

```

1 #####
2 # DCS 229 -- Project 3: Search
3 # Date: March 4, 2026
4 # Name: Benjamin Adovasio
5 # Resources Used:
6 # https://pressbooks.palni.org/
  anopenguidetodatastructuresandalgorithms/chapter/search/
7 #
8 #####
9
10 import random #to generate the 100 random values
11 import time #for part 4, to calculae total time
12 from collections.abc import Sequence
13
14 #I decided to put all 4 functions into one class to keep
  it organized
15 class Search:
16
17     """
18     This function generates the list of random integers.
19     """
20     def generate_random_list(self, n: int) -> list[int]:
21         array: list[int] = [] #array starts as blank
22         for element in range(n): #will run n times
23             array.append(random.randint(0, 1000)) #
  random integer between 0-1000
24
25         #I added this for testing purposes, to make sure
  42 was in the list
26         #if 42 not in array:
27         #    array[random.randint(0, n - 1)] = 42
28
29         return array
30
31     """
32     This funciton preforms a search on the list for the
  target value, and prints the number of checks it took to
  find the target.

```

```

33     """
34     def linear_search_print(self, arr: Sequence[int],
35     target: int) -> int:
36         checks = 0 #check count starts at 0
37         for value in arr:
38             if value == target:
39                 print("Unsuccessful checks:", checks)
40                 return checks
41                 checks += 1 #adds 1 to the number of checks
42         print("Not found.") #Only gets printed if the
43         code doesnt return earlier
44         return checks
45     """
46     This function is the same as the previous one, but
47     instead of printing the number of checks, it returns the
48     number of checks.
49     """
50     def linear_search_count(self, arr: Sequence[int],
51     target: int) -> int:
52         checks = 0
53         for value in arr:
54             checks += 1
55             if value == target:
56                 return checks
57         return checks
58     """
59     This function uses recursive searching.
60     """
61     def binary_search_recursive(
62     self,
63     arr: Sequence[int],
64     target: int,
65     low: int = 0,
66     high: int | None = None,
67     ) -> int:

```

```

66         if high is None:
67             high = len(arr) - 1
68
69         if low > high:
70             return -1 #base case: if low is greater
than high, the target is not found
71
72         mid = (low + high) // 2 #finds the middle
73
74         if arr[mid] == target: #stops if target found
75             return mid #in this case mid would be the
index of the target
76         elif arr[mid] < target: #either the left or
right half of array is then searched
77             return self.binary_search_recursive(arr,
78             target, mid + 1, high)
79         else:
80             return self.binary_search_recursive(arr,
81             target, low, mid - 1)
82
83 def main(n: int = 1000) -> None:
84     search = Search() #search object to call Search()
85     class
86         """
87         Part 1: Linear search on 100 integers
88         """
89         arr = search.generate_random_list(100) #generate
the list of 100 random integers
90         search.linear_search_print(arr, 42) #search for the
value 42 using linear search and print the number of
checks it took to find 42
91         """
92         Part 2: Same as 1, but return number of checks
93         """
94         total = 0
95         tests = 100

```

```

95
96     for _ in range(tests):
97         arr = search.generate_random_list(100) #
generate the list of 100 random integers
98         total += search.linear_search_count(arr, 42) #
search for the value 42 using linear search and add the
number of checks it took to find 42 to the total
99         print("Average number of checks for 100 tests:",
total / tests) #print the average number of checks it
took to find 42 for 100 tests
100
101     """
102     Part 3: Uses the recursive search
103     """
104     arr = sorted(search.generate_random_list(100)) #
generate the list of 100 random integers and sort it
for binary search
105     index = search.binary_search_recursive(arr, 42) #
search for the value 42 using binary search and get the
index of 42
106     print("Index of 42 in sorted array:", index) #print
the index of 42 in the sorted array
107
108     """
109     Part 4: Tests
110
111     At what number of queries does Sorting + Binary
Search start to show an advantage over Linear Search?:
112     Sorting + binary search shows an advantage at
around 500 queries
113     """
114     arr = search.generate_random_list(n) #generate the
list of n random integers
115
116     for multiplier in [0.5, 1, 2, 4]: #I used a "
multiplier" to avoid rewriting the code for each query
117         query_count = int(n * multiplier)
118         queries = []

```

```
119
120     for _ in range(query_count):
121         queries.append(random.choice(arr))
122
123     # Linear timing
124     start = time.perf_counter()
125     for q in queries:
126         search.linear_search_count(arr, q)
127     end = time.perf_counter()
128     linear_time = end - start
129
130     # Sorting + Binary timing
131     start = time.perf_counter()
132     sorted_arr = sorted(arr)
133     for q in queries:
134         search.binary_search_recursive(sorted_arr,
135         q)
136     end = time.perf_counter()
137     binary_time = end - start
138
139     print("\nQueries:", query_count)
140     print("Linear time:", linear_time)
141     print("Sorting + Binary time:", binary_time)
142
143     # only execute when you run this file directly
144     if __name__ == "__main__":
145         main()
```