Task 1:

Implementation and Analysis of Search Algorithms with User Input

BANA 622

May 16, 2024

Emely Callejas, Ashley Cortez, Robert Pimentel, Angelica Verduzco

Table of Contents

Introduction

The goal of this project is to develop regular and recursive programs in Python that conduct linear and binary search algorithms. Each function should take input from the user for the list and the item in the list they want to search for. In this exercise, we will include and use the following concepts: loops, conditionals, recursion, user input handling, and algorithm efficiency. Ultimately, we will need to compare the results of each algorithm and analyze their performance in terms of execution time, number of instructions, and memory usage.

Algorithms are important in our world today for numerous reasons. As we've discussed in our course, algorithms can be described as a set of instructions that are used to produce a result or a solution to a problem. The main reason algorithms are important is because they help provide solutions to problems in a replicable, efficient and effective way. Algorithms should be able to reproduce a set of instructions that can be applied to different scenarios related to the problem at hand.

Methodology

For Task 1, we developed several different functions to complete the assignment. There are two functions to handle user input for both the list of numbers and the number the user would like to search. We then developed iterative and recursive versions of a linear search and of a binary search. Below we will elaborate on each function and how they work, the actual Python code can be referenced in the appendix.

The function named **getListInput()** is what we used to prompt the user to input a list of numbers they'd like to use for the search. Within a while true loop, the function directs the user to input the numbers separated by commas. The function then takes the user-provided list and delimits each item in the list by a comma. It then converts each number in the list into an integer and strips any spaces from the entry so that the final list only includes the actual numbers. We've also added a condition so that if a user inputs anything other than numerical values, the function will prompt the user to enter numbers in the list only.

The function named **searchNum()** is the next input function developed to prompt the user to input the number they'd like to search for within the list. Also within a while true loop, the function asks the user for an input and the input function is masked with 'int' so that the input is transformed into an integer. The same condition to only accept numerical values is also applied here; if the user inputs anything other than a numerical value, they will be prompted to try again.

The function named **linearSearch()** is the first search function implemented in our assignment. The parameters defined in the function are numList (the list of numbers provided by the user) and userSearch (the number that the user would like to search for). Using the for loop, the function will iterate through each number in numList to find a match for userSearch. If the function finds a match within the list, it will return the index position of the number in the list. If it is unable to find a match, the function will return -1.

The function named **linearSearchRec()** has the same outcome as the previous function, but executes the solution recursively. This recursive function takes the same two parameters as the regular linear search function, however a third parameter (indexPosition2=0) was added to initialize the index. As a first step, the function checks if the index position is greater than or equal to the length of numList, if this is true the function will return -1. If the previous conditions were not met, then the function checks if the number in the index position matches the

userSearch number. If there is a match, it will return the index position of the number in the list If it is unable to find a match, the function will also return -1.
The function named **binarySearch()** is an alternative search function to the linear search functions. Within the binary search, the function takes the two parameters, numList and userSearch. The function first initializes “left” as zero to indicate that the algorithm should start from index 0 of the numList. Then “right” is defined as the length of the numList -1 to get the index number of the rightmost item in the list. The function then uses a while conditional statement to say while left is less than or equal to right, the midpoint is defined as the sum of left and right divided by 2. The function adds more conditional statements (if statements) to check if userSearch is equal to the number of the midpoint index, if this is true the function will return the index of the midpoint. If the userSearch is less than the midpoint index, the function will redefine “right” as midpoint - 1. Lastly if anything outside of the previous conditions, the function will redefine left as midpoint +1. If left is not less than or equal to right, the function will return -1 because it will have searched through the entirety of the list.

The function named **binarySearchRec()** has the same outcome as the previous function (conducts a binary search in a list), however it is conducted recursively. The recursive binary search function also has two additional predefined parameters for left and right. For the base case, the function checks if left is less than or equal to right, and calculates the midpoint similarly to the regular binary search function. The next conditional if statements below the base case compare the userSearch value to the midpoint index of the numList. If userSearch equals the midpoint index, the function will return the midpoint index. If it is less than the number at the midpoint index, then the function will call itself recursively, while adjusting the fourth parameter to “midpoint -1”. If the last two conditions are not true, the function will call itself recursively, while adjusting the third parameter to “midpoint +1”. If the base case is not true, the function will return -1.

