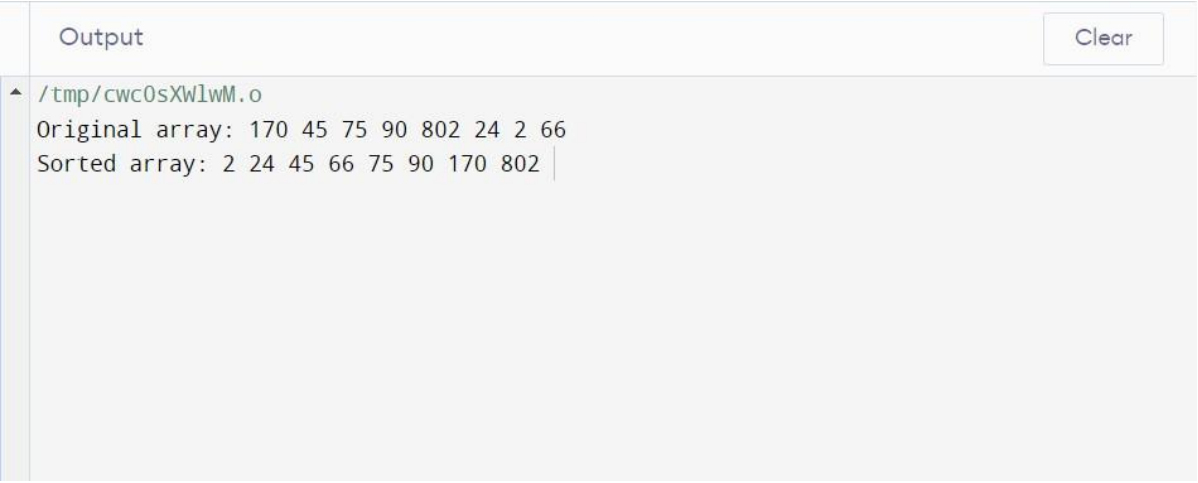
CODE 1- RADIX SORT USING ARRAYS



The time complexity of this algorithm is-

O(d * (n + b)) where n is number of elements to be sorted, maximum number of digits of a given number and b is the base of the numbering system (usually 10 in decimal).

It's important to note that when the base (b) is a constant, and the number of digits (d) is a small constant as well (which is often the case in real-world scenarios), the time complexity simplifies to O(n). However, if the numbers have a large number of digits, the time complexity can become significant.

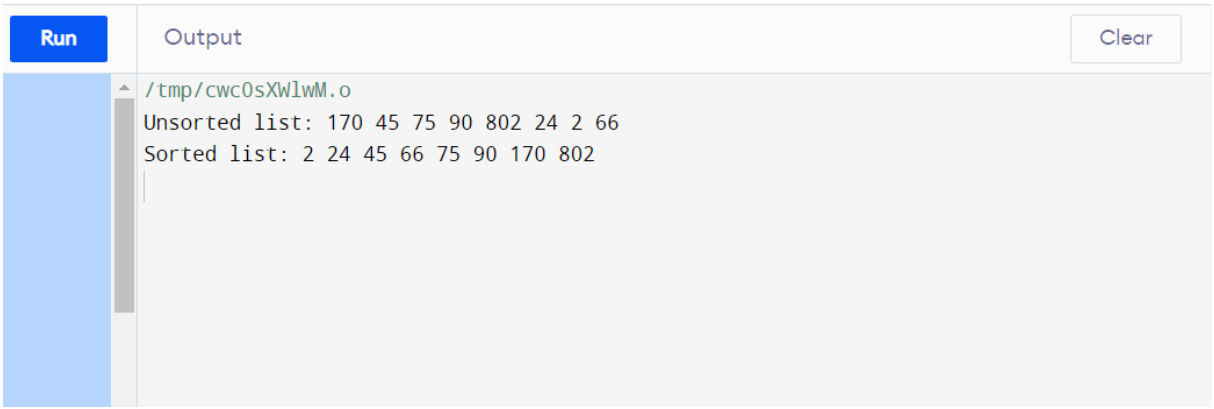
```
#include<bits/stdc++.h>  // Header
#include <iostream>
#include <vector>
using namespace std;


typedef long long int ll;
typedef long double lld;
typedef std::vector<ll> vll;
typedef std::unordered_map<ll, ll> umll;
typedef std::map<ll, ll> mll;
typedef unordered_map<char, ll> umcll;



int getMax(const std::vector<int>& arr) {
```

```
int max = arr[0];
for (int i = 1; i < arr.size(); ++i) {
    if (arr[i] > max) {
        max = arr[i];
    }
```

```
    }
    return max;
}

void countingSort(std::vector<int>& arr, int exp) {
    int n = arr.size();
            vector<int> output(n);
            vector<int> count(10, 0);

    for (int i = 0; i < n; ++i) {
        count[(arr[i] / exp) % 10]++;
    }

    for (int i = 1; i < 10; ++i) {
        count[i] += count[i - 1];
    }

    for (int i = n - 1; i >= 0; --i) {
        output[count[(arr[i] / exp) % 10] - 1] = arr[i];
        count[(arr[i] / exp) % 10]--;
    }

    for (int i = 0; i < n; ++i) {
        arr[i] = output[i];
    }
}

void radixSort(std::vector<int>& arr) {
    int max = getMax(arr);

    for (int exp = 1; max / exp > 0; exp *= 10) {
```

```
        countingSort(arr, exp);
    }
}

int main() {
        vector<int> arr = {170, 45, 75, 90, 802, 24, 2, 66};

    std::cout << "Original array: ";
    for (int num : arr) {
        cout << num << " ";
    }

    radixSort(arr);

    std::cout << "\nSorted array: ";
    for (int num : arr) {
        cout << num << " ";
    }

    return 0;
}
```

