# AVANISH RAJ SRIVASTAVA BT22CSH031

DSA ASSIGNMENT 2

# Q1)

#include <iostream> #include <algorithm>

using namespace std;

void countingSort(int arr[], int n, int exp) { int output[n];

int count[10] = {0};

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

}

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

}

for (int i = n - 1; i >= 0; --i) { int index = arr[i] / exp;

output[count[index % 10] - 1] = arr[i]; count[index % 10]--;

}

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

}

}

void radixSort(int arr[], int n) {

int max\_value = \*max\_element(arr, arr + n); int exp = 1;

while (max\_value / exp > 0) { countingSort(arr, n, exp); exp \*= 10;

}

}

int main() {

int input\_list[] = {136, 487, 358, 469, 570, 247, 598, 639, 205, 609};

int n = sizeof(input\_list) / sizeof(input\_list[0]); radixSort(input\_list, n);

cout << "Sorted list:"; for (int i = 0; i < n; ++i) {

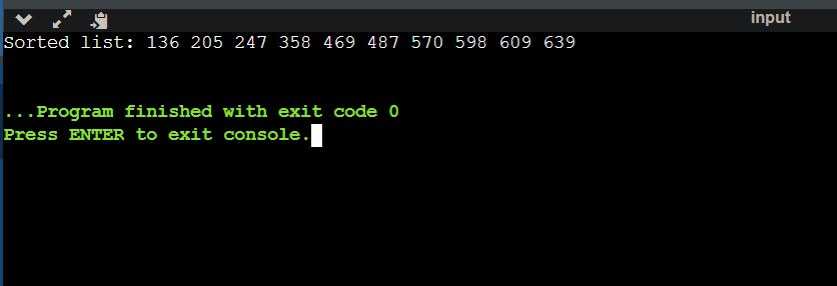
cout << " " << input\_list[i];

}

cout << endl;

return 0;

}



# TIME COMPLEXITY ANALYSIS

* BEST CASE : O(n\*k)
* AVERAGE CASE : O(n\*k)
* WORST CASE : O(n\*k)

Where n = number of elements and k = number of passes

# Q2)

#include <iostream> #include <cmath>

using namespace std; struct Node {

int data;

Node\* next;

Node(int val) : data(val), next(nullptr) {}

};

void insert(Node\*& head, int val) { Node\* newNode = new Node(val); if (!head) {

head = newNode;

} else {

Node\* temp = head; while (temp->next) {

temp = temp->next;

}

temp->next = newNode;

}

}

int getDigit(int num, int place) { return (num / place) % 10;

}

void countingSort(Node\*& head, int exp) { if (!head) return;

int count[10] = {0}; Node\* current = head; Node\* output = nullptr;

while (current) {

int index = getDigit(current->data, exp); count[index]++;

current = current->next;

}

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

}

current = head; while (current) {

int index = getDigit(current->data, exp); Node\* newNode = new Node(current->data); newNode->next = output;

output = newNode; count[index]--;

current = current->next;

}

current = head;

Node\* sortedCurrent = output;

while (sortedCurrent) {

current->data = sortedCurrent->data; current = current->next; sortedCurrent = sortedCurrent->next;

}

while (output) {

Node\* temp = output; output = output->next; delete temp;

}

}

void radixSort(Node\*& head) { int max\_val = 0;

Node\* current = head;

while (current) {

max\_val = max(max\_val, current->data); current = current->next;

}

int exp = 1;

while (max\_val / exp > 0) { countingSort(head, exp); exp \*= 10;

}

}

int main() {

Node\* input\_list = nullptr;

int elements[] = {136, 487, 358, 469, 570, 247, 598, 639, 205, 609};

int n = sizeof(elements) / sizeof(elements[0]); for (int i = 0; i < n; ++i) {

insert(input\_list, elements[i]);

}

radixSort(input\_list);

cout << "Sorted list:"; Node\* current = input\_list; while (current) {

cout << " " << current->data; current = current->next;

}

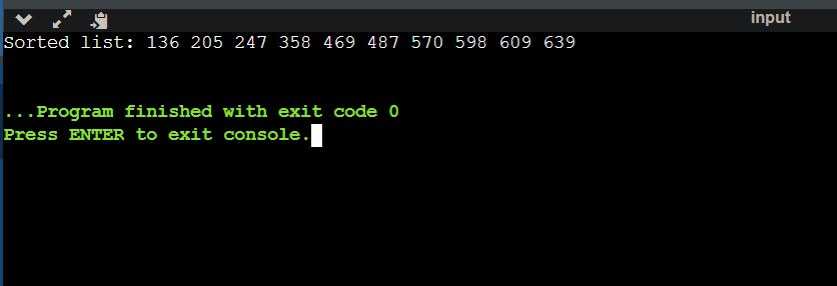
cout << endl;

while (input\_list) {

Node\* temp = input\_list; input\_list = input\_list->next; delete temp;

}

return 0;

}

# TIME COMPLEXITY ANALYSIS

* BEST CASE : O(n\*k)
* AVERAGE CASE : O(n\*k)
* WORST CASE : O(n\*k)

Where n = number of elements and k = number of passes