Results

Once we completed the formulation of the various functions needed for this assignment, we conducted a few performance analyses of each search function. Performance metrics considered were: execution time, number of instructions and memory usage.

In order to test execution time, we imported the time library and then nested each function within “start_time = time.time()” and “end_time=time.time()”. We then calculated the total execution time by taking the difference between end_time and start_time. We conducted five tests for each function and recorded the execution time of each test in the table below:

	Execution Time (seconds)			
Test	linearSearch	linearSearchRec	binarySearch	binarySearchRec
1	0.0068	0.0030	0.0007	0.0005
2	0.0005	0.0090	0.0086	0.0005
3	0.0051	0.0045	0.0003	0.0061
4	0.0038	0.0046	0.0018	0.0016
5	0.0066	0.0005	0.0034	0.0010
Averages	**0.0046**	**0.0043**	**0.0030**	**0.0020**

Interestingly, the recursive version of both the linear and binary search functions resulted in a lower average execution time than their iterative counterparts. For linear vs. binary, binary search functions took less time on average when compared to linear search functions. This performance is as expected since the binary search functions break down the lists into smaller, more manageable sections. Whereas the linear search functions go through the list one number at a time.

Next, we took a look at the iteration and recursive call counts for each function (see table below). For the test, we kept the control variables as the length of the list and the search term constant. In a six-item list, the linear search functions resulted in a total of six iterations/recursive calls. Within the binary search functions, there were only two iterations/recursive calls needed.

	Amount of Iterations/Recursive Calls
linearSearch	6
linearSearchRec	6
binarySearch	2
binarySearchRec	2

Finally, we addressed performance by analyzing memory performance of each function. In order to measure memory usage, we transformed the functions to include functions from the memory profiler library. First we measured the initial memory usage before executing the function, then we measured the final memory usage once the function completed its run. We then took the difference between the final and initial values to understand how much memory was used for each function. From the table below, we can see that the recursive linear search function resulted in the highest memory usage. This makes sense because earlier we confirmed that the linear functions took more time to run and had more iterations when compared to the binary functions. From what we learned, we also know that recursive functions inherently take up more space than iterative functions. With the combination of these two attributes, it makes sense that the recursive linear search function would perform poorly in comparison to all other functions.

	Memory Usage (bytes)		
	Initial Memory Usage	Final Memory Usage	Difference
linearSearch	107,277,378	107,281,020	3,642
linearSearchRec	107,300,410	107,308,095	**7,685**
binarySearch	107,336,784	107,338,644	**1,860**
binarySearchRec	107,353,615	107,355,532	1,917

As binary search functions performed most efficiently in terms of the first two measurements, we can see that the iterative binary search function had the least amount of memory usage. The recursive binary search function had slightly more memory usage, but it was not by a significant amount.

Discussion

When comparing the linear and binary search algorithms it is apparent that each has its own benefits and drawbacks. Linear search is straightforward to implement and understand, knowing that the search method goes through each item in a list one at a time. Alternatively, the recursive linear search function uses the calls on itself recursively instead of using a for loop; this implementation causes the function to appear more complex and can be trickier to debug. Through this exercise, we learned that linear search would be an inefficient solution for a large list because it goes through each element of the list at a time. This performance method causes the linear search algorithms to have a longer execution time than the binary search.

In contrast, the iterative binary search algorithm has a more complex implementation but it offers a better overall performance compared to the linear search methods. This is primarily because binary search algorithms arrive at a solution by splitting the list into smaller, more manageable segments. The recursive binary search algorithm works similar to the iterative binary search except at the end, it calls itself recursively and makes adjustments to the right or left parameters if it is not able to find the target variable in a certain segment. Binary search algorithms offer a faster average performance time, utilizes less iterations/recursive calls, and has a slightly lower memory usage than linear search algorithms. The slight drawback for binary search is due to the complexity of algorithm structure, making it slightly more difficult to understand and debug. Ultimately, the binary search method outperformed linear search in all performance metrics.

