Name : Ankush jha

Roll : BT22CSH023

Q1)(a) Implement radix sort algorithm using arrays for the input list given below. Deduce the timecomplexity T(n) for the best, worst and average cases.

Code 👍

#include <stdio.h>

int maxele(int arr[], int n) {

int maxi = arr[0];

for (int i = 1; i < n; i++) {

if (maxi < arr[i]) maxi = arr[i];

}

return maxi;

}

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

int maxx = maxele(arr, n) + 1;

int count[maxx] ;

for(int i=0;i<maxx;i++){

count[i] = 0;

}

int sorted[n];

for (int i = 0; i < n; i++) {

count[arr[i]]++;

}

for (int i = 1; i < maxx; i++) {

count[i] += count[i - 1];

}

for (int i = n - 1; i >= 0; i--) {

sorted[count[arr[i]] - 1] = arr[i];

count[arr[i]]--;

}

for (int i = 0; i < n; i++) {

printf("%d ", sorted[i]);

}

printf("\n");

}

int main() {

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

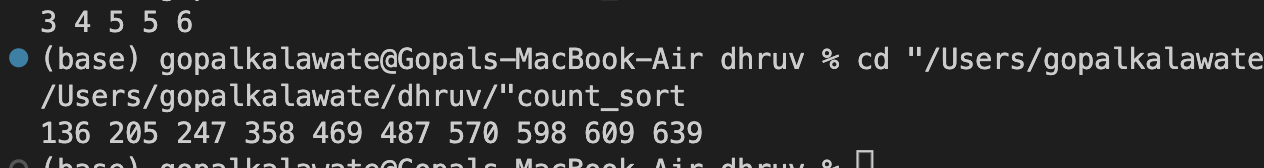
int n = sizeof(arr) / sizeof(arr[0]);

countSort(arr, n);

return 0;

}

Output 👍



**TIME COMPLEXITY**

The time complexity of Radix Sort is dependent on the number of digits in the maximum number and the number of elements in the array.

Let us say that there are k digits in the maximum number and the no. of elements in the array is n. Then the count sort algorithm will be executed for k times in the radix sort function and in that count sort function the loop will be executed n times. Thus,

- Best Case: O(k \* n)

- Average Case: O(k \* n)

- Worst Case: O(k\* n)

Q2)Use linked list for implementation of Radix sort for the same elements given above. Deducethetime complexity T(n) for the best, worst and average cases.

Code:

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node \*next;

} Node;

void insert(Node \*\*head, int data) {

Node \*newNode = (Node \*)malloc(sizeof(Node));

newNode->data = data;

newNode->next = NULL;

if (\*head == NULL) {

\*head = newNode;

} else {

Node \*current = \*head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

}

}

void printList(Node \*head) {

Node \*current = head;

while (current != NULL) {

printf("%d ", current->data);

current = current->next;

}

}

void clearList(Node \*\*head) {

Node \*current = \*head;

while (current != NULL) {

Node \*temp = current;

current = current->next;

free(temp);

}

\*head = NULL;

}

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

int max = arr[0];

for (int i = 1; i < n; i++) {

if (arr[i] > max) {

max = arr[i];

}

}

int digits = 0;

while (max > 0) {

digits++;

max /= 10;

}

for (int exp = 1; digits > 0; exp \*= 10, digits--) {

Node \*buckets[10] = {NULL};

for (int i = 0; i < n; i++) {

int index = (arr[i] / exp) % 10;

insert(&buckets[index], arr[i]);

}

printf("Sorted Array (Phase %d):\n", exp);

int index = 0;

for (int i = 0; i < 10; i++) {

Node \*current = buckets[i];

while (current != NULL) {

arr[index++] = current->data;

Node \*temp = current;

current = current->next;

free(temp);

}

}

for (int i = 0; i < n; i++) {

printf("%d ", arr[i]);

}

printf("\n");

}

printf("Final Sorted Array:\n");

for (int i = 0; i < n; i++) {

printf("%d ", arr[i]);

}

printf("\n");

}

int main() {

int n;

printf("Enter the number of elements:\n");

scanf("%d", &n);

int arr[n];

printf("Enter the array elements:\n");

for (int i = 0; i < n; i++) {

scanf("%d", &arr[i]);

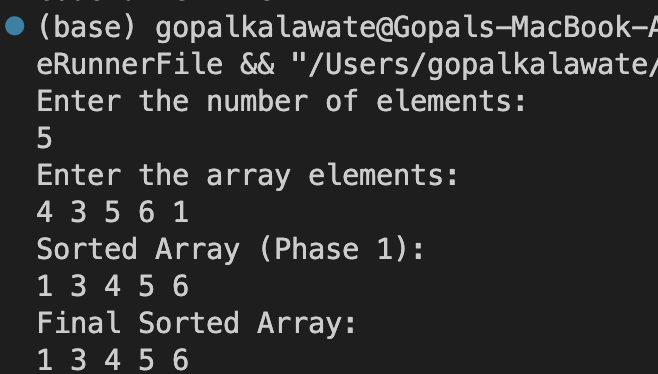
}

radixSort(arr, n);

return 0;

}

Output:



**TIME COMPLEXITY:**

The time complexity of this program, which implements Radix Sort using linked lists, is as follows:

- Best Case: O(k \* n)

- Average Case: O(k \* n)

- Worst Case: O(k \* n)

- Best Case: The linked-list-based Radix Sort still goes through all `k` digits for each of the `n` elements. Thus, the time complexity remains O(k\* n). Here also, ‘k’ refers to the number of digits in the maximum number.

- Average Case: Similar to the best case, the average case time complexity remains O(k \* n).

- Worst Case: The worst case time complexity also remains O(k \* n). This occurs when the digits are distributed in a way that maximizes the number of passes through the digits. Each pass involves processing all `n` elements, distributing them into linked lists, and collecting them back. The number of passes is determined by the number of digits in the maximum number.