Design & Analysis of Algorithm
Assignment - 3

Name - Hemal Chudasama
Roll No. - 21BCP152
Div - 3 , G - 5

Implementation of City Data Base

- **Program - 1[Linked-list Implementation] :-**

```
// Linked list Implementation -

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

struct city {
    char name[100];
    int x;
    int y;
    struct city *next;
};

struct city *head = NULL;

// Insert a record
void insert(char name[], int x, int y) {
    struct city *newCity = (struct city*)
malloc(sizeof(struct city));
    strcpy(newCity->name, name);
    newCity->x = x;
    newCity->y = y;
    newCity->next = head;
    head = newCity;
}

// Delete a record by name
void deleteByName(char name[]) {
    struct city *current = head, *prev = NULL;
    while (current != NULL) {
        if (strcmp(current->name, name) == 0) {
            if (prev == NULL) {
                head = current->next;
            } else {
                prev->next = current->next;
            }
            free(current);
```

```
            return;
        }
        prev = current;
        current = current->next;
    }
}

// Delete a record by coordinate
void deleteByCoordinate(int x, int y) {
    struct city *current = head, *prev = NULL;
    while (current != NULL) {
        if (current->x == x && current->y == y) {
            if (prev == NULL) {
                head = current->next;
            } else {
                prev->next = current->next;
            }
            free(current);
            return;
        }
        prev = current;
        current = current->next;
    }
}

// Search a record by name
struct city* searchByName(char name[]) {
    struct city *current = head;
    while (current != NULL) {
        if (strcmp(current->name, name) == 0) {
            return current;
        }
        current = current->next;
    }
    return NULL;
}

// Search a record by coordinate
struct city* searchByCoordinate(int x, int y) {
    struct city *current = head;
    while (current != NULL) {
        if (current->x == x && current->y == y) {
            return current;
        }
        current = current->next;
```

```
    }
    return NULL;
}

// Print all records within a given distance of a
specified point
void printByDistance(struct city *list, int x, int y, int
distance) {
    struct city *current = head;
    while (current != NULL) {
        int xDiff = current->x - x;
        int yDiff = current->y - y;
        double dist = sqrt(xDiff*xDiff + yDiff*yDiff);
        if (dist <= distance) {
            printf("%s: %d, %d\n", current->name,
current->x, current->y);
        }
        current = current->next;
    }
}

int main() {
    struct city *cityList = (struct city *)
malloc(sizeof(struct city));
    cityList->next = NULL;

    // Insert records
    insert("Ahmedabad", 0, 0);
    insert("Baroda", 10, 20);
    insert("Rajkot", 20, 10);
    insert("Kutchh", 30, 10);
    insert("Surat", 40, 10);
    insert("Nadiad", 50, 10);
    insert("Anand", 60, 10);

    printf("Deleting City -\n");
    // Delete a record by name
    deleteByName("Baroda");
    printf("City Deleted by Name!\n");
    // Delete a record by coordinate
    deleteByCoordinate(50, 10);
    printf("City Deleted by Coordinate!\n");
    printf("\n");

    // Search a record by name
```

```
    printf("Seaching City by name -\n");
    struct city *result1 = searchByName("Ahmedabad");
    if (result1 != NULL) {
        printf("City found: %s\n", result1->name);
    } else {
        printf("City not found\n");
    }
    // Search a record by coordinate
    printf("Seaching City by Coordinate -\n");
    struct city *result2 = searchByCoordinate(20, 10);
    if (result2 != NULL) {
        printf("City found: %s\n", result2->name);
    } else {
        printf("City not found\n");
    }
    printf("\n");

    // Print all records within a given distance of a
specified point
    printf("printing City by given distance -\n");
    printByDistance(cityList, 15, 15, 200);

    return 0;
}
```

- **Output - 1 :-**

```
Deleting City -
City Deleted by Name!
City Deleted by Coordinate!

Seaching City by name -
City found: Ahmedabad
Seaching City by Coordinate -
City found: Rajkot

printing City by given distance -
Anand: 60, 10
Surat: 40, 10
Kutchh: 30, 10
Rajkot: 20, 10
Ahmedabad: 0, 0
```

