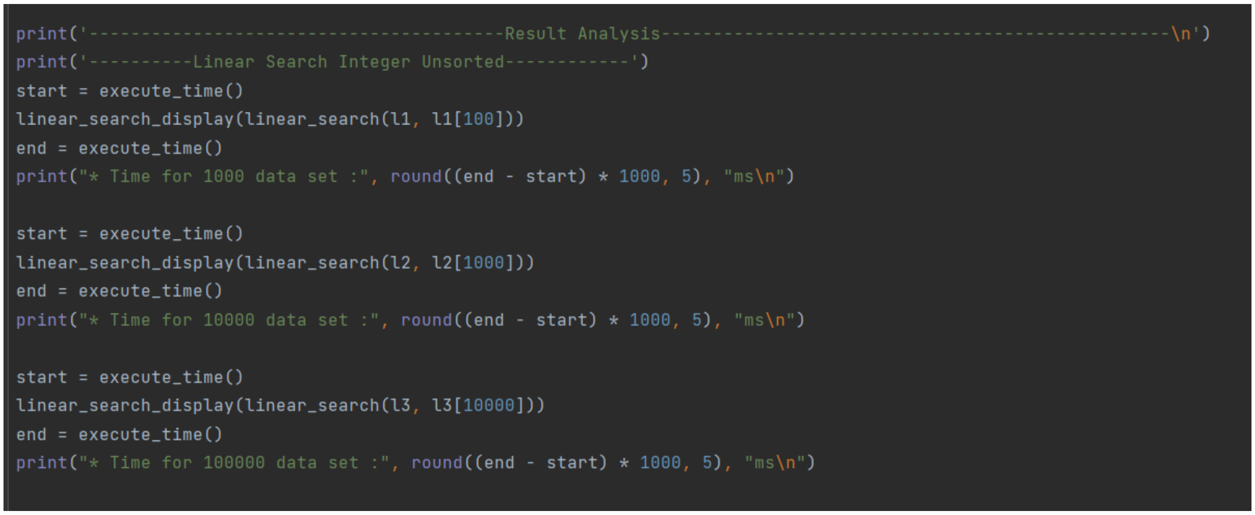
```
----------Jump Search String Sorted -----------------
Jump search string-Key element is found at index : 100
* Time for 1000 data set : 0.0441 ms

Jump search string-Key element is found at index : 100
* Time for 10000 data set : 0.0111 ms

Jump search string-Key element is found at index : 99
* Time for 100000 data set : 2.1786 ms
```

Figure 10-Jump search unsorted string

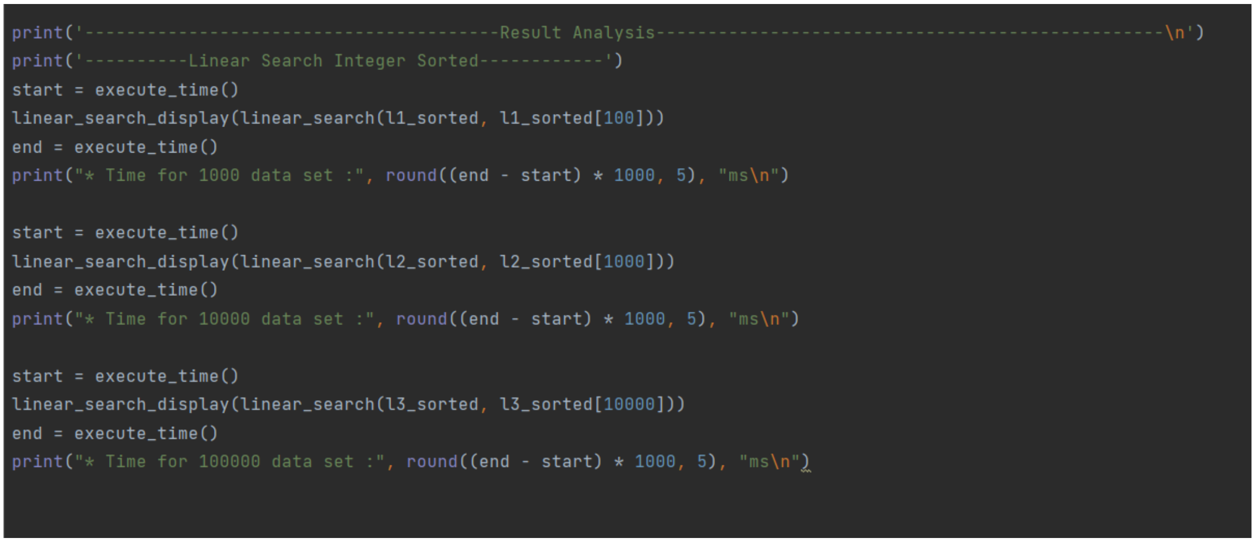


```
----------Exponential Search String Sorted -----------------
(Exponential Search_string)Element is present at index 100
* Time for 1000 data set : 0.028 ms

(Exponential Search_string)Element is present at index 100
* Time for 10000 data set : 0.0094 ms

(Exponential Search_string)Element is present at index 100
* Time for 100000 data set : 0.0074 ms
```

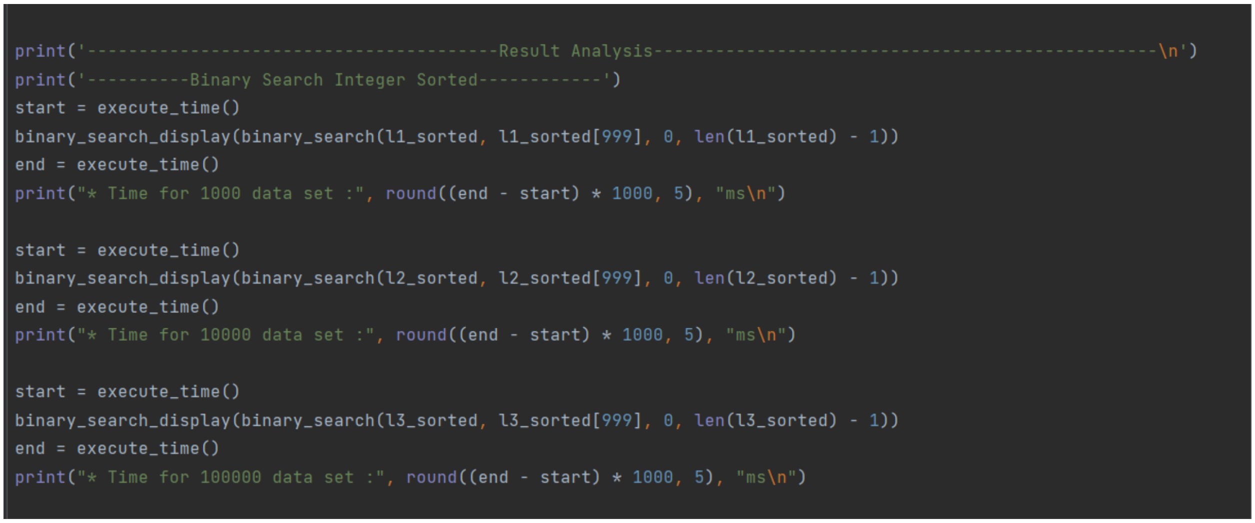
Figure 11-Exponential search unsorted string



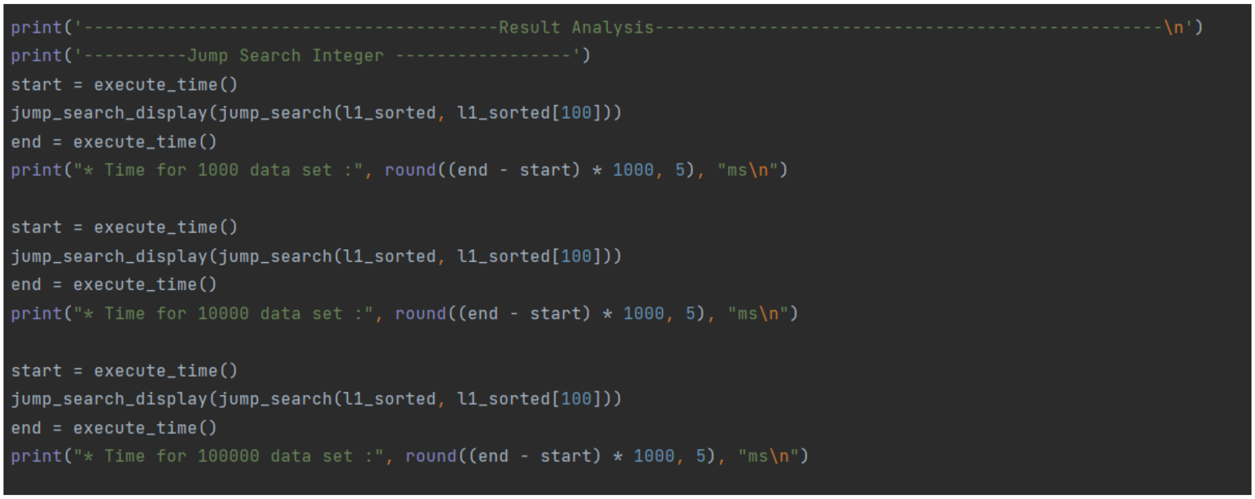
```
print('-----------------------------------------Result Analysis--------------------------------------------------\n')
print('----------Linear Search Integer Unsorted-----------')
start = execute_time()
linear_search_display(linear_search(l1, l1[100]))
end = execute_time()
print("* Time for 1000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
linear_search_display(linear_search(l2, l2[1000]))
end = execute_time()
print("* Time for 10000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
linear_search_display(linear_search(l3, l3[10000]))
end = execute_time()
print("* Time for 100000 data set :", round((end - start) * 1000, 5), "ms\n")
```



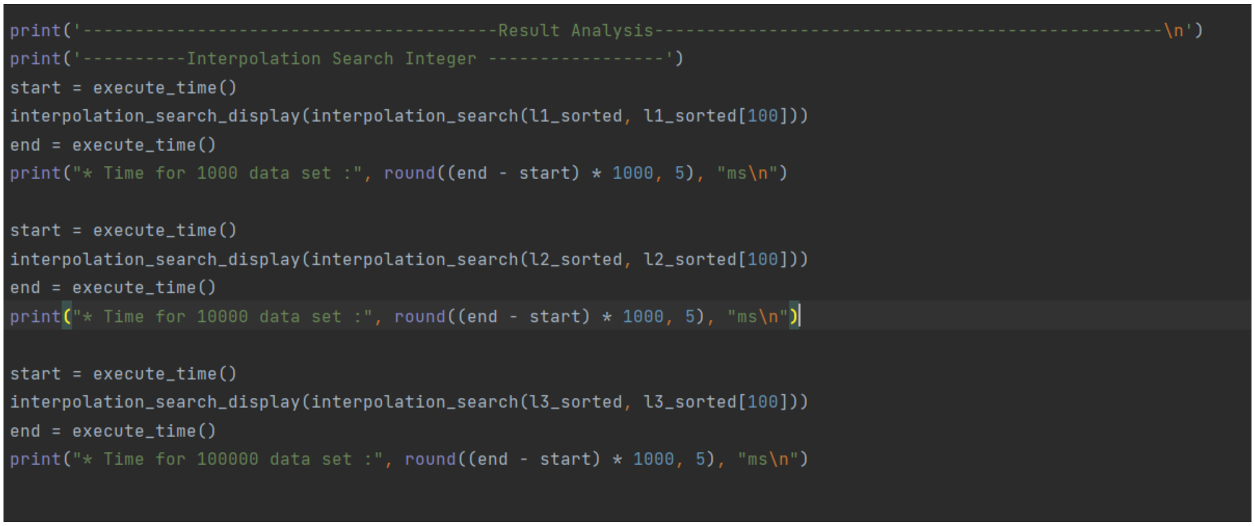
```
print('----------------------------------------Result Analysis-------------------------------------------------\n')
print('----------Linear Search Integer Sorted-----------')
start = execute_time()
linear_search_display(linear_search(l1_sorted, l1_sorted[100]))
end = execute_time()
print("* Time for 1000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
linear_search_display(linear_search(l2_sorted, l2_sorted[1000]))
end = execute_time()
print("* Time for 10000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
linear_search_display(linear_search(l3_sorted, l3_sorted[10000]))
end = execute_time()
print("* Time for 100000 data set :", round((end - start) * 1000, 5), "ms\n")
```



