

Deliverable 1: pseudo code algorithm for double linear search

Return type Vector of integers, Function name (vector with const and reference, search_value)

Size_t vector_size := size of vector

Initialize an empty vector resultant_vector

for i from 0 to less than vector_size do

if vec[i] is equal to search_value then

Append i to resultant_vector

end if

if size of resultant_vector is equal to 2 then

break

end if

end for

if size of resultant_vector is less than 2 then

Clear resultant_vector

Append -1 to resultant_vector

end if

return resultant_vector

Deliverable 2: Attachment of Screenshot of doublelinearSearch.cpp

```

1 // This program find the index positions of a search value but until its 2nd occurrence
2 // Rahul Chaudhari
3
4 #include <iostream>
5 #include <vector>
6
7 using namespace std;
8
9 /**
10  * This function does the double linear search in an array
11  * @param vec is the vector that is passed by reference which has all the values
12  * @param search_value the value which is to be search
13  * @return a vector of integer with the index position of each vector
14  */
15 vector<int> Double_Linear_Search(const vector<int> &vec, int search_value);
16
17 int main()
18 {
19     vector<int> numbers {10, 50, 10, 1, 9, 15, 10, 20, 10, 2, 5};
20     int search_value = 10;
21     vector<int> search_position = Double_Linear_Search(numbers, search_value);
22     for (int i = 0; i < search_position.size(); i++)
23     {
24         cout << search_position.at(i) << " ";
25     }
26     return 0;
27 }
28
29 vector<int> Double_Linear_Search(const vector<int> &vec, int search_value)
30 {
31     size_t vector_size = vec.size();
32     vector<int> resultant_vector;
33     for (int i = 0; i < vector_size; i++)
34     {
35         if (vec.at(i) == search_value)
36         {
37             resultant_vector.push_back(i);
38             if (resultant_vector.size() == 2)
39             {
40                 break; // as soon as the 2nd value is found jump out of the loop
41             }
42         }
43     }
44 }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Code

tempCodeRunnerFile.cpp:1:1: error: 'resultant_vector' does not name a type
1 | resultant_vector.clear();

[Done] exited with code=1 in 0.1 seconds

[Running] cd "C:\Users\rjshaw\OneDrive\Documents\CS310\" && g++ main.cpp -o main && "C:\Users\rjshaw\OneDrive\Documents\CS310\main
-1
[Done] exited with code=0 in 0.721 seconds

[Running] cd "C:\Users\rjshaw\OneDrive\Documents\CS310\" && g++ doubleLinearSearch.cpp -o doubleLinearSearch && "C:\Users\rjshaw\OneDrive\Documents\CS310\doubleLinearSearch
-1
[Done] exited with code=0 in 0.669 seconds

[Running] cd "C:\Users\rjshaw\OneDrive\Documents\CS310\" && g++ doubleLinearSearch.cpp -o doubleLinearSearch && "C:\Users\rjshaw\OneDrive\Documents\CS310\doubleLinearSearch
-1
[Done] exited with code=0 in 0.621 seconds

[Running] cd "C:\Users\rjshaw\OneDrive\Documents\CS310\" && g++ doubleLinearSearch.cpp -o doubleLinearSearch && "C:\Users\rjshaw\OneDrive\Documents\CS310\doubleLinearSearch
2 5
[Done] exited with code=0 in 0.639 seconds

Deliverable 3: Calculating the Big O for the Algorithm.

| Code | Cost | Times |
|---|--------------|-------|
| <code>vector<int> Double_Linear_Search(const vector<int> &vec, int search_value)</code> | | |
| <code>{</code> | | |
| <code>size_t vector_size = vec.size();</code> | C1, | 1 |
| | C2 | 1 |
| <code>vector<int> resultant_vector;</code> | C3 | 1 |
| <code>for(int i = 0; i < vector_size; i++)</code> | C4 (i = 0), | 1 |
| <code>{</code> | C5(i<vec..) | n + 1 |
| <code>if(vec.at(i) == search_value)</code> | C6, (.at) | n |
| | C7 | n |
| <code>{</code> | | |
| <code> resultant_vector.push_back(i);</code> | C8 | n |
| <code>}</code> | | |
| <code> if (resultant_vector.size() == 2)</code> | C9, (.size) | n |
| | C10 | n |
| <code>{</code> | | |
| <code> break;</code> | C11 | 1 |
| <code>}</code> | | |
| <code>}</code> | C12 (i++) | n |
| <code>if (resultant_vector.size() < 2)</code> | C13, (.size) | 1 |
| | C14 | 1 |
| <code>{</code> | | |
| <code> resultant_vector.clear();</code> | C15 | 1 |
| <code> resultant_vector.push_back(-1);</code> | C16 | 1 |
| <code>}</code> | | |
| <code>return resultant_vector;</code> | C17 | 1 |
| <code>}</code> | | |

Cost Function:

$C1+C2+C3+C4+C5+C6+C7+C8+C9+C10+C11+C12+C13+C14+C15+C16+C17$

$T(n) = 1+1+1+1+n+1+n+n+n+n+n+1+n+1+1+1+1+1$

$T(n) = 7n + 11$

Here, our $T(n) = 7n+11$ and has order of $O(n)$ because

$7n+11 \leq 10 * n$ where $n > n_0$

For any values of $n \geq 10$, $c = 10$, $n_0 = 1$

Here, $f(n) = n$

Hence the Big O would be $O(f(n)) = O(n)$.

Deliverable 4: Screenshot of the Code with Simulation Data

The screenshot shows a C++ code editor with the following code in `simulDoubleLinearSearch.cpp`:

