Topic:

1. Linear search
2. Bubble sort

Name: Shimul Sutradhar
ID: 191-15-12614
Github: [algorithm lab]

Linear search

Lineare search is an algorithm for finding an element in the array. It starts searching from the first element and check every element one by one.

```
#include<iostream>
using namespace std;
int main(){
        bool flag = false;
        int arr[5] = {1,0,2,5,6};
        for(int i = 0; i < 5; i++){
            if(arr[i] == 5){
                flag = true;
                break;
            }
        }
        if(flag)
            cout << 5 << " found\n";
        else
            cout << "Not found\n";
        return 0;
}
```

Code 1.0: Linear search.

Time Complexity: O(n)

Analysis:

For the worst case, Let's take an array length n. For finding an element it will start checking elements from index 1. If finding element, not in the array it will check n element one by one.

So, the time complexity of the linear search is O(n).

For the best case, it will take O(1) time if the element was in index 0.

Bubble sort

Bubble sort is one of the most famous and simple algorithms for sorting less amount of data. Bubble sort time complexity is O(n^2). If we do as usual bubble sort it will take O(n^2) in the best case (array was sorted).

Pseudocode:

```
int  a[n]
for i from 1 to N
        for j from 0 to N-1
                if a[j]>a[j+1]
                  swap(a[j], a[j+1])
        end second loop
end first loop
```

By doing some optimization for the best case we can make the best case time complexity O(n)

Pseudocode:

```
int  a[n]
swapped = true
i = 1
while swapped
        swapped = false
        for j from 0 to N - 1
                if a[j] > a[j + 1]
                        swap( a[j], a[j + 1] )
                        swapped = true
        end of 2nd loop
        if !swapped
                Break
end of first loop
```

We will do the further process in an optimized way.

Simulation:

Take an array, a[ ] = {5, 1, 4, 2, 8};

1st Iteration:
(5 1 4 2 8) –> (1 5 4 2 8), Here, the algorithm compares the first two elements and swaps since 5 > 1.
(1 5 4 2 8) –> (1 4 5 2 8), Swap since 5 > 4
(1 4 5 2 8) –> (1 4 2 5 8), Swap since 5 > 2
(1 4 2 5 8) –> (1 4 2 5 8), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

2nd Iteration:
(1 4 2 5 8) –> (1 4 2 5 8)
(1 4 2 5 8) –> (1 2 4 5 8), Swap since 4 > 2
(1 2 4 5 8) –> (1 2 4 5 8)
(1 2 4 5 8) –> (1 2 4 5 8)
Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one whole pass without any swap to know it is sorted.

3rd Iteration:
(1 2 4 5 8) –> (1 2 4 5 8)
(1 2 4 5 8) –> (1 2 4 5 8)
(1 2 4 5 8) –> (1 2 4 5 8)
(1 2 4 5 8) –> (1 2 4 5 8)

Code:

```
#include <bits/stdc++.h>
using namespace std;
int main(){
        const int n = 6;
        int arr[n] = {9,5,1,4,5,2};
        bool swaping = true;
        for(int i = 0; i < n - 1; i++){
                for(int j = 0; j < n - i - 1; j++){
                        if(arr[j] > arr[j + 1]){
                                int temp = arr[j];
```

```
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                    swaping = false;
                }
            }
        if(swaping)
            break;
        }
        for(int i = 0; i < n; i++)
            cout << arr[i] << ' ';
        return 0;
}
```

Time complexity:

For the first loop, every i 2nd loop iteration will be (i - 1). The first loop range was n - 1.
Value of n = 5

Value of i	2nd loop range
0	n - 1
1	n - 2
2	n - 3
3	n - 4
	$\frac{n*(n-1)}{2} - 1$

$$\frac{n*(n-1)}{2} - 1$$
$$\Rightarrow \frac{n*(n-1)-2}{2}$$
$$\Rightarrow \frac{n^2-n-2}{2}$$

Ignoring constant co-efficient, we get O(n^2)
It's for the worst case and average case.
For best case (if the array was sorted) time complexity O(n)