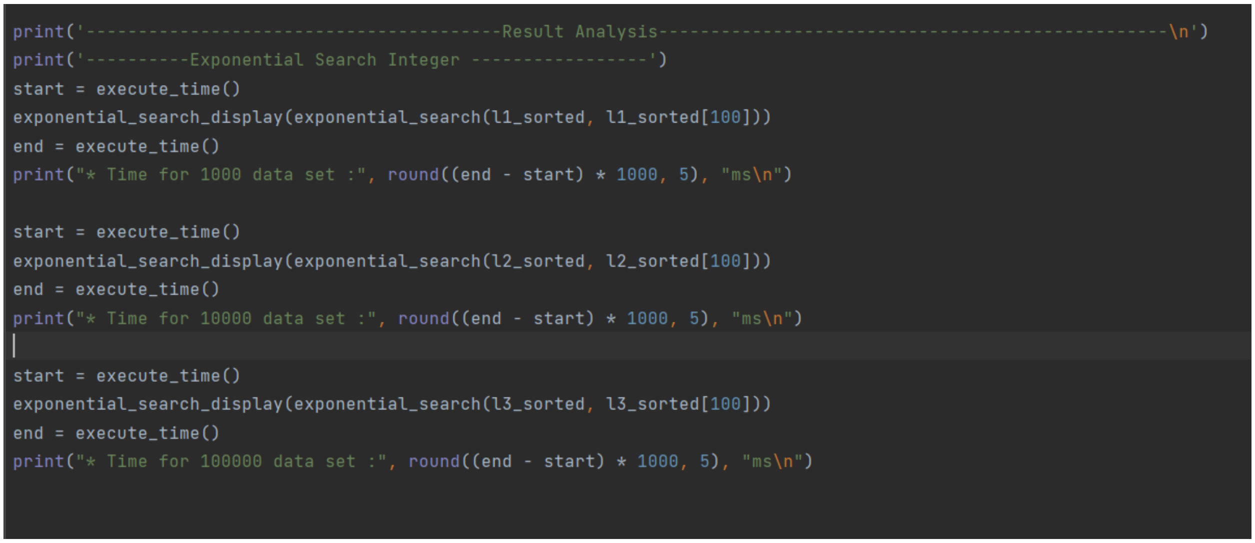
```
print('----------------------------------------Result Analysis-------------------------------------------------\n')
print('----------Binary Search Integer Sorted-----------')
start = execute_time()
binary_search_display(binary_search(l1_sorted, l1_sorted[999], 0, len(l1_sorted) - 1))
end = execute_time()
print("* Time for 1000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
binary_search_display(binary_search(l2_sorted, l2_sorted[999], 0, len(l2_sorted) - 1))
end = execute_time()
print("* Time for 10000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
binary_search_display(binary_search(l3_sorted, l3_sorted[999], 0, len(l3_sorted) - 1))
end = execute_time()
print("* Time for 100000 data set :", round((end - start) * 1000, 5), "ms\n")
```



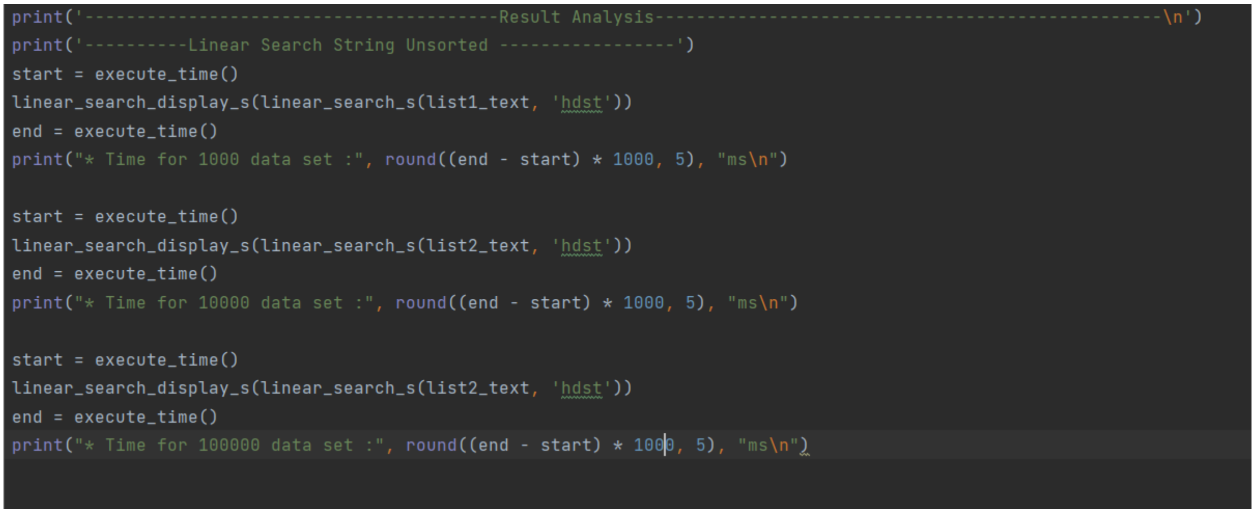
```
print('----------------------------------------Result Analysis-------------------------------------------------\n')
print('----------Jump Search Integer -----------------')
start = execute_time()
jump_search_display(jump_search(l1_sorted, l1_sorted[100]))
end = execute_time()
print("* Time for 1000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
jump_search_display(jump_search(l1_sorted, l1_sorted[100]))
end = execute_time()
print("* Time for 10000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
jump_search_display(jump_search(l1_sorted, l1_sorted[100]))
end = execute_time()
print("* Time for 100000 data set :", round((end - start) * 1000, 5), "ms\n")
```



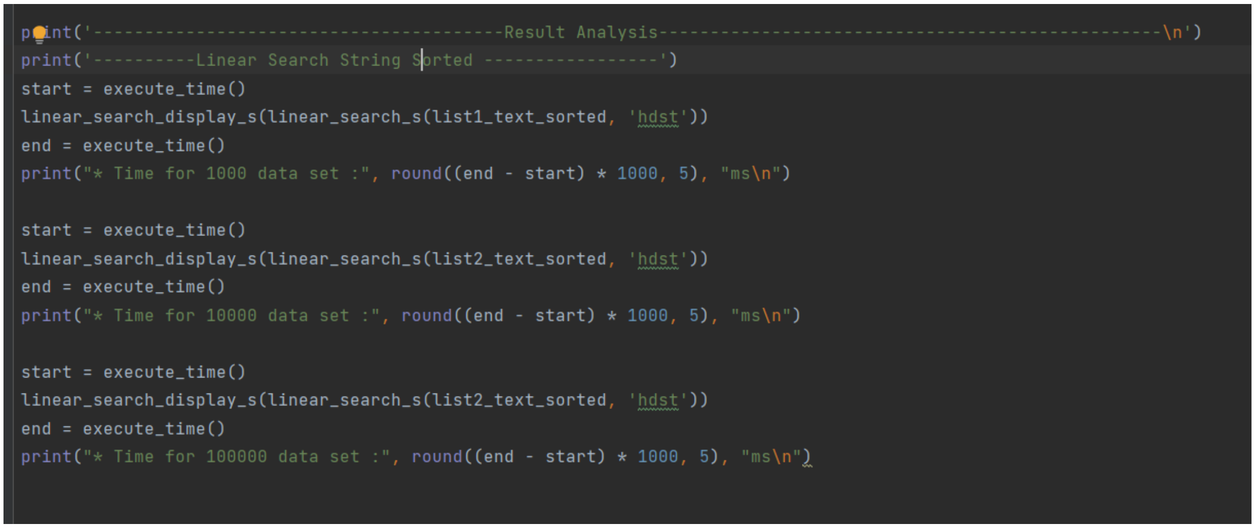
```
print('----------------------------------------Result Analysis-------------------------------------------------\n')
print('----------Interpolation Search Integer -----------------')
start = execute_time()
interpolation_search_display(interpolation_search(l1_sorted, l1_sorted[100]))
end = execute_time()
print("* Time for 1000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
interpolation_search_display(interpolation_search(l2_sorted, l2_sorted[100]))
end = execute_time()
print("* Time for 10000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
interpolation_search_display(interpolation_search(l3_sorted, l3_sorted[100]))
end = execute_time()
print("* Time for 100000 data set :", round((end - start) * 1000, 5), "ms\n")
```



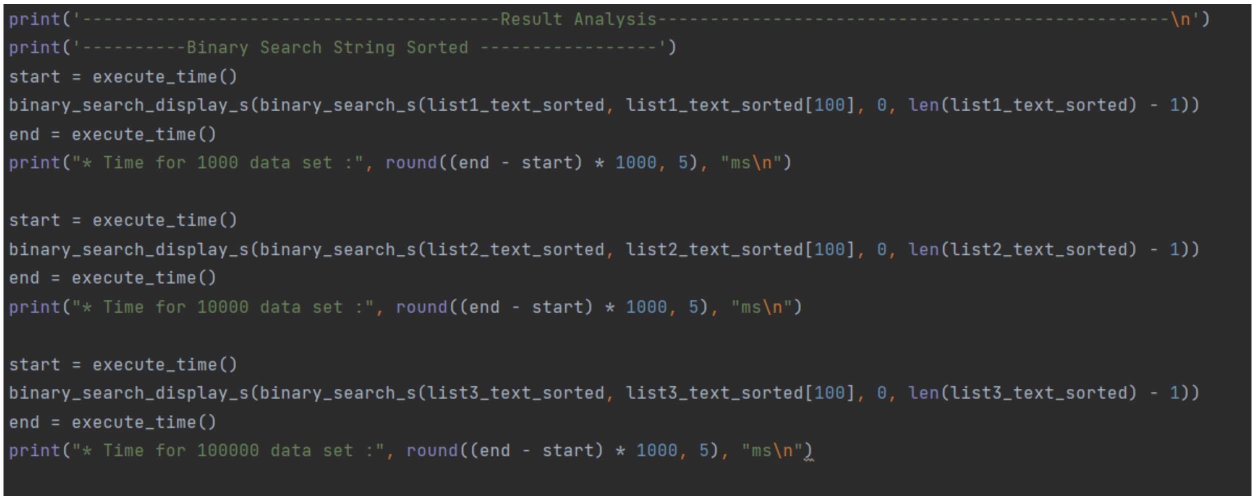
```
print('----------------------------------------Result Analysis-------------------------------------------------\n')
print('----------Exponential Search Integer -----------------')
start = execute_time()
exponential_search_display(exponential_search(l1_sorted, l1_sorted[100]))
end = execute_time()
print("* Time for 1000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
exponential_search_display(exponential_search(l2_sorted, l2_sorted[100]))
end = execute_time()
print("* Time for 10000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
exponential_search_display(exponential_search(l3_sorted, l3_sorted[100]))
end = execute_time()
print("* Time for 100000 data set :", round((end - start) * 1000, 5), "ms\n")
```



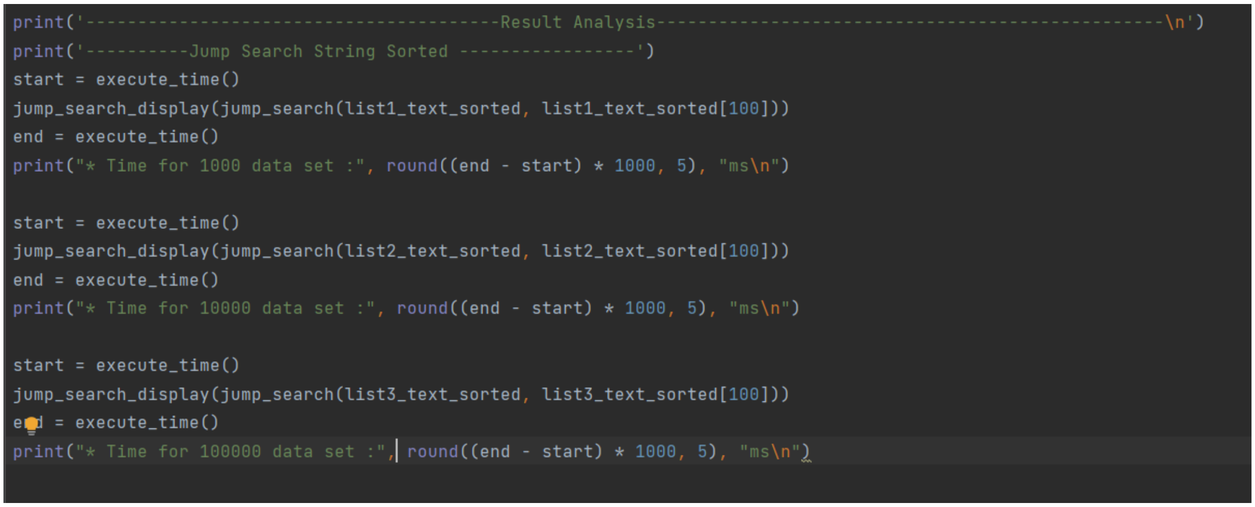
```
print('----------------------------------------Result Analysis-------------------------------------------------\n')
print('----------Linear Search String Unsorted -----------------')
start = execute_time()
linear_search_display_s(linear_search_s(list1_text, 'hdst'))
end = execute_time()
print("* Time for 1000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
linear_search_display_s(linear_search_s(list2_text, 'hdst'))
end = execute_time()
print("* Time for 10000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
linear_search_display_s(linear_search_s(list2_text, 'hdst'))
end = execute_time()
print("* Time for 100000 data set :", round((end - start) * 1000, 5), "ms\n")
```