```

1 // Rahul Chaudhari
2 // Simulation of DoubleLinearSearch Algorithm
3 #include <iostream>
4 #include <cstdlib>
5 #include <ctime>
6 #include <vector>
7
8 using namespace std;
9
10 const int NOT_FOUND = -1;
11
12 /**
13  * This function generate random values
14  * @param min the minimum range for random value
15  */
16 int generateRandomValue(int min) {
17     return min + rand() % (100000 - min);
18 }
19
20 int doubleLinearSearch(vector<int> arr, int target) {
21     for (int i = 0; i < arr.size(); i++) {
22         if (arr[i] == target) {
23             return i;
24         }
25     }
26     return NOT_FOUND;
27 }
28
29 int main() {
30     srand(time(0));
31     int inputSize;
32     int hits = 0;
33     int misses = 0;
34     int minSteps = 0;
35     double avgSteps = 0.0;
36
37     for (int i = 0; i < 6; i++) {
38         inputSize = generateRandomValue(10000);
39         vector<int> arr(inputSize);
40         for (int j = 0; j < arr.size(); j++) {
41             arr[j] = generateRandomValue(100000);
42         }
43         int target = generateRandomValue(100000);
44         int index = doubleLinearSearch(arr, target);
45         if (index != NOT_FOUND) {
46             hits++;
47         } else {
48             misses++;
49         }
50         minSteps = min(minSteps, i + 1);
51         avgSteps += i + 1;
52     }
53     avgSteps /= 6;
54     cout << "Input Size (N): " << inputSize << endl;
55     cout << "Average Number of Steps: " << avgSteps << endl;
56     cout << "Hits: " << hits << endl;
57     cout << "Misses: " << misses << endl;
58     cout << "Minimum Steps: " << minSteps << endl;
59     cout << "-----" << endl;
60 }

```

The terminal output shows the simulation results for different input sizes:

