I used Java for this homework because I’m more familiar with it

Deliverable 1: Write down the steps by using an algorithm. Define the steps by using pseudo code instead of using programming code.

```
int[] doubleLinearSearch(int target, int[] arr) {
      result = new array

      for (element in arr) {
            if (element is equal to target) {
                  array append elements index
            }

            if (array.length == 2) {
                  return result // early break
            }
      }

      return [-1]
}
```

Deliverable 2: Convert the algorithm into a function and use the function (as shown above) in the main function. Pass necessary parameters and verify the accuracy of your program. Name your program, doubleLinearSearch.cpp and attach this program with your submission. Attach a screenshot depicting a sample run of the program

```
array: [10, 50, 16, 1, 9, 15, 16, 20, 16, 2, 5]
target: 10
result: [-1]
array: [10, 50, 16, 1, 9, 15, 16, 20, 16, 2, 5]
target: 16
result: [2, 6]
```

Deliverable 3: We know, Big O is an abstract function that describes how fast the cost of a function increases as the size of the problem becomes large. The order given by Big O is a least upper bound on the rate of growth.
We say that a function T(n) has order O(f(n)) if there exist positive constants c and n0 such that: T(n) ≤ c * f(n) when n ≥ n0.
Calculate the Big O of your algorithm by summing up the Big O of the individual statements used in your program. Find the equation T(n) of the program and furthermore, find the equation f(n), c, and n0 in order to determine the Big O of your algorithm.

$$t(n) = 1 + (n+1) + 1 + 1 + 1 + 1$$
$$= 6 + n$$
$$= O(n)$$

$f(n) = n$

$c = 1$

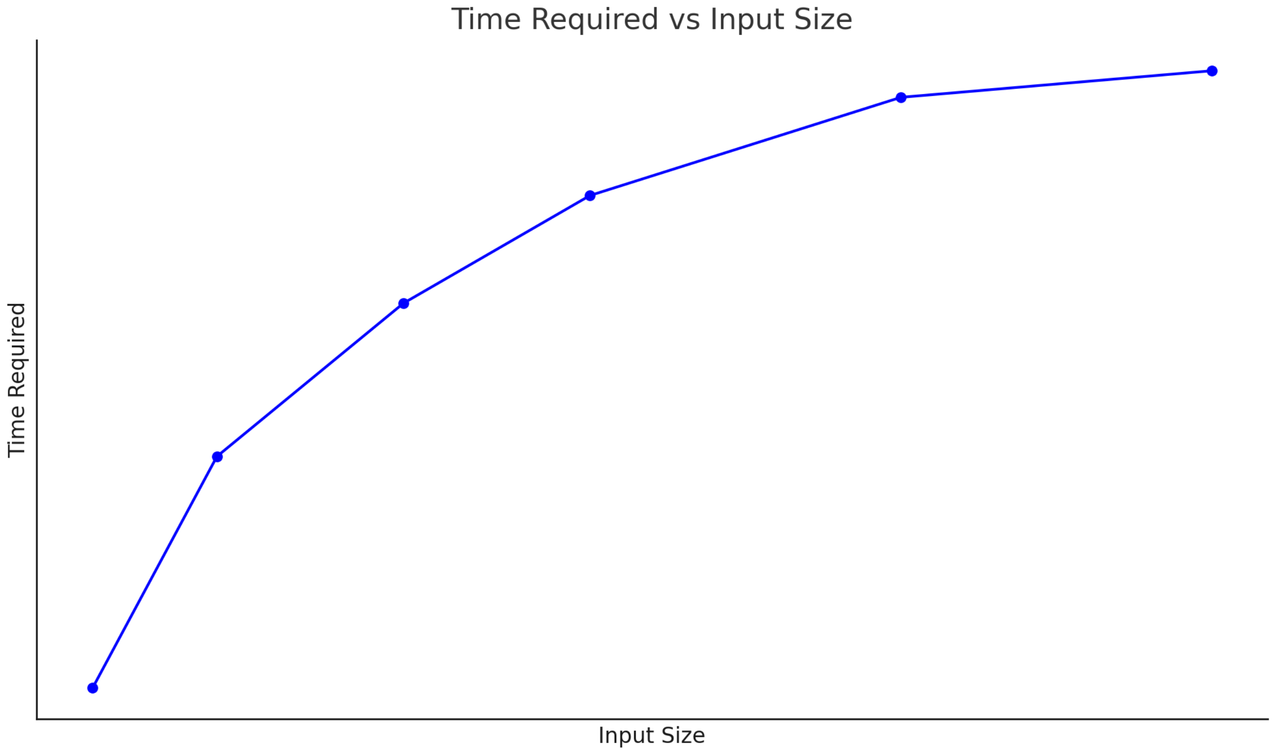
$n_0 = 1$

Deliverable 4:

```
10K: {hits=138, misses=862, averageSteps=9385, minSteps=1665}
20K: {hits=380, misses=620, averageSteps=17062, minSteps=1052}
35K: {hits=690, misses=310, averageSteps=22141, minSteps=298}
50K: {hits=845, misses=155, averageSteps=25713, minSteps=80}
75K: {hits=960, misses=40, averageSteps=28968, minSteps=983}
100K: {hits=990, misses=10, averageSteps=29849, minSteps=181}
125K: {hits=998, misses=2, averageSteps=29967, minSteps=242}
150K: {hits=999, misses=1, averageSteps=29611, minSteps=344}
```

Input size	Hits	Misses	Average steps	Min Steps
10K	157	843	9385	1665
20K	380	620	17062	1052
35K	690	310	22141	298
50K	845	155	25713	80
75K	960	40	28968	983
100K	990	10	29849	181

Deliverable 5:



This looks more like a logN than linear, but this is because of the early break. Without the early break it would be linear. The early break does not change big O notation, because it calculates the worst case.
If you run the simulation for longer, you can see, that at average the element is found two times after 30000 Steps