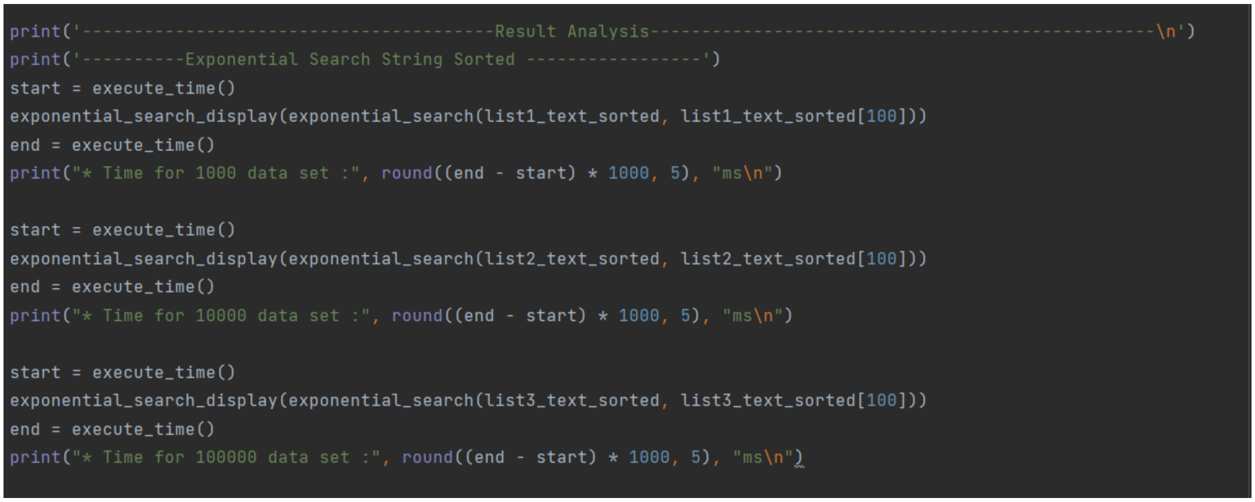
```
print('----------------------------------------Result Analysis-------------------------------------------------\n')
print('----------Linear Search String Sorted -----------------')
start = execute_time()
linear_search_display_s(linear_search_s(list1_text_sorted, 'hdst'))
end = execute_time()
print("* Time for 1000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
linear_search_display_s(linear_search_s(list2_text_sorted, 'hdst'))
end = execute_time()
print("* Time for 10000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
linear_search_display_s(linear_search_s(list2_text_sorted, 'hdst'))
end = execute_time()
print("* Time for 100000 data set :", round((end - start) * 1000, 5), "ms\n")
```



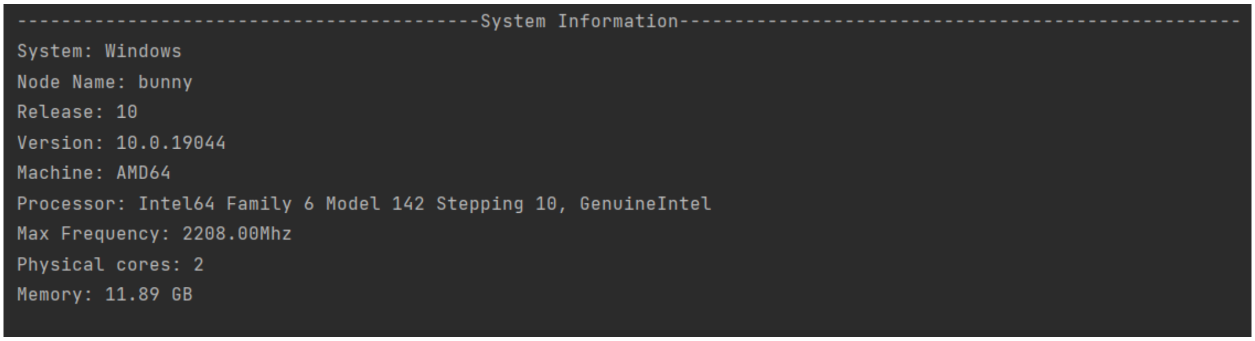
```
print('----------------------------------------Result Analysis-------------------------------------------------\n')
print('----------Binary Search String Sorted -----------------')
start = execute_time()
binary_search_display_s(binary_search_s(list1_text_sorted, list1_text_sorted[100], 0, len(list1_text_sorted) - 1))
end = execute_time()
print("* Time for 1000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
binary_search_display_s(binary_search_s(list2_text_sorted, list2_text_sorted[100], 0, len(list2_text_sorted) - 1))
end = execute_time()
print("* Time for 10000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
binary_search_display_s(binary_search_s(list3_text_sorted, list3_text_sorted[100], 0, len(list3_text_sorted) - 1))
end = execute_time()
print("* Time for 100000 data set :", round((end - start) * 1000, 5), "ms\n")
```



```
print('----------------------------------------Result Analysis-------------------------------------------------\n')
print('----------Jump Search String Sorted -----------------')
start = execute_time()
jump_search_display(jump_search(list1_text_sorted, list1_text_sorted[100]))
end = execute_time()
print("* Time for 1000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
jump_search_display(jump_search(list2_text_sorted, list2_text_sorted[100]))
end = execute_time()
print("* Time for 10000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
jump_search_display(jump_search(list3_text_sorted, list3_text_sorted[100]))
end = execute_time()
print("* Time for 100000 data set :", round((end - start) * 1000, 5), "ms\n")
```

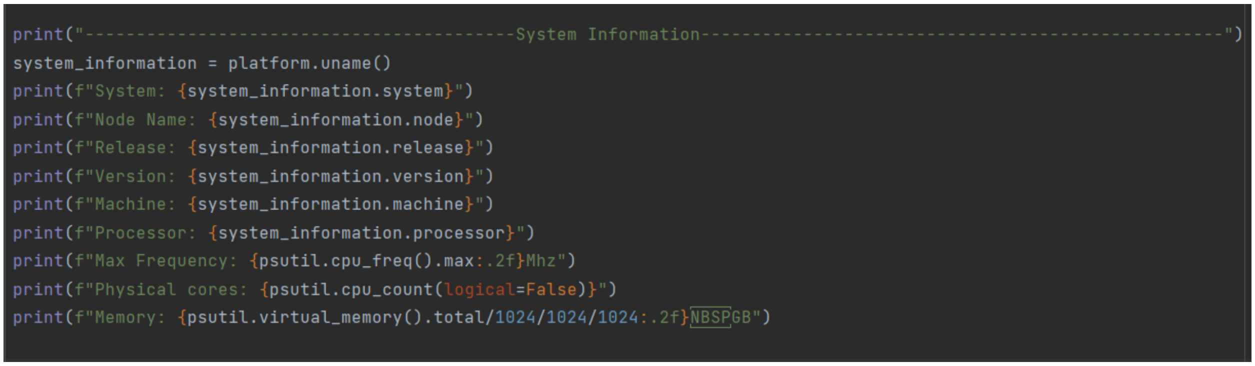


```
print('----------------------------------------Result Analysis-------------------------------------------------\n')
print('----------Exponential Search String Sorted -----------------')
start = execute_time()
exponential_search_display(exponential_search(list1_text_sorted, list1_text_sorted[100]))
end = execute_time()
print("* Time for 1000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
exponential_search_display(exponential_search(list2_text_sorted, list2_text_sorted[100]))
end = execute_time()
print("* Time for 10000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
exponential_search_display(exponential_search(list3_text_sorted, list3_text_sorted[100]))
end = execute_time()
print("* Time for 100000 data set :", round((end - start) * 1000, 5), "ms\n")
```



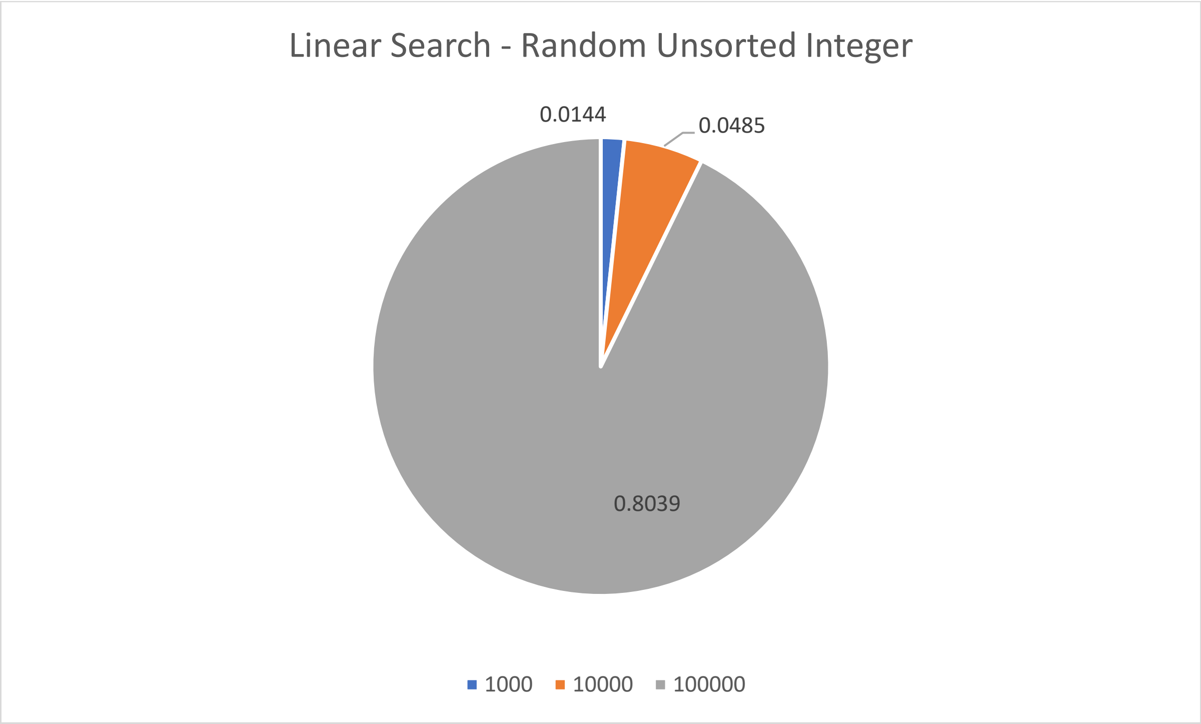
```
------------------------------------------System Information---------------------------------------------------
System: Windows
Node Name: bunny
Release: 10
Version: 10.0.19044
Machine: AMD64
Processor: Intel64 Family 6 Model 142 Stepping 10, GenuineIntel
Max Frequency: 2208.00Mhz
Physical cores: 2
Memory: 11.89 GB
```