Conclusion

In conclusion, this project allows us to set up and compare the different ways of running linear and binary search algorithms, both iteratively and recursively. Each function is designed to handle user input and carry out searches efficiently. From the analysis, we saw differences in time consumption, how many steps each version takes and how much memory is used. The binary search functions performed better than the linear search functions in a general sense since the binary search can break down the search section better. Although the recursive functions (for both linear and binary search) used more memory than their iterative counterparts, these

functions finished in a faster time frame. The results demonstrate the importance of choosing the right algorithm and approach depending on the task. As linear search algorithms tend to be straightforward in structure and search for a target in a list one item at a time, we'd recommend using an iterative linear search algorithm when searching a small list or dataset. Alternatively, we'd recommend using the iterative binary search algorithm for searching a larger dataset or list, as it is the most time efficient and takes less memory. Lastly, as a key takeaway from this project, we do not only consider applying these algorithms simply to provide solutions to tasks, but we now further understand the compromises in various performance metrics when picking certain algorithms.

Appendices


```
# Get user input for the list they'd like to search
def getListInput():
    while True:
        numList = input("Enter a list of numbers separated by commas: ")
        inputList = numList.split(',')
        try:
            inputList = [int(num.strip()) for num in inputList]
            return inputList
        except ValueError:
            print("Invalid entry, please enter only numeric values in the list")


# Get user input for the number they'd like to search
def searchNum():
    while True:
        try:
            userSearch = int(input("Enter the number you want to search for in the list: "))
            return userSearch
        except ValueError:
            print("Invalid entry, please enter a numeric value.")

# Search for the desired number in the list, one number at a time (linearSearch)
def linearSearch(numList,userSearch):
    for indexPosition1, num in enumerate(numList):
        if num == userSearch:
            return indexPosition1
    return -1


# Search for the desired number in the list, one number at a time & recursively (linearSearchRe
def linearSearchRec(numList, userSearch, indexPosition2 = 0):
    if indexPosition2 >= len(numList):
        return -1
    if numList[indexPosition2] == userSearch:
        return indexPosition2
    return linearSearchRec(numList, userSearch, indexPosition2 + 1)


# Search for the desired number in the list, using binary search methods
def binarySearch(numList, userSearch):
    left = 0
    right = (len(numList)-1)
    while left <= right:
        midpoint = (left + right) //2
        if userSearch == numList[midpoint]:
            return midpoint
        elif userSearch < numList[midpoint]:
            right = midpoint - 1
        else:
            left = midpoint + 1
    return -1
```

```
# Search for the desired number in the list, using binary search methods, recursively
def binarySearchRec(numList, userSearch, left=0, right=None):
    if right is None:
        right = len(numList) - 1

    if left <= right:
        midpoint = (left + right) // 2
        if userSearch == numList[midpoint]:
            return midpoint
        elif userSearch < numList[midpoint]:
            return binarySearchRec(numList, userSearch, left, midpoint - 1)
        else:
            return binarySearchRec(numList, userSearch, midpoint + 1, right)
    else:
        return -1


def main():
    userList = getListInput()
    userSearch = searchNum()
    indexPositionLinearReg = linearSearch(userList,userSearch)
    indexPositionLinearRec = linearSearchRec(userList,userSearch)
    indexPositionBinaryReg = binarySearch(userList, userSearch)
    indexPositionBinaryRec = binarySearchRec(userList, userSearch)

    print("You entered the following list of numbers: ", userList)
    print("You entered the following number to search: ", userSearch)
    if indexPositionLinearReg == -1:
        print("linearSearch Results = The number you searched for was not found in the list pro
    else:
        print("linearSearch Results = The number you searched for was found at index position:"
    if indexPositionLinearRec == -1:
        print("linearSearchRec Results = The number you searched for was not found in the list
    else:
        print("linearSearchRec Results = The number you searched for was found at index positio
    if indexPositionBinaryReg == -1:
        print("binarySearchReg Results = The number you searched for was not found in the list.
    else:
        print("binarySearchReg Results = The number you searched for was found at index positio
    if indexPositionBinaryRec == -1:
        print("binarySearchReg Results = The number you searched for was not found in the list.
    else:
        print("binarySearchReg Results = The number you searched for was found at index positio
if __name__ =="__main__":
    main()
```

```
Enter a list of numbers separated by commas: 1,2,3
Enter the number you want to search for in the list: 3
You entered the following list of numbers:  [1, 2, 3]
You entered the following number to search:  3