CODE 2- RADIX SORT USING LINKED LIST



TIME COMPLEXITY OF RADIX SORT USING LINKED LIST IS-

The time complexity of distributing elements for all d digit positions is O(d * n).
Adding up the complexities:

- **Creating Linked Lists: O(b)**
- **Distributing and Collecting Elements: O(d * n)**
- **Total time complexity: O(b + d * n)**

In the worst case, where d is the number of digits in the largest number and b is the base, the time complexity simplifies to:
O(d * (n + b))
As before, if the base (b) and the number of digits (d) are constants, and if you have efficient node manipulation and memory management in the linked list implementation, the time complexity could be practically linear, i.e., O(n). However, the constants and implementation details can still impact the real-world performance of the algorithm.

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

typedef long long int ll;
typedef long double lld;
typedef std::vector<ll> vll;
typedef std::unordered_map<ll, ll> umll;
typedef std::map<ll, ll> mll;
typedef unordered_map<char, ll> umcll;
```

```
struct Node {
    int data;
    Node* next;
    Node(int d) {
        data = d;
        next = nullptr;
```

```
    }
};
void insert(Node*& head, int data) {
    Node* newNode = new Node(data);
    newNode->next = head;
    head = newNode;
}

void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        std::cout << temp->data << " ";
        temp = temp->next;
    }
    cout << std::endl;
}

int getMaxDigits(int arr[], int size) {
    int maxNum = arr[0];
    for (int i = 1; i < size; ++i) {
        if (arr[i] > maxNum) {
            maxNum = arr[i];
        }
    }

    int digits = 0;
    while (maxNum > 0) {
        maxNum /= 10;
        ++digits;
    }
    return digits;
```

```
}

void radixSort(Node*& head, int digits) {
    for (int d = 1; d <= digits; ++d) {
        Node* buckets[10] = {nullptr};

        // Distribute elements into buckets
        Node* current = head;
        while (current != nullptr) {
            int digit = (current->data / (int)pow(10, d - 1)) % 10;
            Node* newNode = new Node(current->data);
            newNode->next = buckets[digit];
            buckets[digit] = newNode;
            current = current->next;
        }

        // Collect elements from buckets
        Node* tempHead = nullptr;
        for (int i = 9; i >= 0; --i) {
            if (buckets[i] != nullptr) {
                Node* temp = buckets[i];
                while (temp != nullptr) {
                    Node* nextNode = temp->next;
                    temp->next = tempHead;
                    tempHead = temp;
                    temp = nextNode;
                }
            }
        }

        head = tempHead;
```

```
    }
}

int main() {
    int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
    int size = sizeof(arr) / sizeof(arr[0]);

    Node* head = nullptr;
    for (int i = 0; i < size; ++i) {
        insert(head, arr[i]);
    }

    int digits = getMaxDigits(arr, size);

    radixSort(head, digits);
    cout << "Unsorted list: ";
    for(int i=0; i<8;i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
    cout << "Sorted list: ";
    printList(head);

    return 0;
}
```