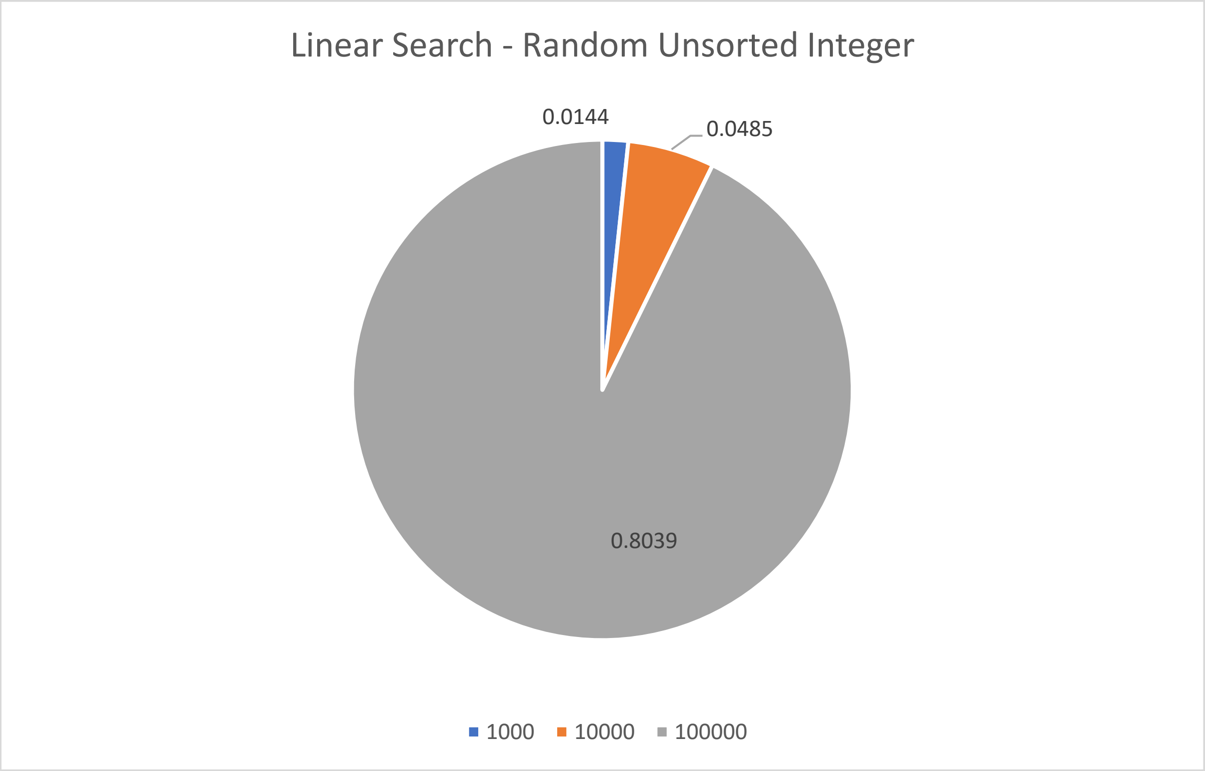
Figure 12-System information

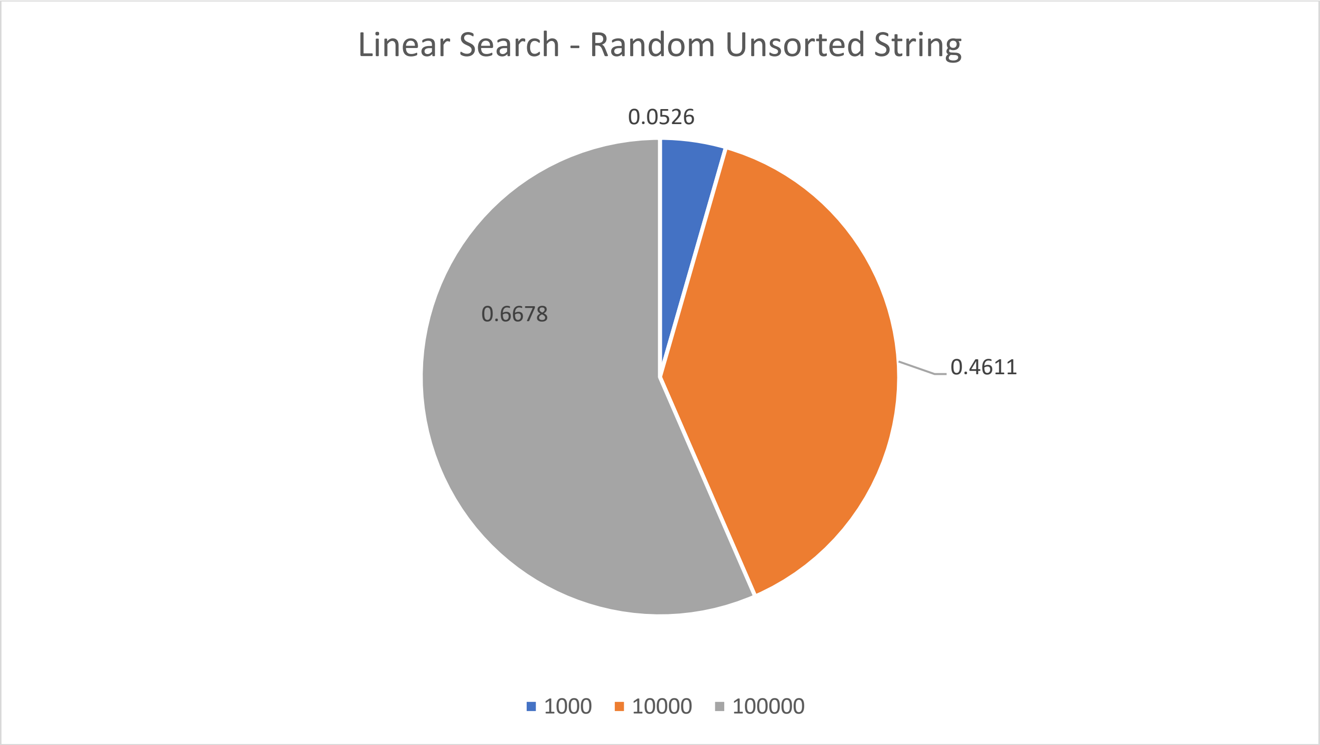


```
print("-------------------------------------------System Information----------------------------------------------------")
system_information = platform.uname()
print(f"System: {system_information.system}")
print(f"Node Name: {system_information.node}")
print(f"Release: {system_information.release}")
print(f"Version: {system_information.version}")
print(f"Machine: {system_information.machine}")
print(f"Processor: {system_information.processor}")
print(f"Max Frequency: {psutil.cpu_freq().max:.2f}Mhz")
print(f"Physical cores: {psutil.cpu_count(logical=False)}")
print(f"Memory: {psutil.virtual_memory().total/1024/1024/1024:.2f}NBSPGB")
```