linearSearch Results = The number you searched for was found at index position: 2
linearSearchRec Results = The number you searched for was found at index position: 2
binarySearchReg Results = The number you searched for was found at index position: 2
binarySearchReg Results = The number you searched for was found at index position: 2
```

```
import time
userList = getListInput()
userSearch = searchNum()
```

```
Enter a list of numbers separated by commas: 1,4,5,7,6,8,9
Enter the number you want to search for in the list: 6
```

```
# Time measurement - Linear Search (Regular)
start_time = time.time()
indexPositionLinearReg = linearSearch(userList,userSearch)
print("You entered the following list of numbers: ", userList)
print("You entered the following number to search: ", userSearch)
if indexPositionLinearReg == -1:
    print("linearSearch Results = The number you searched for was not found in the list provide
else:
    print("linearSearch Results = The number you searched for was found at index position:",ind
end_time = time.time()
execution_time = end_time - start_time
print("Execution Time:", execution_time, "seconds")
```

```
You entered the following list of numbers:  [1, 4, 5, 7, 6, 8, 9]
You entered the following number to search:  6
linearSearch Results = The number you searched for was found at index position: 4
Execution Time: 0.006551504135131836 seconds
```

```
# Time measurement - Linear Search (Recursive)

start_time = time.time()
indexPositionLinearRec = linearSearchRec(userList,userSearch)
print("You entered the following list of numbers: ", userList)
print("You entered the following number to search: ", userSearch)
if indexPositionLinearRec == -1:
        print("linearSearchRec Results = The number you searched for was not found in the list
else:
    print("linearSearchRec Results = The number you searched for was found at index position:",
end_time = time.time()
execution_time = end_time - start_time
print("Execution Time:", execution_time, "seconds")
```

```
You entered the following list of numbers:  [1, 4, 5, 7, 6, 8, 9]
You entered the following number to search:  6
linearSearchRec Results = The number you searched for was found at index position: 4
Execution Time: 0.0005199909210205078 seconds
```

```
# Time measurement - Binary Search (Regular)

start_time = time.time()
indexPositionBinaryReg = binarySearch(userList, userSearch)
print("You entered the following list of numbers: ", userList)
print("You entered the following number to search: ", userSearch)
if indexPositionBinaryReg == -1:
    print("binarySearchReg Results = The number you searched for was not found in the list.")
else:
    print("binarySearchReg Results = The number you searched for was found at index position:",
end_time = time.time()
execution_time = end_time - start_time
print("Execution Time:", execution_time, "seconds")
```

```
You entered the following list of numbers:  [1, 4, 5, 7, 6, 8, 9]
You entered the following number to search:  6
binarySearchReg Results = The number you searched for was not found in the list.
```

```
    Execution Time: 0.0034246444702148438 seconds


# Time measurement - Binary Search (Recursive)

start_time = time.time()
indexPositionBinaryRec = binarySearchRec(userList, userSearch)
print("You entered the following list of numbers: ", userList)
print("You entered the following number to search: ", userSearch)
if indexPositionBinaryRec == -1:
    print("binarySearchReg Results = The number you searched for was not found in the list.")
else:
    print("binarySearchReg Results = The number you searched for was found at index position:",
end_time = time.time()
execution_time = end_time - start_time
print("Execution Time:", execution_time, "seconds")
```

```
You entered the following list of numbers:  [1, 4, 5, 7, 6, 8, 9]
You entered the following number to search:  6
binarySearchReg Results = The number you searched for was not found in the list.
Execution Time: 0.0010089874267578125 seconds
```

```
import sys

IN_COLAB = "google.colab" in sys.modules
if IN_COLAB:
    %pip install --quiet line_profiler snakeviz pyinstrument eliot eliot-tree
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 717.6/717.6 kB 6.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 283.7/283.7 kB 9.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 106.9/106.9 kB 7.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 113.1/113.1 kB 6.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 40.1/40.1 kB 3.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 247.5/247.5 kB 11.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 117.7/117.7 kB 12.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 191.7/191.7 kB 12.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 142.5/142.5 kB 10.9 MB/s eta 0:00:00
```

```
%timeit linearSearch(userList,userSearch)
```

```
764 ns ± 223 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

```
pip install memory-profiler
```

```
Collecting memory-profiler
  Downloading memory_profiler-0.61.0-py3-none-any.whl (31 kB)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from memo
Installing collected packages: memory-profiler
Successfully installed memory-profiler-0.61.0
```

```
pip install guppy3
```

```
Collecting guppy3
  Downloading guppy3-3.1.4.post1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 645.4/645.4 kB 8.8 MB/s eta 0:00:00
Installing collected packages: guppy3
Successfully installed guppy3-3.1.4.post1
```