- **Program - 2[Array Implementation] :-**

```
// Array Implementation -

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

struct City {
    char name[100]; // name of the city
    int x; // x coordinate
    int y; // y coordinate
};

struct City cityArray[100]; // array to store city
records
int numCities = 0; // number of cities currently in the
array

void insertRecord(struct City newCity) {
    if (numCities < 100) {
        cityArray[numCities++] = newCity;
    } else {
        printf("Error: City array is full\n");
    }
}

void deleteRecordByName(char *name) {
    int i, j;
    for (i = 0; i < numCities; i++) {
        if (strcmp(cityArray[i].name, name) == 0) {
            for (j = i; j < numCities-1; j++) {
                cityArray[j] = cityArray[j+1];
            }
            numCities--;
            return;
        }
    }
    printf("City not found\n");
}

struct City* searchRecordByName(char *name) {
    int i;
```

```
    for (i = 0; i < numCities; i++) {
        if (strcmp(cityArray[i].name, name) == 0) {
            return &cityArray[i];
        }
    }
    return NULL;
}

void printRecordsWithinDistance(int x, int y, int
distance) {
    int i;
    for (i = 0; i < numCities; i++) {
        int xDiff = abs(cityArray[i].x - x);
        int yDiff = abs(cityArray[i].y - y);
        if (sqrt(xDiff*xDiff + yDiff*yDiff) <= distance)
{
            printf("%s\n", cityArray[i].name);
        }
    }
}

int main() {
    // Insert records
    struct City newCity1 = {"Ahmedabad", 0, 0};
    insertRecord(newCity1);
    struct City newCity2 = {"Baroda", 10, 20};
    insertRecord(newCity2);
    struct City newCity3 = {"Rajkot", 20, 10};
    insertRecord(newCity3);
    struct City newCity4 = {"Kutchh", 30, 10};
    insertRecord(newCity4);
    struct City newCity5 = {"Surat", 40, 10};
    insertRecord(newCity5);

    printf("Deleting City -\n");
    // Delete a record by name
    deleteRecordByName("Kutchh");
    printf("City Deleted by Name!\n");
    printf("\n");

    // Search a record by name
    printf("Seaching City by name -\n");
    struct City *result = searchRecordByName("Surat");
    if (result != NULL) {
        printf("City found: %s\n", result->name);
```

```
    } else {
        printf("City not found\n");
    }
    printf("\n");

    // Print all records within a given distance of a
specified point
    printf("printing City by given distance -\n");
    printRecordsWithinDistance(15, 15, 200);

    return 0;
}
```

- **Output - 2 :-**

```
Deleting City -
City Deleted by Name!

Seaching City by name -
City found: Surat

printing City by given distance -
Ahmedabad
Baroda
Rajkot
Surat
```

- **Answer - (a) :-**

The running time for each operation in a city database implemented using unordered lists (such as a linked list) would be:

1. Insert a record: O(1) - The operation takes constant time, as it only requires allocating memory for a new node and updating the pointers of the previous and next nodes.
2. Delete a record by name: O(n) - The operation takes linear time, as it needs to traverse the list to find the node with the specified name. The time complexity will depend on the number of elements in the list.
3. Search a record by name: O(n) - The operation takes linear time, as it needs to traverse the list to find the node with the specified name. The time complexity will depend on the number of elements in the list.
4. Print all records within a given distance of a specified point: O(n) - The operation takes linear time, as it needs to traverse the list and calculate the distance from each node to the specified point. The time complexity will depend on the number of elements in the list.

The running time for each operation in a city database implemented using arrays would be:

1. Insert a record: O(1) - The operation takes constant time, as it only requires adding the new record to the next available slot in the array.
2. Delete a record by name: O(n) - The operation takes linear time, as it needs to traverse the array to find the record with the specified name and then shift the remaining records to fill the gap. The time complexity will depend on the number of elements in the array.
3. Search a record by name: O(n) - The operation takes linear time, as it needs to traverse the array to find the record with the specified name. The time complexity will depend on the number of elements in the array.
4. Print all records within a given distance of a specified point: O(n) - The operation takes linear time, as it needs to traverse the array and calculate the distance from each record to the

specified point. The time complexity will depend on the number of elements in the array.

It's worth noting that when the number of elements in the array becomes very large, the time complexity of the array implementation may become faster than the linked list implementation due to the cache locality of arrays.

- **Answer - (b) :-**

The main advantage of a city database implemented using unordered lists (such as a linked list) is that it can grow or shrink dynamically, which allows for efficient insertion and deletion operations. Since the linked list doesn't have a fixed size, it can continue to grow as new elements are added, and it doesn't require any resizing operations.

On the other hand, a city database implemented using arrays has the advantage of better cache locality, as all elements are stored in contiguous memory. This can result in faster traversals and access times when the number of elements in the array becomes very large.

- **Answer - (c) :-**

Storing records on the list in alphabetical order by city name could potentially speed up certain operations for a city database implemented using unordered lists (such as a linked list) or an array.
In the case of a linked list, storing the records in alphabetical order would make it easier to perform binary search on the list when searching for a record by name. Instead of having to traverse the entire list, you could use the binary search algorithm to narrow down the search area, which would significantly reduce the time complexity of the search operation.

In an array, if the records are stored in alphabetical order by city name, you could use the binary search algorithm to search for a specific record by name, the time complexity for this operation will be O(log n) instead of O(n) if you have to traverse the array.

- **Answer - (d) :-**

Keeping the list in alphabetical order could potentially slow down certain operations for a city database implemented using unordered lists (such as a linked list) or an array.
In the case of a linked list, inserting new records would take longer, since the records would need to be inserted into the correct position in the list to maintain the alphabetical order. This would require traversing the list to find the correct position for the new record, which could take linear time depending on the number of elements in the list.
In an array, similarly, inserting new records would take longer, since the records would need to be inserted into the correct position in the array to maintain the alphabetical order. This would require shifting existing elements to make room for the new record, which could take linear time depending on the number of elements in the array.