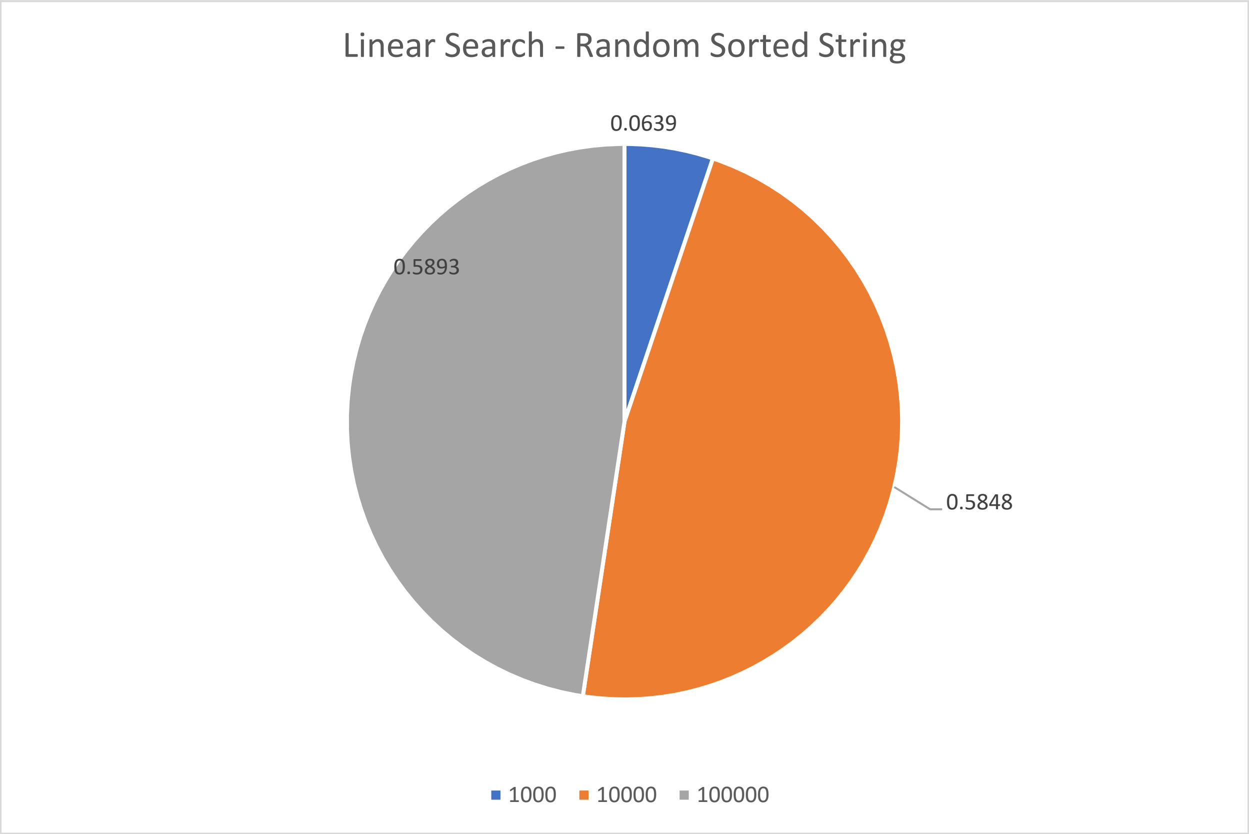
2. Graphs


Linear Search - Random Unsorted Integer
0.0144
0.0485
0.8039
1000
10000
100000

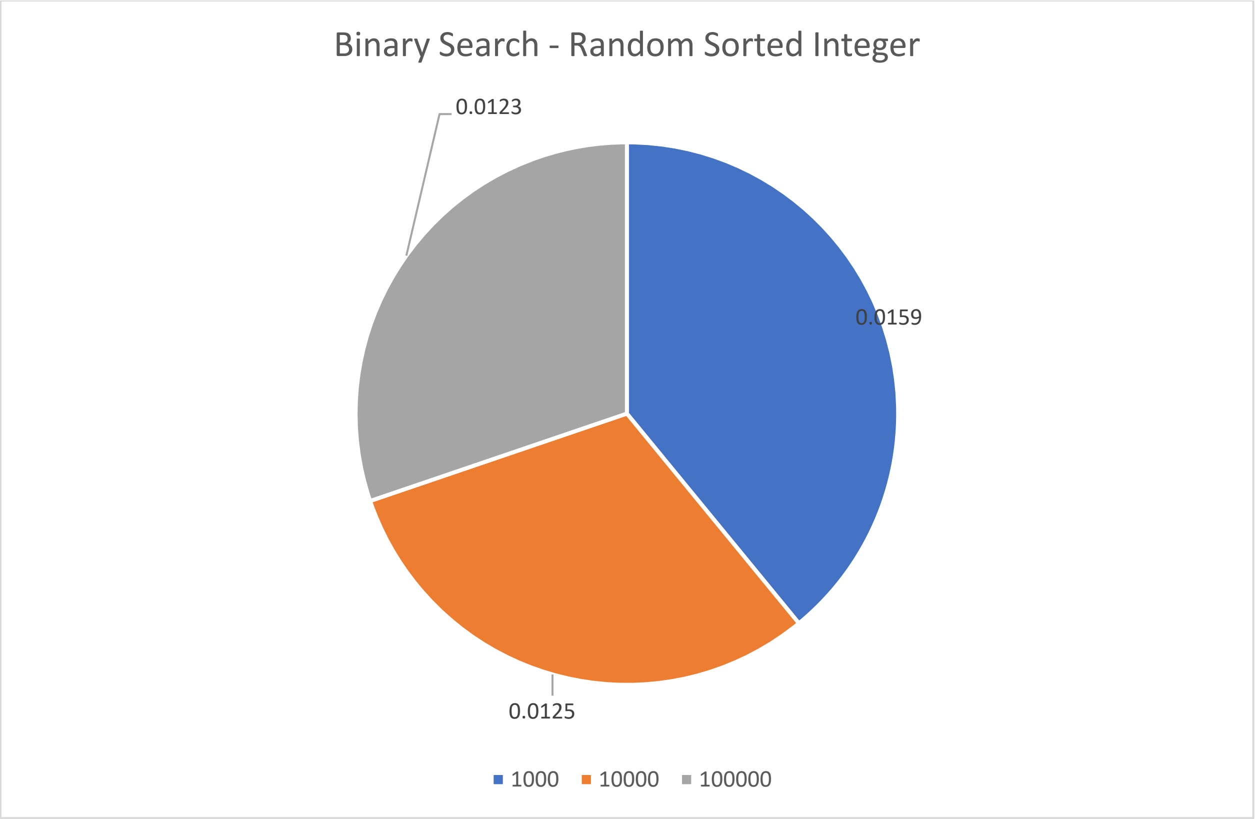

Linear Search - Random Unsorted Integer
0.0144
0.0485
0.8039
1000
10000
100000

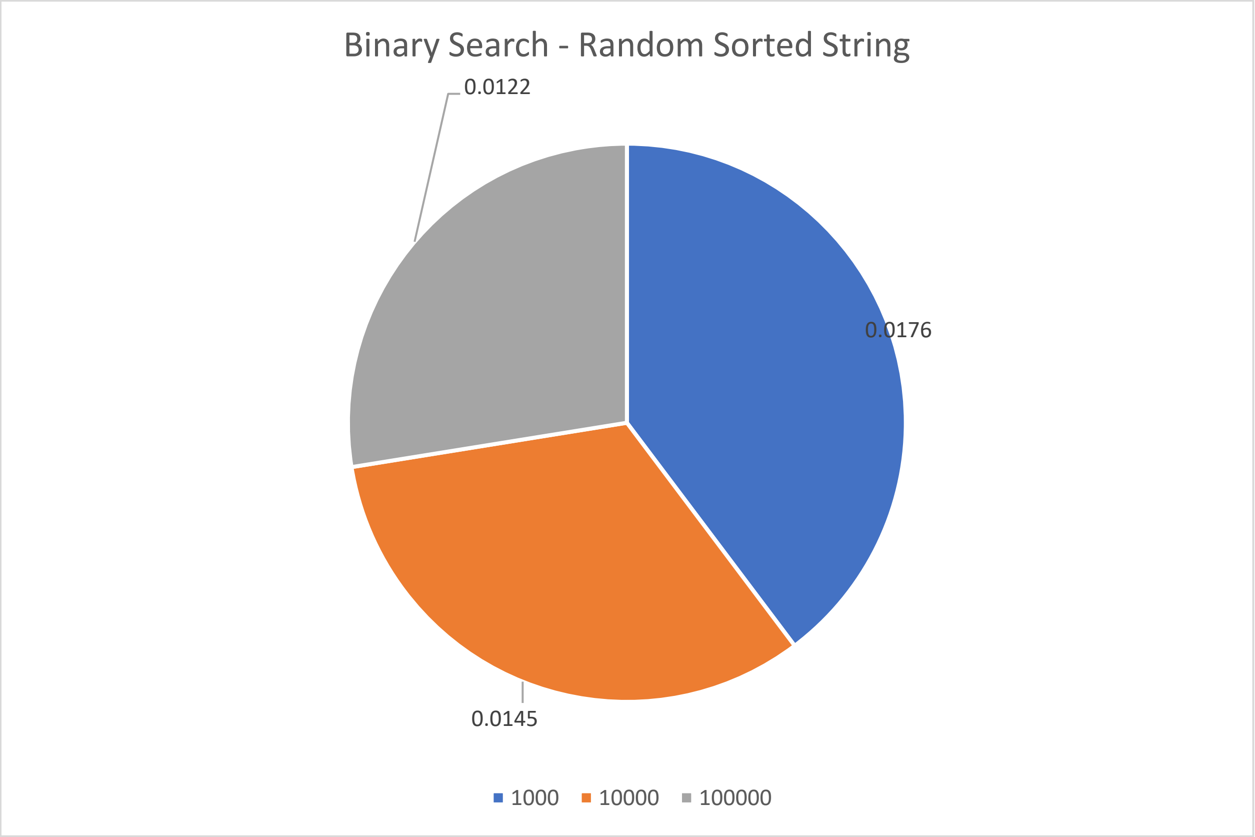


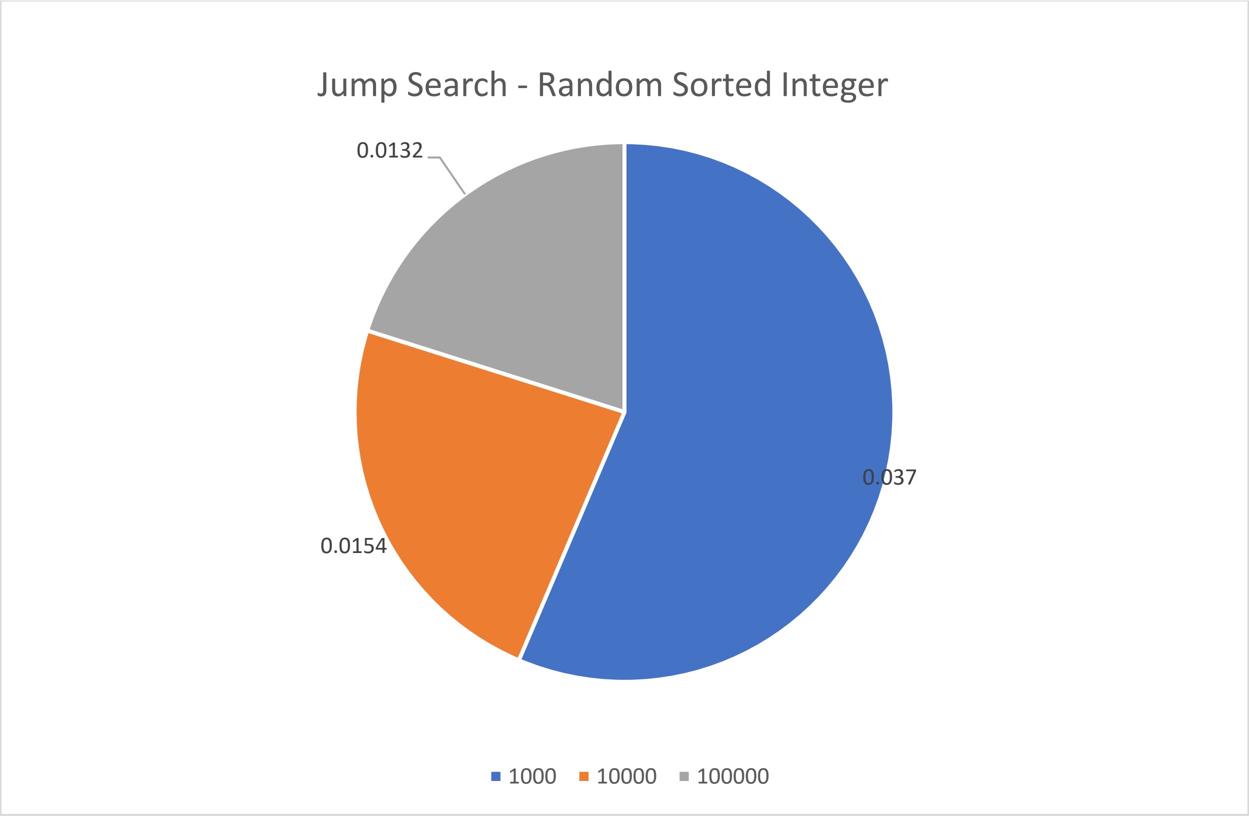
Linear Search - Random Unsorted String

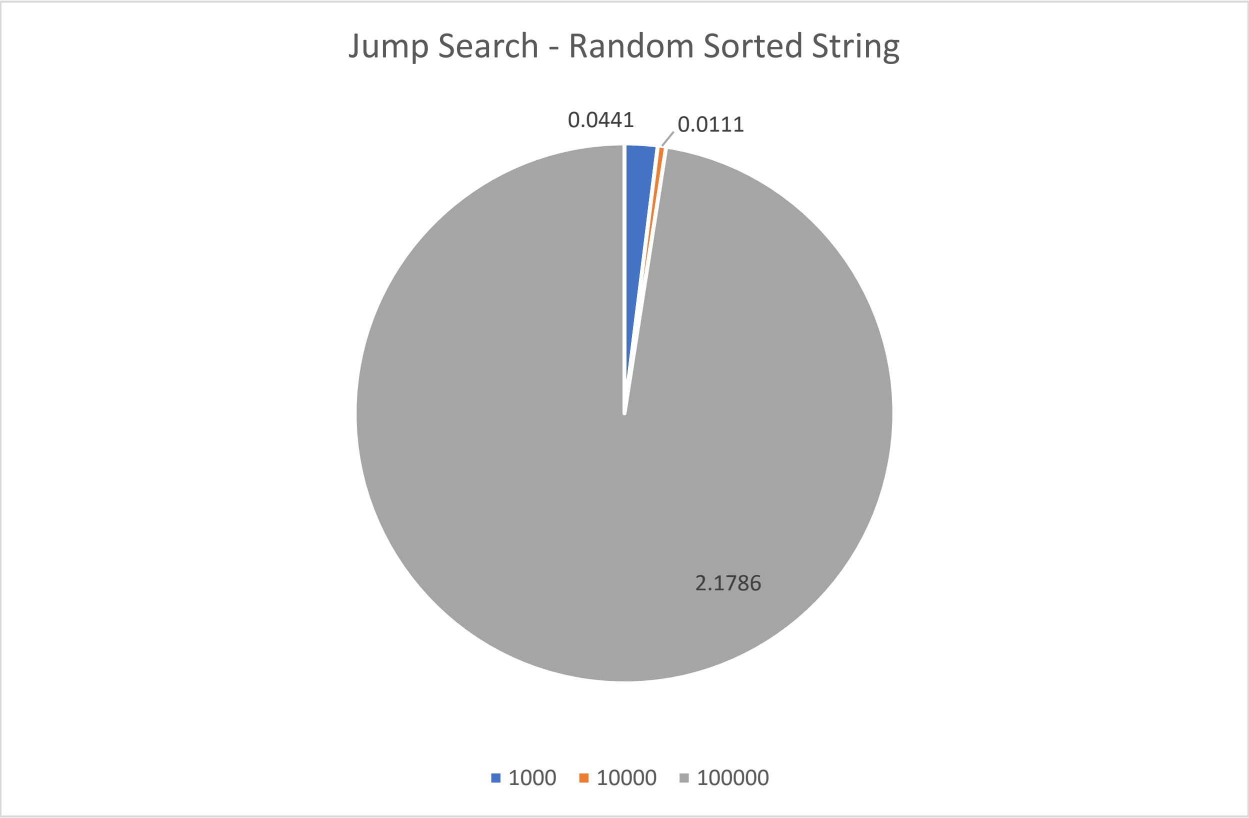


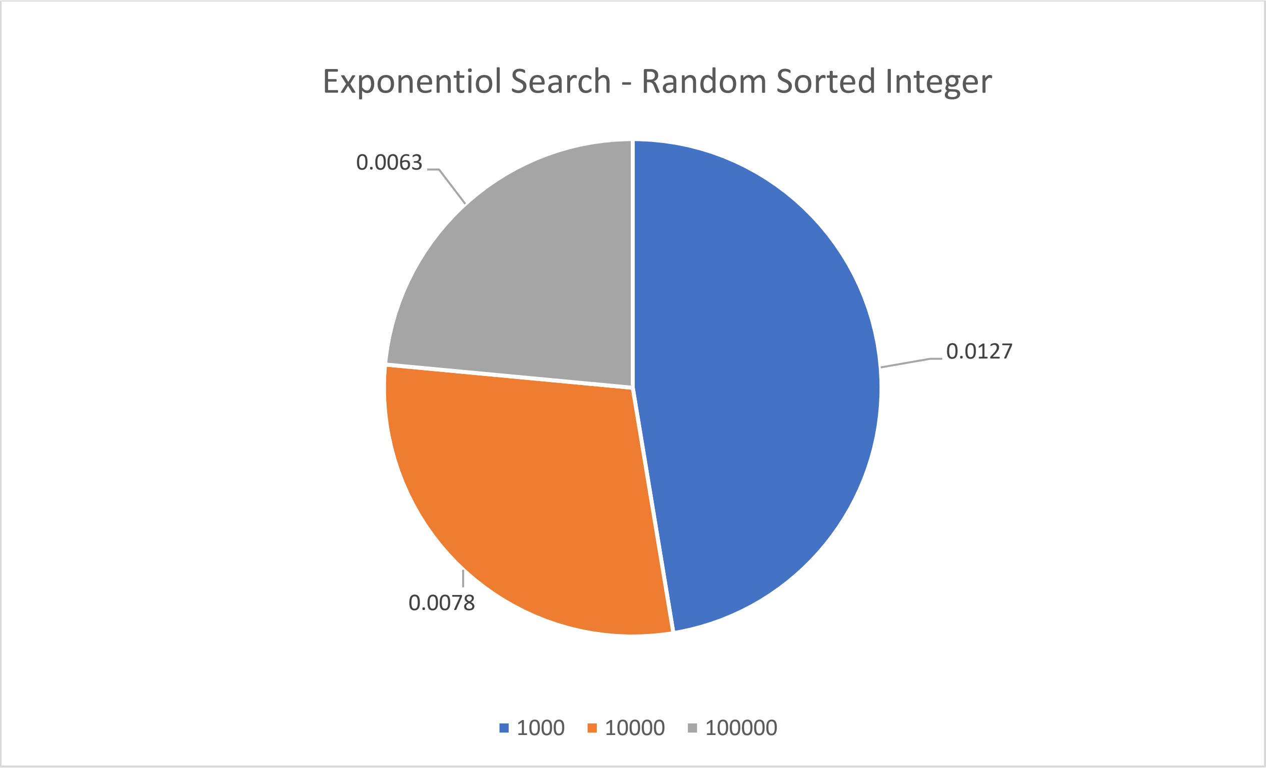
Linear Search - Random Sorted String

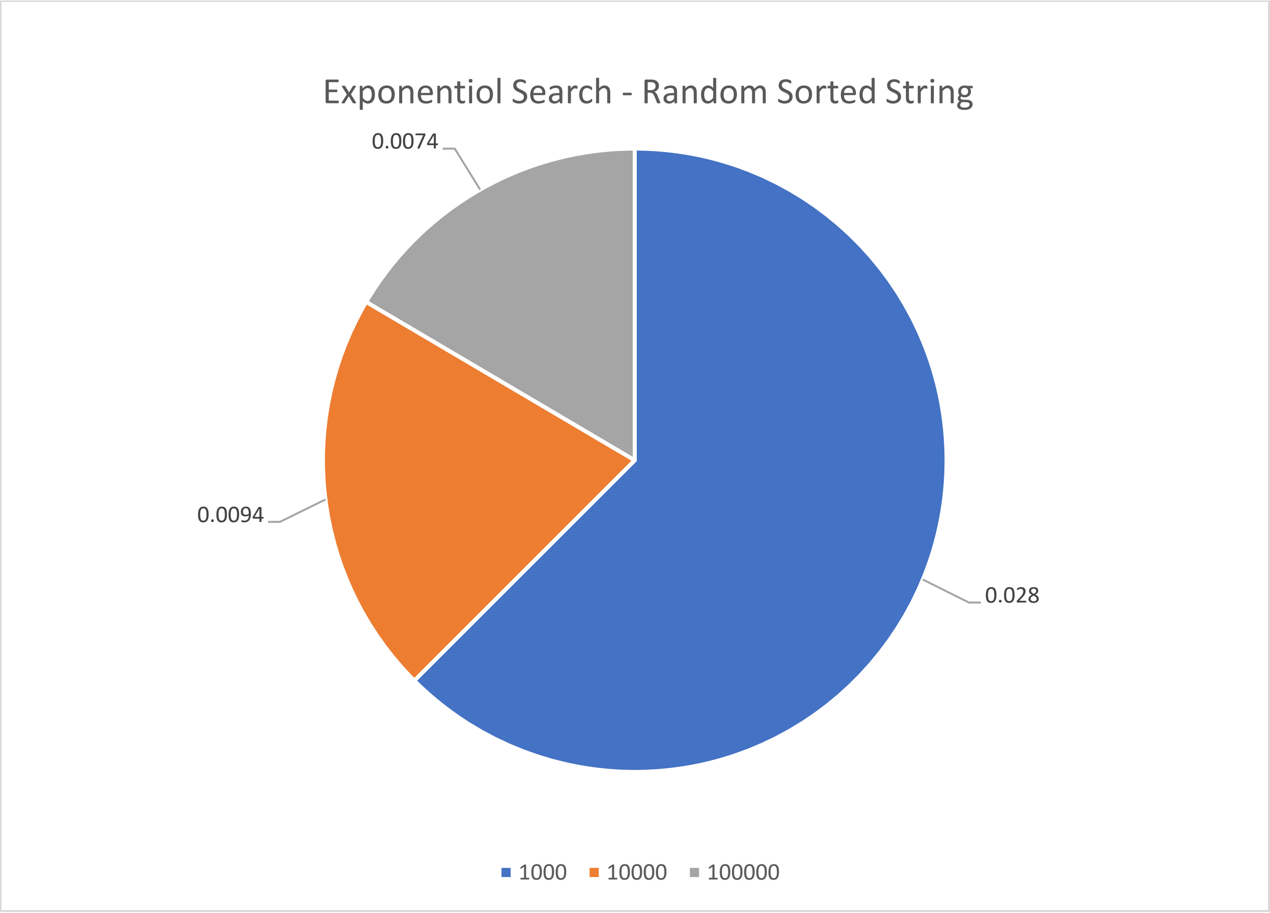

Binary Search - Random Sorted Integer
0.0123
0.0159
0.0125
1000
10000
100000

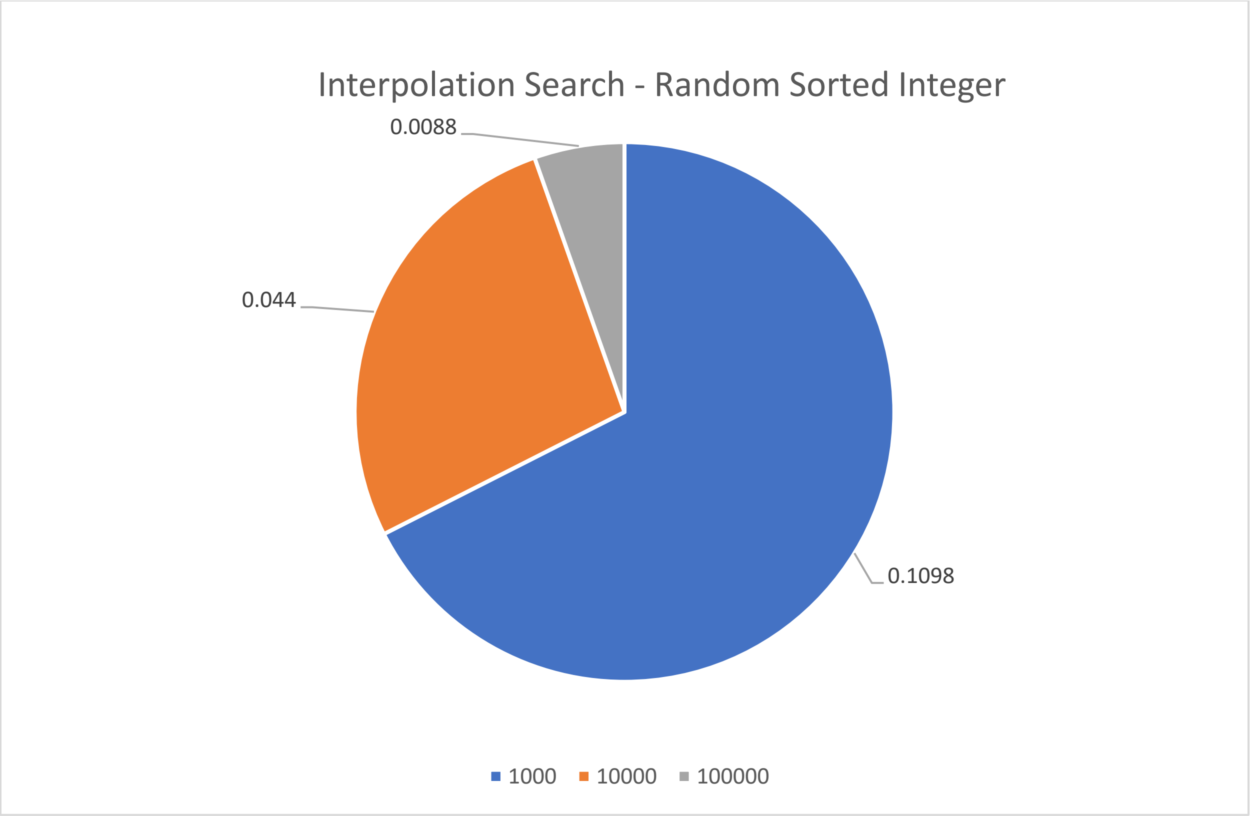

Binary Search - Random Sorted String
0.0122
0.0176
0.0145
1000
10000
100000

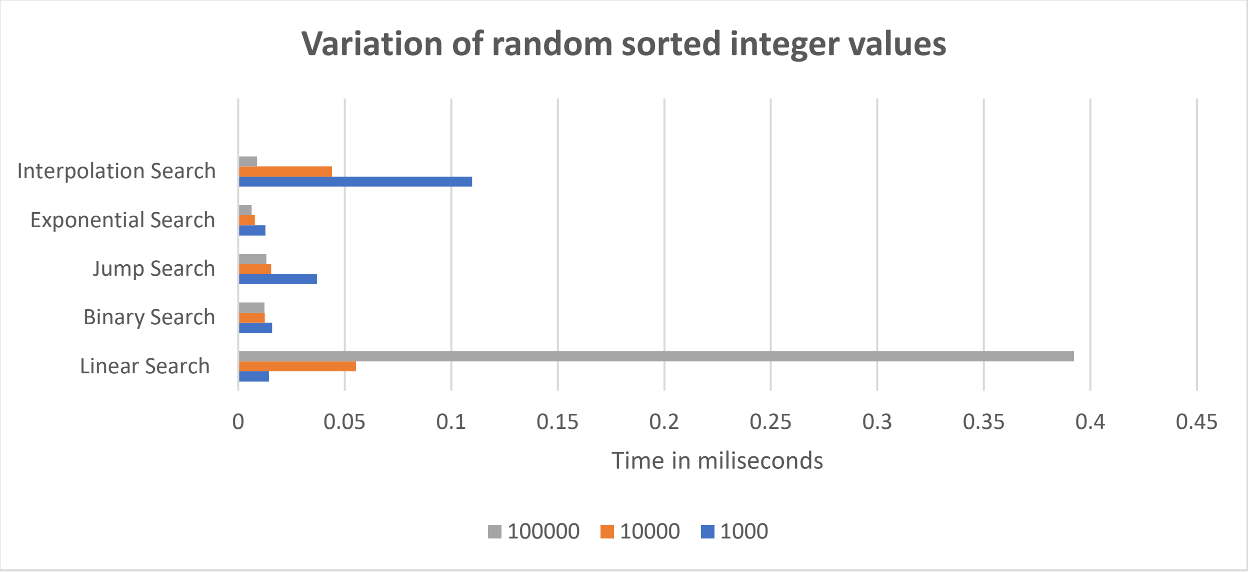

Jump Search - Random Sorted Integer
0.0132
0.037
0.0154
1000
10000
100000


Jump Search - Random Sorted String