```
from guppy import hpy
from memory_profiler import profile

#ITERATIONS AND MEMORY PROFILER - LINEARSEARCH
def linearSearch(numList, userSearch):
    iteration_count = 0  # Initialize the iteration counter
    for indexPosition1, num in enumerate(numList):
        iteration_count += 1
        if num == userSearch:
            print(f"Number of iterations: {iteration_count}")
            return indexPosition1
    print(f"Number of iterations: {iteration_count}")
    return -1

@profile
def main():
    h = hpy()
    userList = [1, 2, 6, 5, 3, 7, 9, 8, 4]
    userSearch = 7

    print("Initial memory usage:", h.heap())

    indexPositionLinearReg = linearSearch(userList, userSearch)
    print("You entered the following list of numbers: ", userList)
    print("You entered the following number to search: ", userSearch)
    if indexPositionLinearReg == -1:
        print("linearSearch Results = The number you searched for was not found in the list pro
    else:
        print("linearSearch Results = The number you searched for was found at index position:"

    print("Final memory usage:", h.heap())

if __name__ == "__main__":
    main()
```

```
PYDEV DEBUGGER WARNING:
sys.settrace() should not be used when the debugger is being used.
This may cause the debugger to stop working correctly.
If this is needed, please check:
http://pydev.blogspot.com/2007/06/why-cant-pydev-debugger-work-with.html
to see how to restore the debug tracing back correctly.
Call Location:
  File "/usr/local/lib/python3.10/dist-packages/memory_profiler.py", line 847, in enable
    sys.settrace(self.trace_memory_usage)

ERROR: Could not find file <ipython-input-35-bba7710669df>
NOTE: %mprun can only be used on functions defined in physical files, and not in the IPythoı
Initial memory usage: Partition of a set of 892301 objects. Total size = 107277378 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0 359276  40 38946991  36  38946991  36 str
     1 166607  19 12540424  12  51487415  48 tuple
     2  46658   5  8355813   8  59843228  56 types.CodeType
     3  74366   8  6255679   6  66098907  62 bytes
     4  15309   2  6202064   6  72300971  67 dict (no owner)
     5  42777   5  6159888   6  78460859  73 function
```

```
     6   5490   1  5318200   5  83779059  78 type
     7   2150   0  3140976   3  86920035  81 dict of module
     8   5490   1  2850312   3  89770347  84 dict of type
     9   9775   1  2422000   2  92192347  86 list
<1882 more rows. Type e.g. '_.more' to view.>
Number of iterations: 6
You entered the following list of numbers:  [1, 2, 6, 5, 3, 7, 9, 8, 4]
You entered the following number to search:  7
linearSearch Results = The number you searched for was found at index position: 5
Final memory usage:
PYDEV DEBUGGER WARNING:
sys.settrace() should not be used when the debugger is being used.
This may cause the debugger to stop working correctly.
If this is needed, please check:
http://pydev.blogspot.com/2007/06/why-cant-pydev-debugger-work-with.html
to see how to restore the debug tracing back correctly.
Call Location:
  File "/usr/local/lib/python3.10/dist-packages/memory_profiler.py", line 850, in disable
    sys.settrace(self._original_trace_function)

Partition of a set of 892360 objects. Total size = 107281020 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0 359276  40 38946991  36  38946991  36 str
     1 166628  19 12541752  12  51488743  48 tuple
     2  46658   5  8356251   8  59844994  56 types.CodeType
     3  74366   8  6255679   6  66100673  62 bytes
     4  15312   2  6202256   6  72302929  67 dict (no owner)
     5  42777   5  6159888   6  78462817  73 function
     6   5490   1  5318200   5  83781017  78 type
     7   2150   0  3140976   3  86921993  81 dict of module
     8   5490   1  2850312   3  89772305  84 dict of type
     9   9775   1  2422000   2  92194305  86 list
<1885 more rows. Type e.g. '_.more' to view.>
```

```
#ITERATIONS AND MEMORY PROFILER - LINEARSEARCH - RECURSIVE
def linearSearchRec(numList, userSearch, indexPosition2=0, call_count=0):
    call_count += 1  # Increment the recursive call counter
    if indexPosition2 >= len(numList):
        print(f"Total recursive calls: {call_count}")
        return -1
    if numList[indexPosition2] == userSearch:
        print(f"Total recursive calls: {call_count}")
        return indexPosition2
    return linearSearchRec(numList, userSearch, indexPosition2 + 1, call_count)

