

```

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
13 #I decided to put all 4 functions into one class to
  keep it organized
14 class Search:
15
16     """
17         This function generates the list of random
        integers.
18     """
19     def generate_random_list(self, n):
20         array = [] #array starts as blank
21         for element in range(n): #will run n times
22             array.append(random.randint(0, 1000)) #
        random integer between 0-1000
23
24         #I added this for testing purposes, to make
        sure 42 was in the list
25         #if 42 not in array:
26         #    array[random.randint(0, n - 1)] = 42
27
28         return array
29
30     """
31     This funciton preforms a search on the list for
        the target value, and prints the number of checks it

```

```
31 took to find the target.
32 """
33     def linear_search_print(self, arr, target):
34         checks = 0 #check count starts at 0
35         for value in arr:
36             if value == target:
37                 print("Unsuccessful checks:", checks
38             )
39                 return checks
40                 checks += 1 #adds 1 to the number of
41                 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,
47 but instead of printing the number of checks, it
48 returns the number of checks.
49 """
50     def linear_search_count(self, arr, target):
51         checks = 0
52         for value in arr:
53             checks += 1
54             if value == target:
55                 return checks
56         return checks
57 """
58 This function uses recursive searching.
59 """
60     def binary_search_recursive(self, arr, target,
61     low, high):
62         if low > high:
63             return -1 #base case: if low is greater
64             than high, the target is not found
```

```
62         mid = (low + high) // 2 #finds the middle
63
64         if arr[mid] == target: #stops if target found
65             return mid #in this case mid would be the
        index of the target
66         elif arr[mid] < target: #either the left or
        right half of array is then searched
67             return self.binary_search_recursive(arr,
        target, mid + 1, high)
68         else:
69             return self.binary_search_recursive(arr,
        target, low, mid - 1)
70
71 def main():
72     search = Search() #search object to call Search
    () class
73
74     """
75     Part 1: Linear search on 100 integers
76     """
77     arr = search.generate_random_list(100) #generate
        the list of 100 random integers
78     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
79
80     """
81     Part 2: Same as 1, but return number of checks
82     """
83     total = 0
84     tests = 100
85
86     for _ in range(tests):
87         arr = search.generate_random_list(100) #
        generate the list of 100 random integers
88         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
```

```

89     print("Average number of checks for 100 tests:"
, total / tests) #print the average number of checks
it took to find 42 for 100 tests
90
91     """
92     Part 3: Uses the recursive search
93     """
94     arr = sorted(search.generate_random_list(100)) #
generate the list of 100 random integers and sort it
for binary search
95     index = search.binary_search_recursive(arr, 42,
0, len(arr) - 1) #search for the value 42 using
binary search and get the index of 42
96     print("Index of 42 in sorted array:", index) #
print the index of 42 in the sorted array
97
98     """
99     Part 4: Tests
100
101     At what number of queries does Sorting + Binary
Search start to show an advantage over Linear Search
?:
102     Sorting + binary search shows an advantage at
around 500 queries
103     """
104     n = 1000
105     arr = search.generate_random_list(n) #generate
the list of n random integers
106
107     for multiplier in [0.5, 1, 2, 4]: #I used a "
multiplier" to avoid rewriting the code for each
query
108         query_count = int(n * multiplier)
109         queries = []
110
111         for _ in range(query_count):
112             queries.append(random.choice(arr))
113

```

```
114         # Linear timing
115         start = time.perf_counter()
116         for q in queries:
117             search.linear_search_count(arr, q)
118         end = time.perf_counter()
119         linear_time = end - start
120
121         # Sorting + Binary timing
122         start = time.perf_counter()
123         sorted_arr = sorted(arr)
124         for q in queries:
125             search.binary_search_recursive(
sorted_arr, q, 0, len(sorted_arr) - 1)
126         end = time.perf_counter()
127         binary_time = end - start
128
129         print("\nQueries:", query_count)
130         print("Linear time:", linear_time)
131         print("Sorting + Binary time:", binary_time
    )
132
133 # only execute when you run this file directly
134 if __name__ == "__main__":
135     main()
```