0.0441
0.0111
2.1786
1000
10000
100000


Exponentiol Search - Random Sorted Integer
0.0063
0.0127
0.0078
1000
10000
100000


Exponentiol Search - Random Sorted String
0.0074
0.0094
0.028
1000
10000
100000


Interpolation Search - Random Sorted Integer
0.0088
0.044
0.1098
1000
10000
100000


Variation of random sorted integer values
Interpolation Search
Exponential Search
Jump Search
Binary Search
Linear Search
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
Time in miliseconds
100000
10000
1000

3. Conclusion

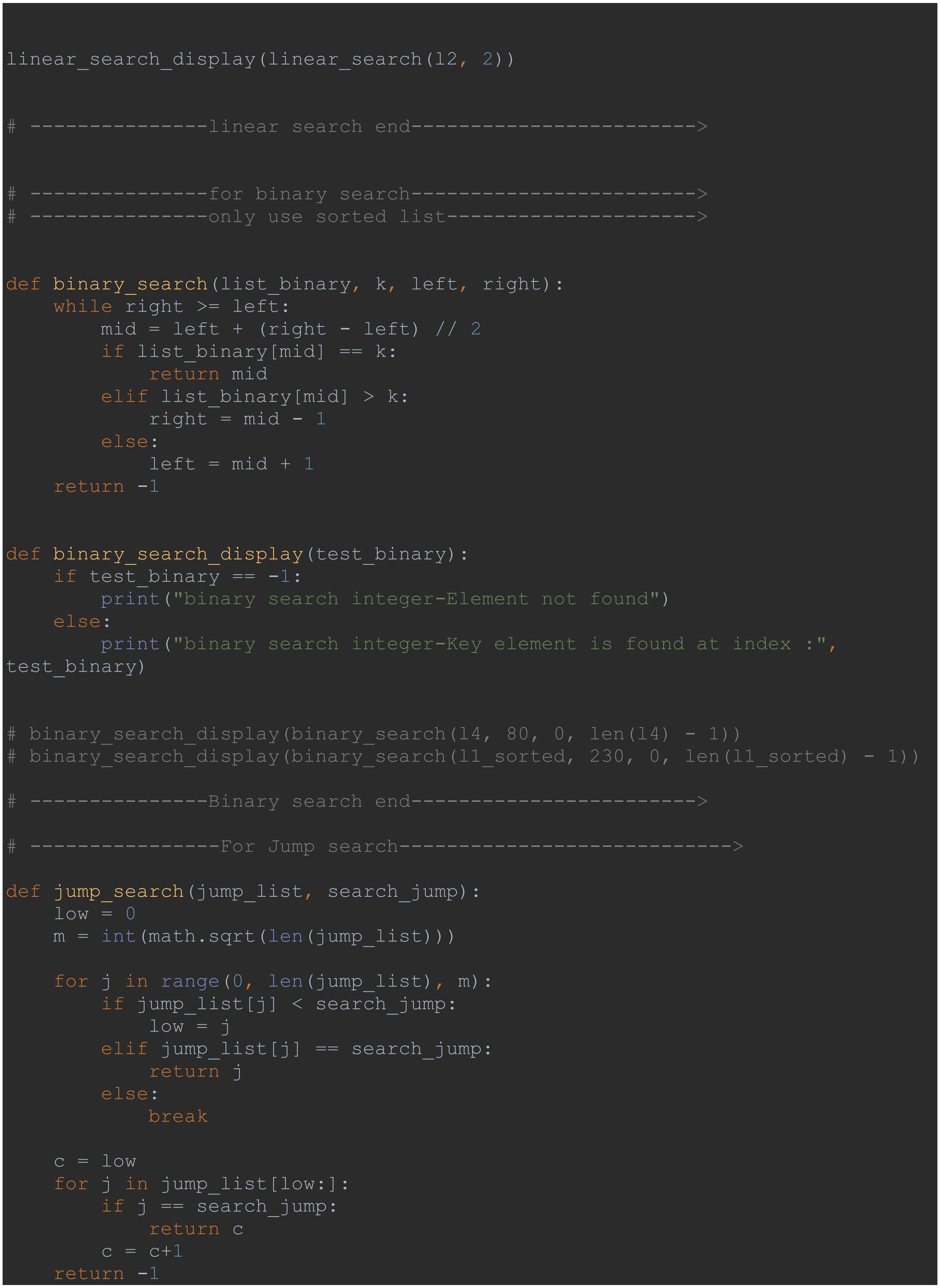
According to the above results, the best searching algorithm is the exponential search algorithm. As a result, it works with both string and integer data types, and it is an enhancement to the binary search algorithm. Exponential Binary Search is useful for unbounded searches with infinite list sizes.

