```
//NUMBER 1
//Esteron,Jenel F.
//CPE21S1
#include<iostream>

using namespace std;

void PrintArray(int array[], int size){
     int i=0;
     cout<<"<";
     while (i<10){
        cout<<array[i]<<", ";
        i++;
     }
     cout<<">";
}

void BubbleSort(int array[], int size){
  for (int step = 0; step < size; ++step) {
    for (int i = 0; i < size - step; ++i) {
      if (array[i] > array[i + 1]) {
        int temp = array[i];
        array[i] = array[i + 1];
        array[i + 1] = temp;
      }
    }
  }
}

void SearchArray(int array[], int size){
    int i=0, Num, index;
    bool found = false;
    cout<<"\nEnter Number:";
    cin>>Num;

    while(i<10){
        if(Num==array[i]){
            i = index;
            found = true;
        }
```

```
        else{
            found = false;
        }
        i++;
    }
    if(found==true){
      cout<<"Number is in the array";
    }else{
      cout<<"Number not found";
      SearchArray(array, size);
    }
}

int main(){
    int array[10] = {9945, 3118, 3399, 2001, 9983, 8641, 2557, 8742, 5957,
315};
    int size = 9;
    PrintArray(array, size);
    BubbleSort(array, size);
    cout<<"\nSorted Array: ";
    PrintArray(array, size);
    SearchArray(array,size);
    return 0;
}
```

```
<9945, 3118, 3399, 2001, 9983, 8641, 2557, 8742, 5957, 315, >
Sorted Array: <315, 2001, 2557, 3118, 3399, 5957, 8641, 8742, 9945, 9983, >
Enter Number:43
Number not found
Enter Number:8742
Number is in the array
```

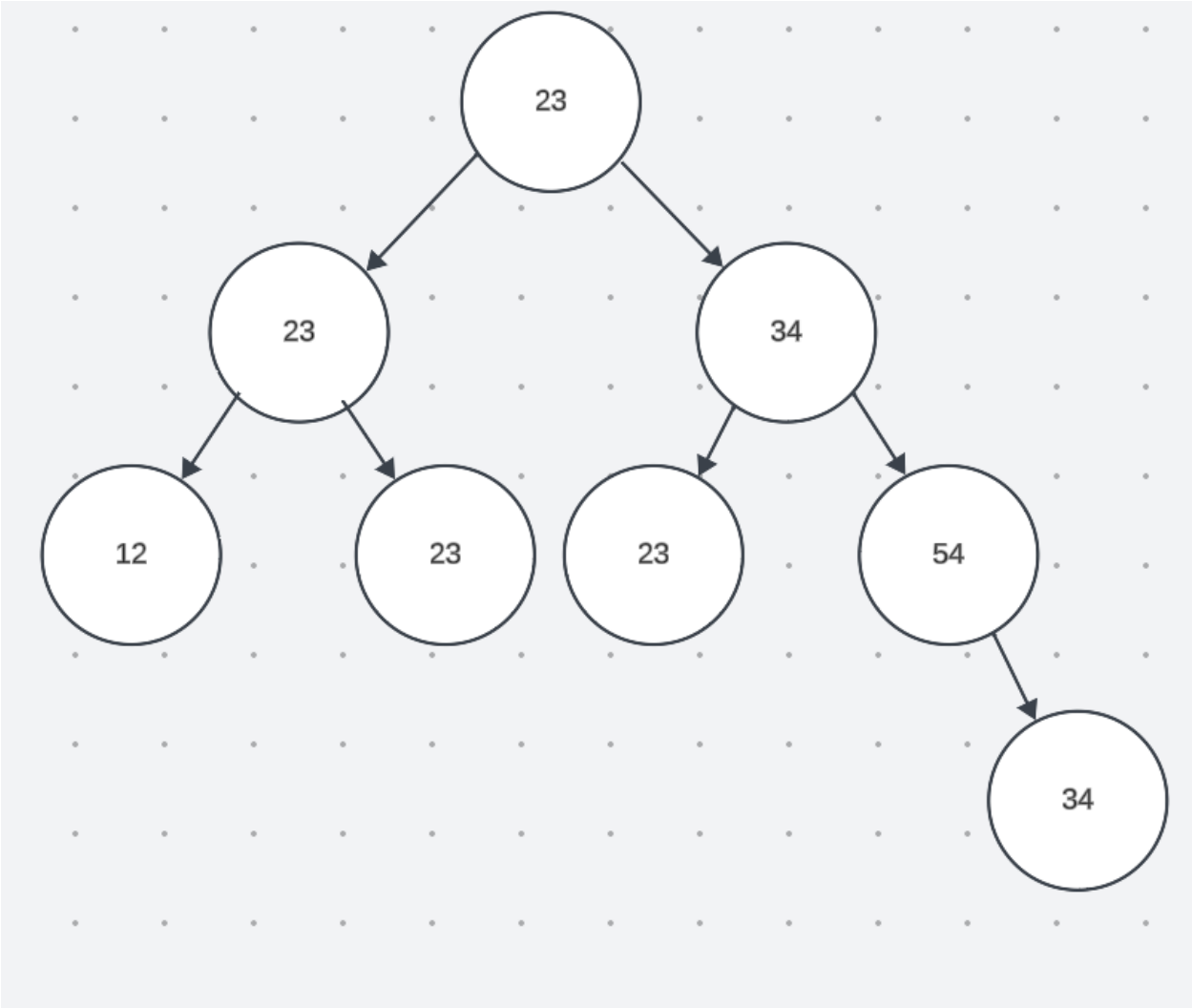
BubbleSort has the advantage of being a really simple and easy to understand sorting algorithm. Although it is slow it is very reliable in its own right.