@profile
def main():
    h = hpy()
    userList = [1, 2, 6, 5, 3, 7, 9, 8, 4]
    userSearch = 7

    print("Initial memory usage:", h.heap())

    indexPositionLinearRec = linearSearchRec(userList, userSearch)
    print("You entered the following list of numbers: ", userList)
    print("You entered the following number to search: ", userSearch)
    if indexPositionLinearRec == -1:
        print("linearSearchRec Results = The number you searched for was not found in the list pı
    else:
        print("linearSearchRec Results = The number you searched for was found at index position:
```

```
        print("Final memory usage:", h.heap())

if __name__ == "__main__":
    main()
```

```
ERROR: Could not find file <ipython-input-36-c08ba4b68d59>
NOTE: %mprun can only be used on functions defined in physical files, and not in the IPython
Initial memory usage: Partition of a set of 892515 objects. Total size = 107300410 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0 359321  40 38954380  36  38954380  36 str
     1 166685  19 12545528  12  51499908  48 tuple
     2  46660   5  8361366   8  59861274  56 types.CodeType
     3  74370   8  6256253   6  66117527  62 bytes
     4  15311   2  6202824   6  72320351  67 dict (no owner)
     5  42777   5  6159888   6  78480239  73 function
     6   5490   1  5318200   5  83798439  78 type
     7   2150   0  3140976   3  86939415  81 dict of module
     8   5490   1  2850312   3  89789727  84 dict of type
     9   9776   1  2422304   2  92212031  86 list
<1879 more rows. Type e.g. '_.more' to view.>
Total recursive calls: 6
You entered the following list of numbers:  [1, 2, 6, 5, 3, 7, 9, 8, 4]
You entered the following number to search:  7
linearSearchRec Results = The number you searched for was found at index position: 5
Final memory usage: Partition of a set of 892568 objects. Total size = 107308095 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0 359322  40 38954433  36  38954433  36 str
     1 166701  19 12546520  12  51500953  48 tuple
     2  46660   5  8361366   8  59862319  56 types.CodeType
     3  74370   8  6256253   6  66118572  62 bytes
     4  15315   2  6207864   6  72326436  67 dict (no owner)
     5  42777   5  6159888   6  78486324  73 function
     6   5490   1  5318200   5  83804524  78 type
     7   2150   0  3140976   3  86945500  81 dict of module
     8   5490   1  2850312   3  89795812  84 dict of type
     9   9776   1  2422304   2  92218116  86 list
<1882 more rows. Type e.g. '_.more' to view.>
```

```
#ITERATIONS AND MEMORY PROFILER - BINARY SEARCH
def binarySearch(numList, userSearch):
    left = 0
    right = len(numList) - 1
    iteration_count = 0
    while left <= right:
        iteration_count += 1
        midpoint = (left + right) // 2
        if userSearch == numList[midpoint]:
            print(f"Total iterations: {iteration_count}")
            return midpoint
        elif userSearch < numList[midpoint]:
            right = midpoint - 1
        else:
            left = midpoint + 1
    print(f"Total iterations: {iteration_count}")
    return -1

@profile
def main():
```

```
    h = hpy()
    userList = [1, 2, 6, 5, 3, 7, 9, 8, 4]
    userList.sort()
    userSearch = 7

    print("Initial memory usage:", h.heap())

    indexPositionBinaryReg = binarySearch(userList, userSearch)
    print("You entered the following list of numbers: ", userList)
    print("You entered the following number to search: ", userSearch)
    if indexPositionBinaryReg == -1:
      print("binarySearch Results = The number you searched for was not found in the list.")
    else:
      print("binarySearch Results = The number you searched for was found at index position:",inc

    print("Final memory usage:", h.heap())