```

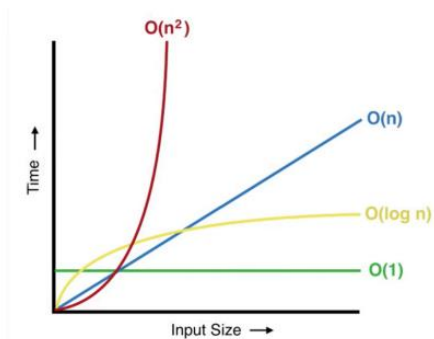
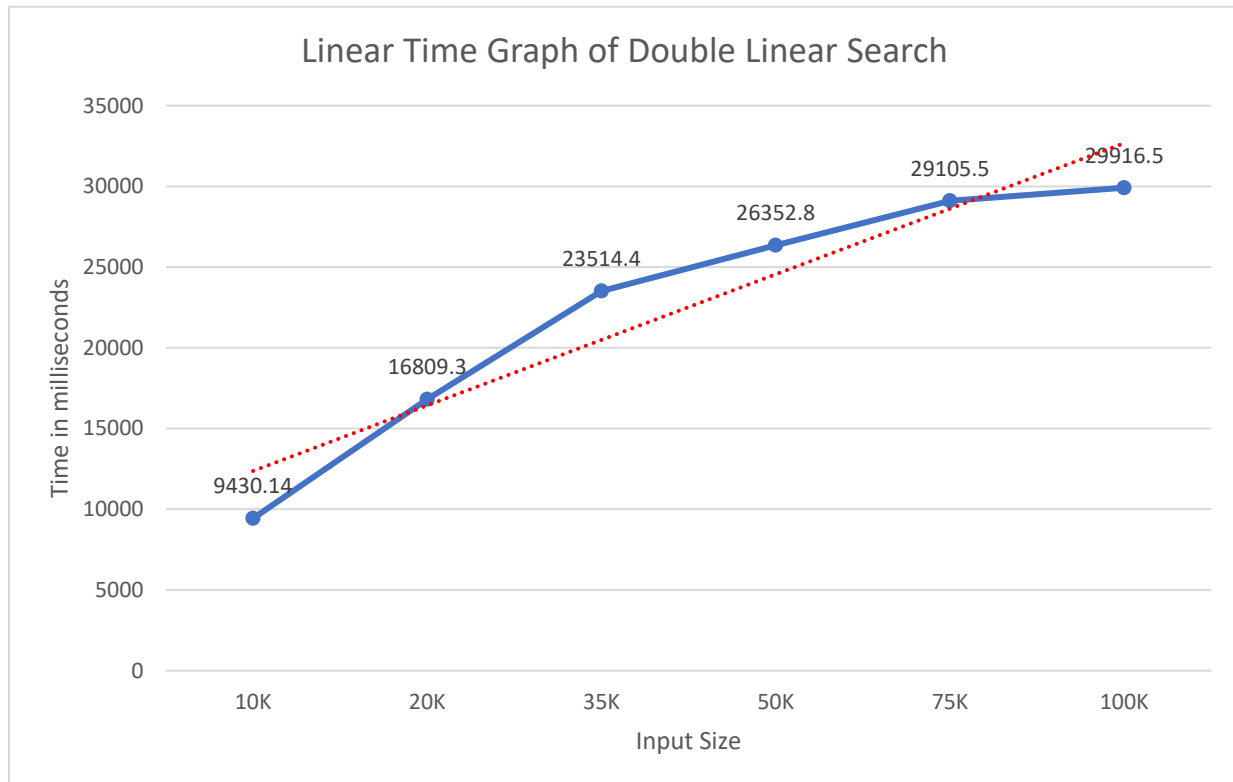
[Running] cd "C:\Users\rjasa\OneDrive\Documents\CS310" && g++ simulDoubleLinearSearch.cpp -o simulDoubleLinearSearch && "C:\Users\rjasa\OneDrive\Documents\CS310\simulDoubleLinearSearch
Input Size (N): 10000
Average Number of Steps: 9430.14
Hits: 149
Misses: 851
Minimum Steps: 140
-----
Input Size (N): 20000
Average Number of Steps: 16809.3
Hits: 385
Misses: 615
Minimum Steps: 370
-----
Input Size (N): 35000
Average Number of Steps: 23514.4
Hits: 671
Misses: 329
Minimum Steps: 333
-----
Input Size (N): 50000
Average Number of Steps: 26352.8
Hits: 844
Misses: 156
Minimum Steps: 605
-----
Input Size (N): 75000
Average Number of Steps: 29105.5
Hits: 944
Misses: 56
Minimum Steps: 1287
-----
Input Size (N): 100000
Average Number of Steps: 29916.5
Hits: 984
Misses: 16
Minimum Steps: 717
-----
[Done] exited with code=0 in 9.775 seconds

```

This Table represents the hits, misses, min steps and average steps for the double linear search algorithm for different input sizes ranging from 10K to 100K. We can also notice the average steps increasing as the input sizes is increasing.

| Input Size | Hits | Misses | Min Steps | Average Steps |
|------------|------|--------|-----------|---------------|
| 10K | 149 | 851 | 140 | 9430.14 |
| 20K | 385 | 615 | 370 | 16809.3 |
| 35K | 671 | 329 | 333 | 23514.4 |
| 50K | 844 | 156 | 605 | 26352.8 |
| 75K | 944 | 56 | 1287 | 29105.5 |
| 100K | 984 | 16 | 717 | 29916.5 |

Deliverable 5: Graph of the Data



In this graph we can notice that the red trendline is going straight, which means the overall function is following a linear fashion. If we compare it with the Big O graph of different functions, we can see that it clearly resembles $O(n)$, which we also got from the cost function calculation. Thus, this graph comparison, the cost function and the simulation verifies that the algorithm works in linear time.