Binary search is a more efficient search algorithm that relies on sorting the elements in a list. Linear search is the best that we can do when trying to find a value in an unsorted list. The major purpose of using binary search is that it does not scan every element in the list. Instead of scanning each element, it searches the first half of the list. As a result, a binary search takes less time to find an element than a linear search.

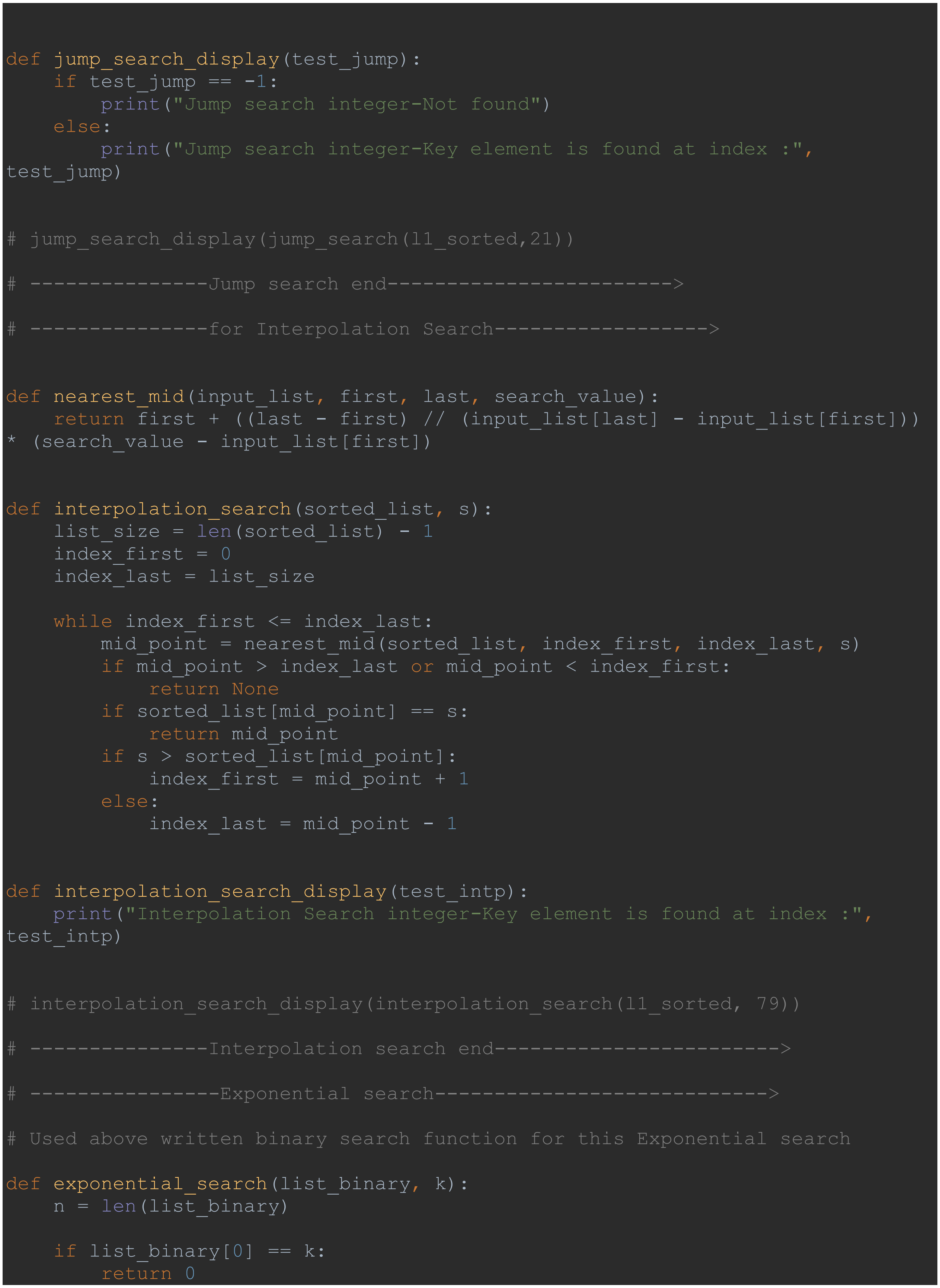
Appendix

Python code for Integers

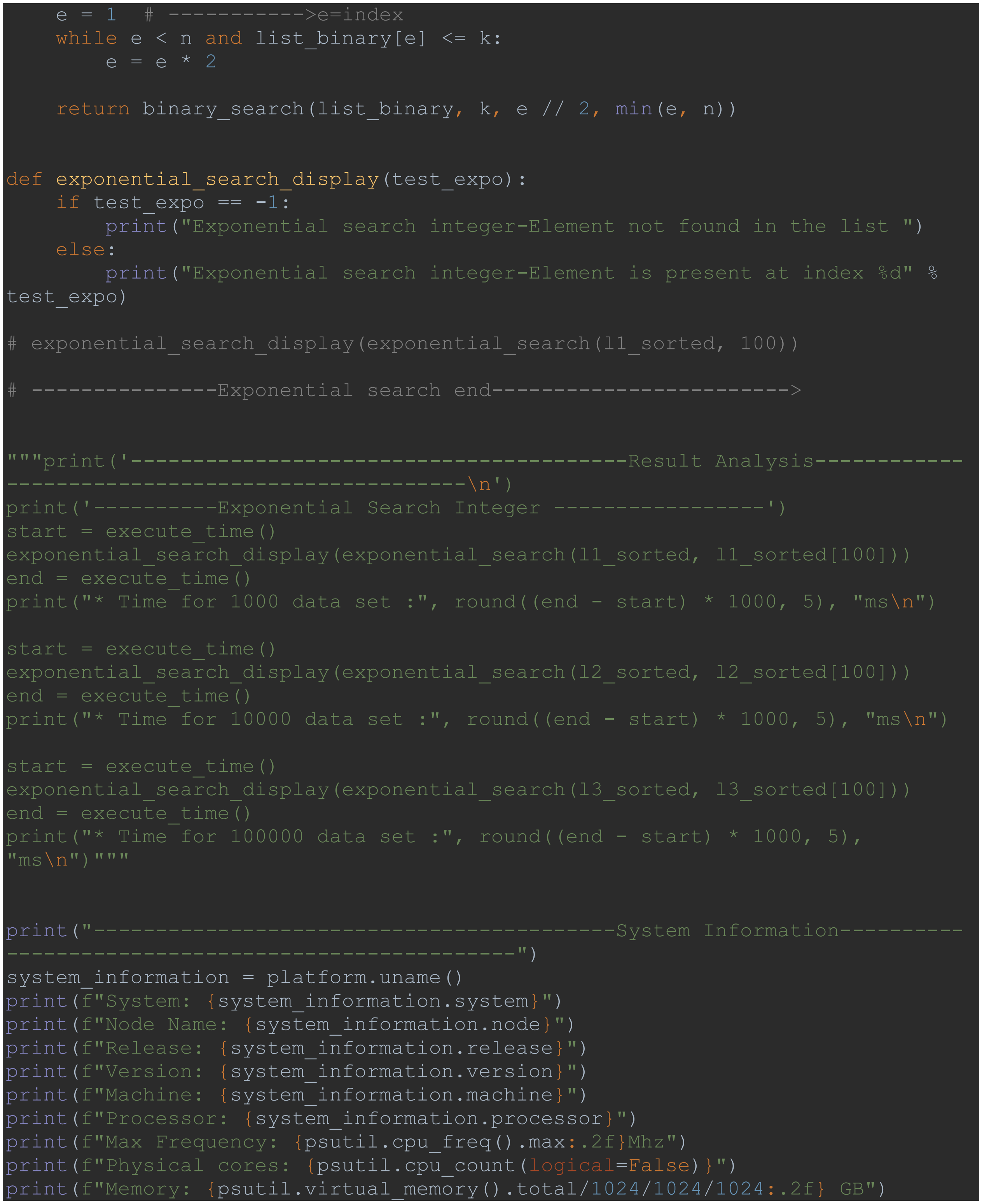
```python
import random
from timeit import default_timer as execute_time
import math
import psutil
import platform


list_range = range(0, 100000)  # range for creating lists (l1,l2,l3)

l1 = random.sample(list_range, 1000)  # random list - not sorted
l1_sorted = sorted(l1)                # random list - sorted
l2 = random.sample(list_range, 10000)
l2_sorted = sorted(l2)
l3 = random.sample(list_range, 100000)
l3_sorted = sorted(l3)

l4 = [1, 2, 3, 4, 5, 6, 7, 80]       #  only test purpose

# ---------------for linear search----------------------->


def linear_search(list_linear, key):
    for i in range(len(list_linear)):
        if list_linear[i] == key:
            return i
    return -1


def linear_search_display(test_linear):
    if test_linear == -1:
        print("linear search integer-Element not found")
    else:
        print("linear search integer-Key element is found at index :",
test_linear)
```



```
linear_search_display(linear_search(l2, 2))


# ---------------linear search end------------------------>


# ---------------for binary search------------------------>
# ---------------only use sorted list--------------------->


def binary_search(list_binary, k, left, right):
    while right >= left:
        mid = left + (right - left) // 2
        if list_binary[mid] == k:
            return mid
        elif list_binary[mid] > k:
            right = mid - 1
        else:
            left = mid + 1
    return -1


def binary_search_display(test_binary):
    if test_binary == -1:
        print("binary search integer-Element not found")
    else:
        print("binary search integer-Key element is found at index :",
test_binary)


# binary_search_display(binary_search(l4, 80, 0, len(l4) - 1))
# binary_search_display(binary_search(l1_sorted, 230, 0, len(l1_sorted) - 1))

# ---------------Binary search end------------------------>

# ----------------For Jump search---------------------------->

def jump_search(jump_list, search_jump):
    low = 0
    m = int(math.sqrt(len(jump_list)))

    for j in range(0, len(jump_list), m):
        if jump_list[j] < search_jump:
            low = j
        elif jump_list[j] == search_jump:
            return j
        else:
            break

    c = low
    for j in jump_list[low:]:
        if j == search_jump:
            return c
        c = c+1
    return -1
```