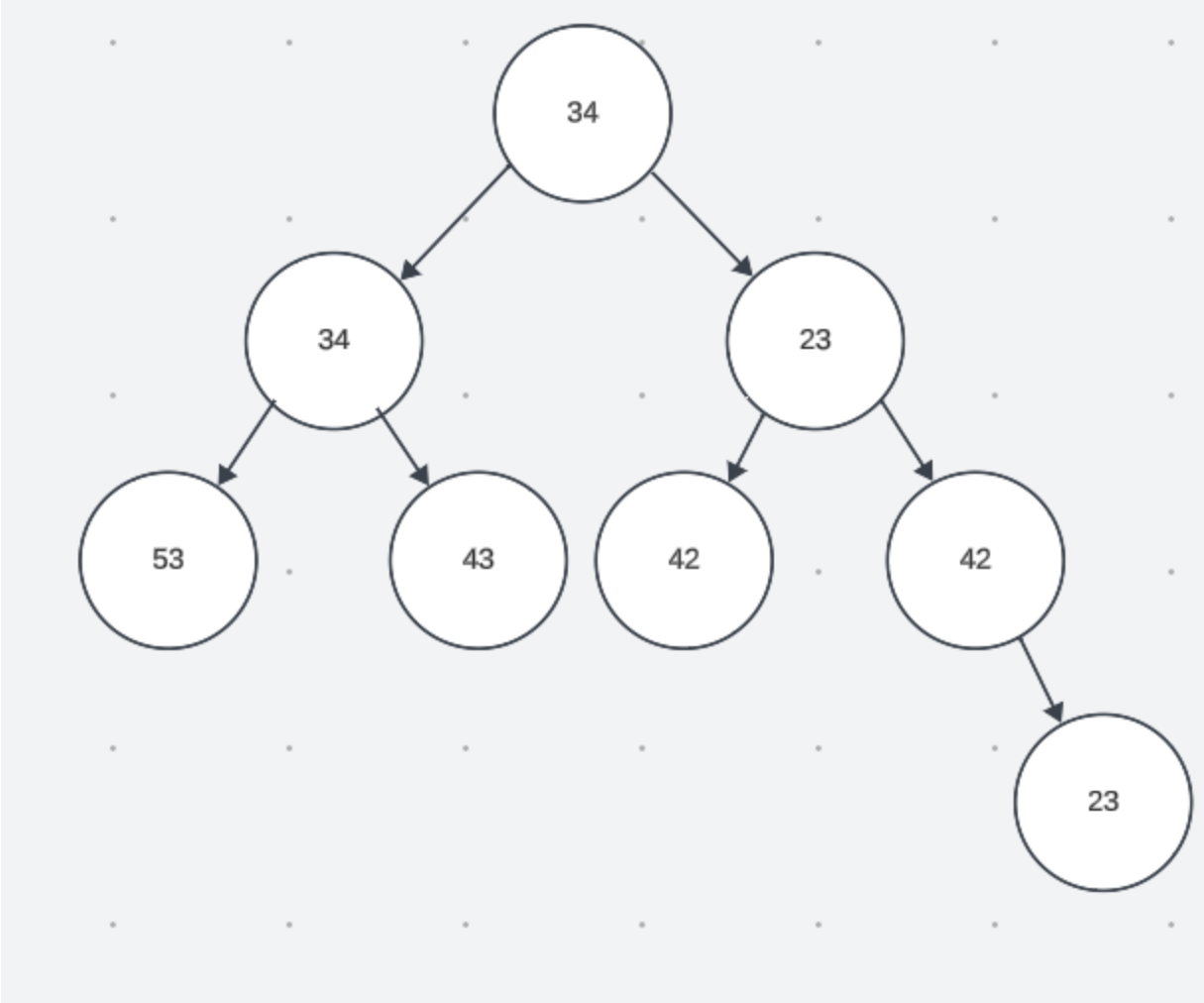
LinearSearch is similar to bubble sort in the sense that it is simple but as a search algorithm its advantage is that you are able to search data that are not sorted unlike binary search where the data should be sorted.

```cpp
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* left;
    Node* right;

    Node(int value) {
        data = value;
        left = nullptr;
        right = nullptr;
    }
};

Node* insertNode(Node* root, int value) {
    if (root == nullptr) {
        return new Node(value);
    }

    if (value < root->data) {
        root->left = insertNode(root->left, value);
    } else {
        root->right = insertNode(root->right, value);
    }

    return root;
}

void inOrderTraversal(Node* root) {
    if (root == nullptr) {
        return;
    }

    inOrderTraversal(root->left);
```

```
    cout << root->data << " ";
    inOrderTraversal(root->right);
}

int main() {
    Node* root = nullptr;

    while (true) {
        int value;
        cout << "Enter a value to insert into the tree (Enter y to stop):
";
        if (!(cin >> value)) {
            break;
        }

        root = insertNode(root, value);
    }

    cout << "In-order traversal of the tree: ";
    inOrderTraversal(root);
    cout << endl;

    return 0;
}
```

```
Enter a value to insert into the tree (or any non-integer to exit): 23
Enter a value to insert into the tree (or any non-integer to exit): 23
Enter a value to insert into the tree (or any non-integer to exit): 23
Enter a value to insert into the tree (or any non-integer to exit): 54
Enter a value to insert into the tree (or any non-integer to exit): 34
Enter a value to insert into the tree (or any non-integer to exit): 23
Enter a value to insert into the tree (or any non-integer to exit): y
In-order traversal of the tree: 12 23 23 23 23 34 34 54
```


23
23
34
12
23
23
54
34


34
34
23
53
43
42
42
23