if __name__ == "__main__":
    main()
```

```
ERROR: Could not find file <ipython-input-37-dc4ce7aa3a65>
NOTE: %mprun can only be used on functions defined in physical files, and not in the IPythor
Initial memory usage: Partition of a set of 892915 objects. Total size = 107336784 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0 359369  40 38960804  36  38960804  36 str
     1 166737  19 12548968  12  51509772  48 tuple
     2  46662   5  8361846   8  59871618  56 types.CodeType
     3  74374   8  6256459   6  66128077  62 bytes
     4  15314   2  6208136   6  72336213  67 dict (no owner)
     5  42777   5  6159888   6  78496101  73 function
     6   5490   1  5318200   5  83814301  78 type
     7   2150   0  3140976   3  86955277  81 dict of module
     8   5490   1  2850312   3  89805589  84 dict of type
     9   9787   1  2423296   2  92228885  86 list
<1884 more rows. Type e.g. '_.more' to view.>
Total iterations: 2
You entered the following list of numbers:  [1, 2, 3, 4, 5, 6, 7, 8, 9]
You entered the following number to search:  7
binarySearch Results = The number you searched for was found at index position: 6
Final memory usage: Partition of a set of 892955 objects. Total size = 107338644 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0 359369  40 38960804  36  38960804  36 str
     1 166747  19 12549608  12  51510412  48 tuple
     2  46662   5  8361846   8  59872258  56 types.CodeType
     3  74374   8  6256459   6  66128717  62 bytes
     4  15316   2  6208264   6  72336981  67 dict (no owner)
     5  42777   5  6159888   6  78496869  73 function
     6   5490   1  5318200   5  83815069  78 type
     7   2150   0  3140976   3  86956045  81 dict of module
     8   5490   1  2850312   3  89806357  84 dict of type
     9   9787   1  2423296   2  92229653  86 list
<1886 more rows. Type e.g. '_.more' to view.>
```

```
#ITERATIONS AND MEMORY PROFILER - BINARY SEARCH - RECURSIVE
def binarySearchRec(numList, userSearch, left=0, right=None, call_count=0):
    if right is None:
        right = len(numList) - 1
    call_count += 1
    if left <= right:
        midpoint = (left + right) // 2
```

```
            if userSearch == numList[midpoint]:
                print(f"Total recursive calls: {call_count}")
                return midpoint
            elif userSearch < numList[midpoint]:
                return binarySearchRec(numList, userSearch, left, midpoint - 1, call_count)
            else:
                return binarySearchRec(numList, userSearch, midpoint + 1, right, call_count)
        else:
            print(f"Total recursive calls: {call_count}")
            return -1

    @profile
    def main():
        h = hpy()
        userList = [1, 2, 6, 5, 3, 7, 9, 8, 4]
        userList.sort()
        userSearch = 7

        print("Initial memory usage:", h.heap())

        indexPositionBinaryRec = binarySearchRec(userList, userSearch)
        print("You entered the following list of numbers: ", userList)
        print("You entered the following number to search: ", userSearch)
        if indexPositionBinaryRec == -1:
          print("binarySearch Results = The number you searched for was not found in the list.")
        else:
          print("binarySearch Results = The number you searched for was found at index position:",inc

        print("Final memory usage:", h.heap())

    if __name__ == "__main__":
        main()
```

```
ERROR: Could not find file <ipython-input-38-62ad0b523b25>
NOTE: %mprun can only be used on functions defined in physical files, and not in the IPythor
Initial memory usage: Partition of a set of 893051 objects. Total size = 107353615 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0 359415  40 38967907  36  38967907  36 str
     1 166755  19 12550272  12  51518179  48 tuple
     2  46664   5  8365117   8  59883296  56 types.CodeType
     3  74378   8  6256960   6  66140256  62 bytes
     4  15317   2  6208496   6  72348752  67 dict (no owner)
     5  42777   5  6159888   6  78508640  73 function
     6   5490   1  5318200   5  83826840  78 type
     7   2150   0  3140976   3  86967816  81 dict of module
     8   5490   1  2850312   3  89818128  84 dict of type
     9   9792   1  2424000   2  92242128  86 list
<1885 more rows. Type e.g. '_.more' to view.>
Total recursive calls: 2
You entered the following list of numbers:  [1, 2, 3, 4, 5, 6, 7, 8, 9]
You entered the following number to search:  7
binarySearch Results = The number you searched for was found at index position: 6
Final memory usage: Partition of a set of 893096 objects. Total size = 107355532 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0 359416  40 38967960  36  38967960  36 str
     1 166768  19 12551088  12  51519048  48 tuple
     2  46664   5  8365117   8  59884165  56 types.CodeType
     3  74378   8  6256960   6  66141125  62 bytes
     4  15316   2  6208600   6  72349725  67 dict (no owner)
     5  42777   5  6159888   6  78509613  73 function
```