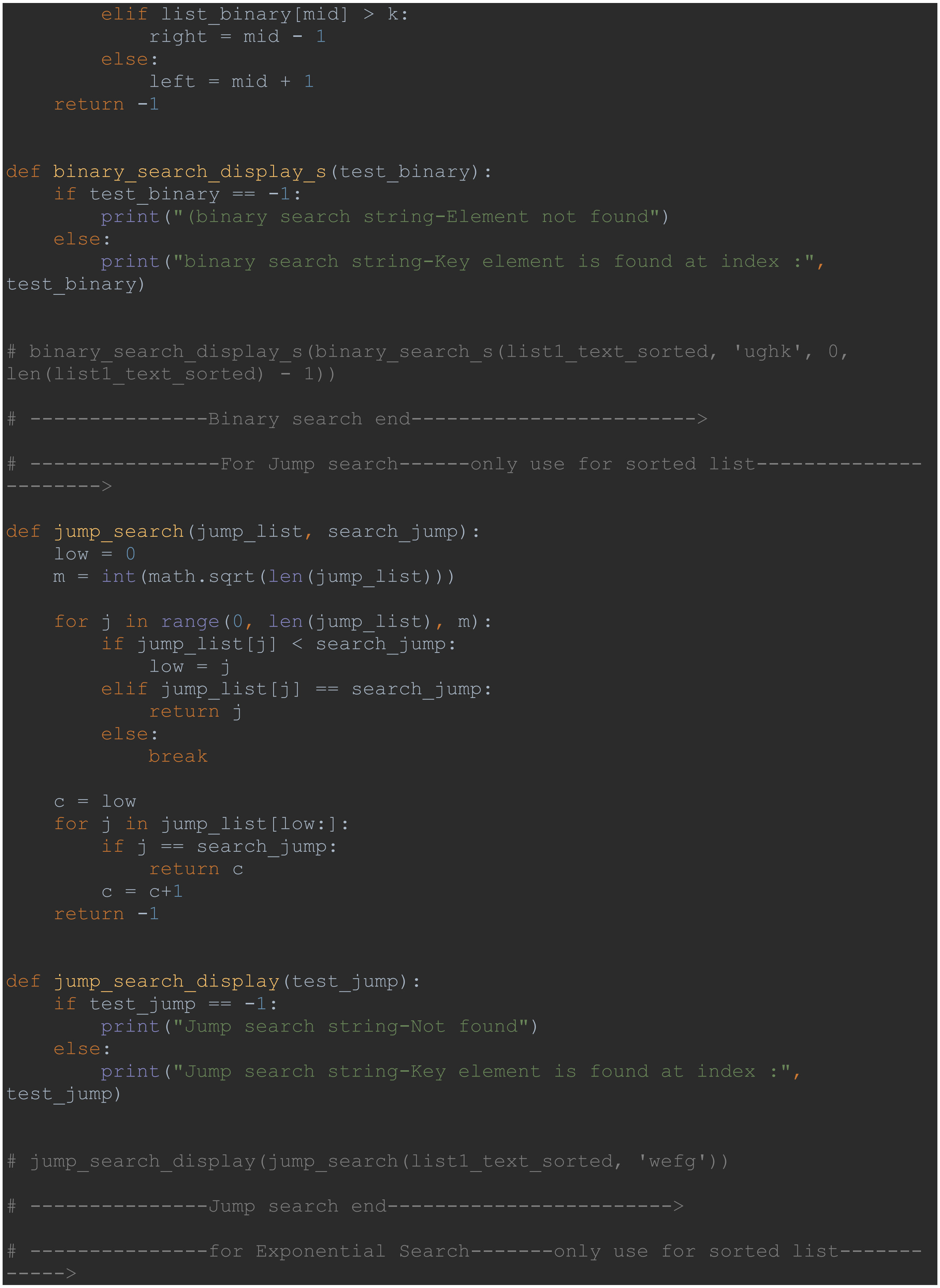
```
def jump_search_display(test_jump):
    if test_jump == -1:
        print("Jump search integer-Not found")
    else:
        print("Jump search integer-Key element is found at index :",
test_jump)


# jump_search_display(jump_search(l1_sorted,21))

# ---------------Jump search end------------------------>

# ---------------for Interpolation Search------------------>


def nearest_mid(input_list, first, last, search_value):
    return first + ((last - first) // (input_list[last] - input_list[first]))
* (search_value - input_list[first])


def interpolation_search(sorted_list, s):
    list_size = len(sorted_list) - 1
    index_first = 0
    index_last = list_size

    while index_first <= index_last:
        mid_point = nearest_mid(sorted_list, index_first, index_last, s)
        if mid_point > index_last or mid_point < index_first:
            return None
        if sorted_list[mid_point] == s:
            return mid_point
        if s > sorted_list[mid_point]:
            index_first = mid_point + 1
        else:
            index_last = mid_point - 1


def interpolation_search_display(test_intp):
    print("Interpolation Search integer-Key element is found at index :",
test_intp)


# interpolation_search_display(interpolation_search(l1_sorted, 79))

# ---------------Interpolation search end------------------------>

# ----------------Exponential search----------------------------->

# Used above written binary search function for this Exponential search

def exponential_search(list_binary, k):
    n = len(list_binary)

    if list_binary[0] == k:
        return 0
```



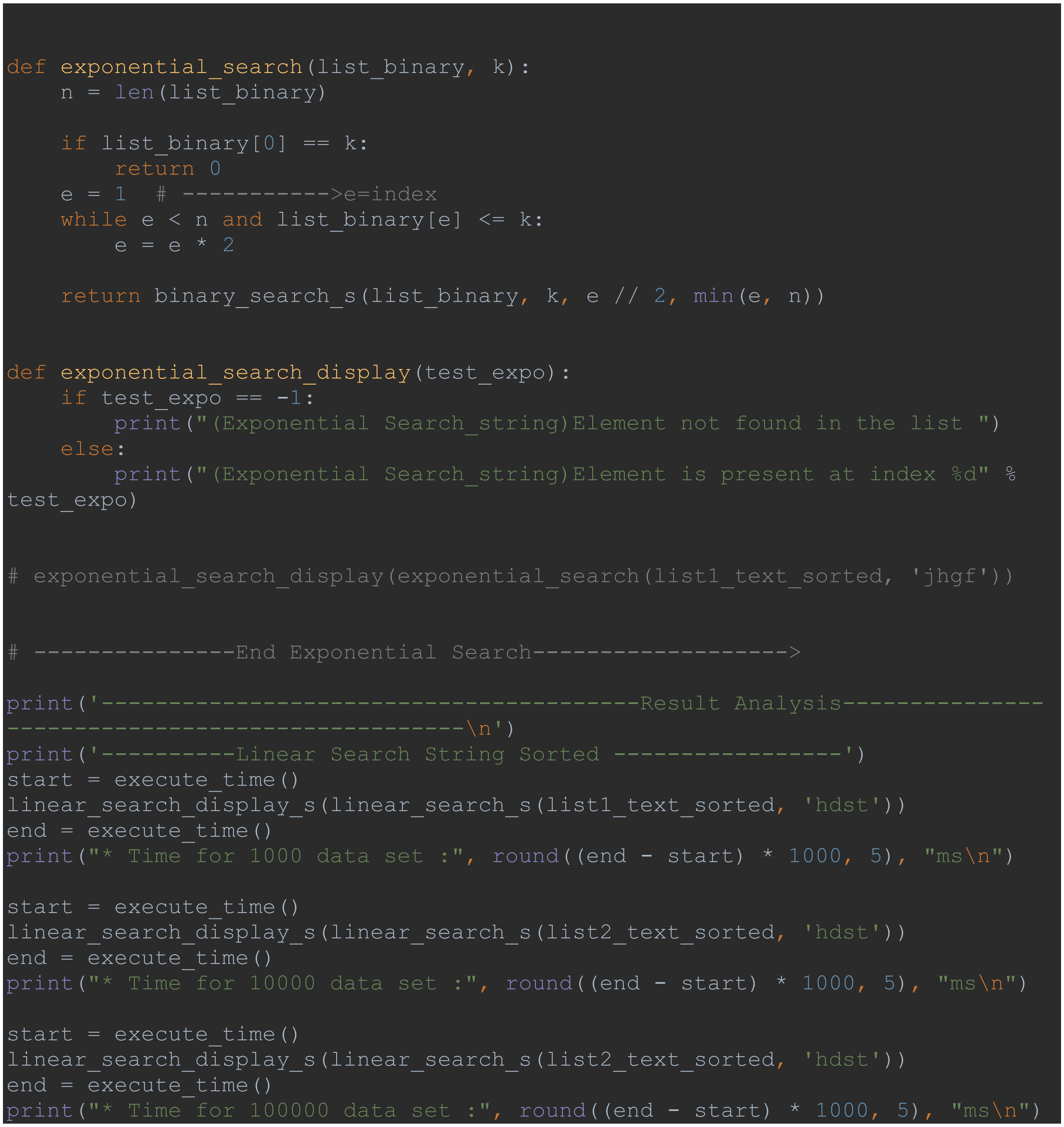
```
    e = 1  # ----------->e=index
    while e < n and list_binary[e] <= k:
        e = e * 2

    return binary_search(list_binary, k, e // 2, min(e, n))


def exponential_search_display(test_expo):
    if test_expo == -1:
        print("Exponential search integer-Element not found in the list ")
    else:
        print("Exponential search integer-Element is present at index %d" %
test_expo)

# exponential_search_display(exponential_search(l1_sorted, 100))

# ---------------Exponential search end------------------------>


"""print('---------------------------------------Result Analysis------------
-------------------------------------\n')
print('----------Exponential Search Integer -----------------')
start = execute_time()
exponential_search_display(exponential_search(l1_sorted, l1_sorted[100]))
end = execute_time()
print("* Time for 1000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
exponential_search_display(exponential_search(l2_sorted, l2_sorted[100]))
end = execute_time()
print("* Time for 10000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
exponential_search_display(exponential_search(l3_sorted, l3_sorted[100]))
end = execute_time()
print("* Time for 100000 data set :", round((end - start) * 1000, 5),
"ms\n")"""


print("------------------------------------------System Information----------
-----------------------------------------")
system_information = platform.uname()
print(f"System: {system_information.system}")
print(f"Node Name: {system_information.node}")
print(f"Release: {system_information.release}")
print(f"Version: {system_information.version}")
print(f"Machine: {system_information.machine}")
print(f"Processor: {system_information.processor}")
print(f"Max Frequency: {psutil.cpu_freq().max:.2f}Mhz")
print(f"Physical cores: {psutil.cpu_count(logical=False)}")
print(f"Memory: {psutil.virtual_memory().total/1024/1024/1024:.2f} GB")
```

Python code for Strings

```
import random
import string
import math
import psutil
from timeit import default_timer as execute_time

list1_text = []
list2_text = []
list3_text = []

for i in range(0, 1000):
    list1_text.append(''.join(random.sample(string.ascii_lowercase, 4)))
for i in range(0, 10000):
    list2_text.append(''.join(random.sample(string.ascii_lowercase, 4)))
for i in range(0, 100000):
    list3_text.append(''.join(random.sample(string.ascii_lowercase, 4)))

list1_text_sorted = sorted(list1_text)
list2_text_sorted = sorted(list2_text)
list3_text_sorted = sorted(list3_text)


# ---------------for linear search------------------------>


def linear_search_s(list_linear_s, key):
    for s in range(len(list_linear_s)):
        if list_linear_s[s] == key:
            return s
    return -1


def linear_search_display_s(test_linear_s):
    if test_linear_s == -1:
        print("(linear search_string)Element not found")
    else:
        print("(linear search_string)Key element is found at index :",
test_linear_s)


# linear_search_display_s(linear_search_s(list1_text, list1_text[100]))
# linear_search_display_s(linear_search_s(list1_text_sorted,
list1_text_sorted[100]))

# ---------------linear search end------------------------>

# ---------------for binary search-------only use for sorted list------------
--------->


def binary_search_s(list_binary, k, left, right):
    while right >= left:
        mid = left + (right - left) // 2
        if list_binary[mid] == k:
            return mid
```



```
        elif list_binary[mid] > k:
            right = mid - 1
        else:
            left = mid + 1
    return -1


def binary_search_display_s(test_binary):
    if test_binary == -1:
        print("(binary search string-Element not found")
    else:
        print("binary search string-Key element is found at index :",
test_binary)


# binary_search_display_s(binary_search_s(list1_text_sorted, 'ughk', 0,
len(list1_text_sorted) - 1))

# ---------------Binary search end----------------------->

# ----------------For Jump search------only use for sorted list--------------
-------->

def jump_search(jump_list, search_jump):
    low = 0
    m = int(math.sqrt(len(jump_list)))

    for j in range(0, len(jump_list), m):
        if jump_list[j] < search_jump:
            low = j
        elif jump_list[j] == search_jump:
            return j
        else:
            break

    c = low
    for j in jump_list[low:]:
        if j == search_jump:
            return c
        c = c+1
    return -1


def jump_search_display(test_jump):
    if test_jump == -1:
        print("Jump search string-Not found")
    else:
        print("Jump search string-Key element is found at index :",
test_jump)


# jump_search_display(jump_search(list1_text_sorted, 'wefg'))

# ---------------Jump search end----------------------->

# ---------------for Exponential Search-------only use for sorted list-------
----->
```



```
def exponential_search(list_binary, k):
    n = len(list_binary)

    if list_binary[0] == k:
        return 0
    e = 1  # ----------->e=index
    while e < n and list_binary[e] <= k:
        e = e * 2

    return binary_search_s(list_binary, k, e // 2, min(e, n))


def exponential_search_display(test_expo):
    if test_expo == -1:
        print("(Exponential Search_string)Element not found in the list ")
    else:
        print("(Exponential Search_string)Element is present at index %d" %
test_expo)


# exponential_search_display(exponential_search(list1_text_sorted, 'jhgf'))


# ---------------End Exponential Search------------------->

print('----------------------------------------Result Analysis---------------
----------------------------------\n')
print('----------Linear Search String Sorted -----------------')
start = execute_time()
linear_search_display_s(linear_search_s(list1_text_sorted, 'hdst'))
end = execute_time()
print("* Time for 1000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
linear_search_display_s(linear_search_s(list2_text_sorted, 'hdst'))
end = execute_time()
print("* Time for 10000 data set :", round((end - start) * 1000, 5), "ms\n")

start = execute_time()
linear_search_display_s(linear_search_s(list2_text_sorted, 'hdst'))
end = execute_time()
print("* Time for 100000 data set :", round((end - start) * 1000, 5), "ms\n")
```