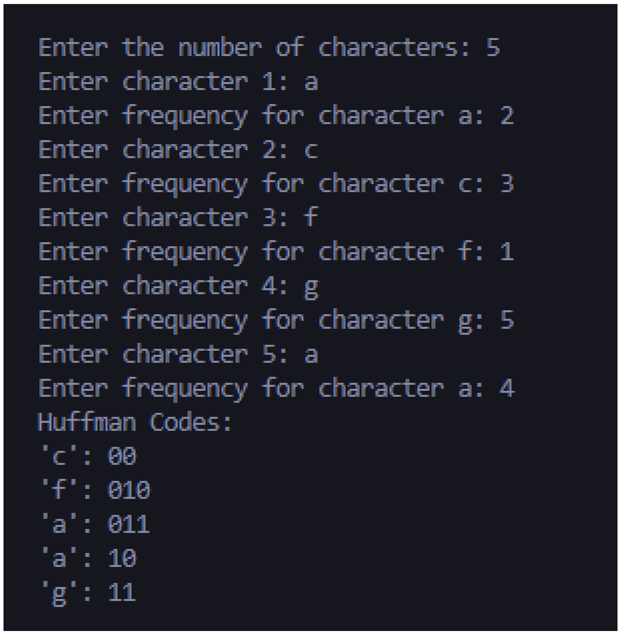
```
// Algorithm referenced from Programiz
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_TREE_NODES 1000
struct Node {
    char data;
    int frequency;
    struct Node* left;
    struct Node* right;};
struct MinHeap {
    int size;
    int capacity;
    struct Node** array;};
struct Node* createNode(char data, int frequency) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->frequency = frequency;
    newNode->left = newNode->right = NULL;
    return newNode;
}
struct MinHeap* createMinHeap(int capacity) {
    struct MinHeap* minHeap = (struct MinHeap*)malloc(sizeof(struct MinHeap));
    minHeap->size = 0;
    minHeap->capacity = capacity;
    minHeap->array = (struct Node**)malloc(capacity * sizeof(struct Node*));
    return minHeap;
}
void swapNodes(struct Node** a, struct Node** b) {
    struct Node* temp = *a;
    *a = *b;
    *b = temp;
}
void minHeapify(struct MinHeap* minHeap, int idx) {
    int smallest = idx;
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;
    if (left < minHeap->size && minHeap->array[left]->frequency <
                            minHeap->array[smallest]->frequency) smallest = left;
    if (right < minHeap->size && minHeap->array[right]->frequency <
                            minHeap->array[smallest]->frequency) smallest = right;
    if (smallest != idx) {
        swapNodes(&minHeap->array[smallest], &minHeap->array[idx]);
        minHeapify(minHeap, smallest);
    }
}
int isSizeOne(struct MinHeap* minHeap) {
    return minHeap->size == 1;
}
struct Node* extractMin(struct MinHeap* minHeap) {
    struct Node* temp = minHeap->array[0];

    minHeap->array[0] = minHeap->array[minHeap->size - 1];

    --minHeap->size;

    minHeapify(minHeap, 0);

    return temp;

}

void insertMinHeap(struct MinHeap* minHeap, struct Node* node) {

    ++minHeap->size;

    int i = minHeap->size - 1;

    while (i > 0 && node->frequency < minHeap->array[(i - 1) / 2]->frequency) {

        minHeap->array[i] = minHeap->array[(i - 1) / 2];

        i = (i - 1) / 2;

    }

    minHeap->array[i] = node;

}

struct MinHeap* buildMinHeap(char data[], int freq[], int size) {

    struct MinHeap* minHeap = createMinHeap(size);

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

        minHeap->array[i] = createNode(data[i], freq[i]);

    }

    minHeap->size = size;

    int n = minHeap->size - 1;

    for (int i = (n - 1) / 2; i >= 0; --i) {

        minHeapify(minHeap, i);

    }
    return minHeap;
}
struct Node* buildHuffmanTree(char data[], int freq[], int size) {
    struct Node *left, *right, *top;
    struct MinHeap* minHeap = buildMinHeap(data, freq, size);
    while (!isSizeOne(minHeap)) {
        left = extractMin(minHeap);
        right = extractMin(minHeap);
        top = createNode('$', left->frequency + right->frequency);
        top->left = left;
        top->right = right;
        insertMinHeap(minHeap, top);
    }
    return extractMin(minHeap);
}
void printCodes(struct Node* root, int arr[], int top) {
    if (root->left) {
        arr[top] = 0;
        printCodes(root->left, arr, top + 1);
    }
    if (root->right) {
        arr[top] = 1;
        printCodes(root->right, arr, top + 1);
    }
    if (!root->left && !root->right) {
        printf("'%c': ", root->data);
        for (int i = 0; i < top; ++i) {
            printf("%d", arr[i]);
        }
        printf("\n");
    }
}
void HuffmanCodes(char data[], int freq[], int size) {
    struct Node* root = buildHuffmanTree(data, freq, size);
    int arr[MAX_TREE_NODES], top = 0;
    printf("Huffman Codes:\n");
    printCodes(root, arr, top);
}
int main() {
    int n;
    printf("Enter the number of characters: ");
    scanf("%d", &n);
    char data[n];
    int freq[n];
    for (int i = 0; i < n; ++i) {
        printf("Enter character %d: ", i + 1);
        scanf(" %c", &data[i]);
        printf("Enter frequency for character %c: ", data[i]);
        scanf("%d", &freq[i]);
    }
    HuffmanCodes(data, freq, n);

    return 0;

}
```

OUTPUT :

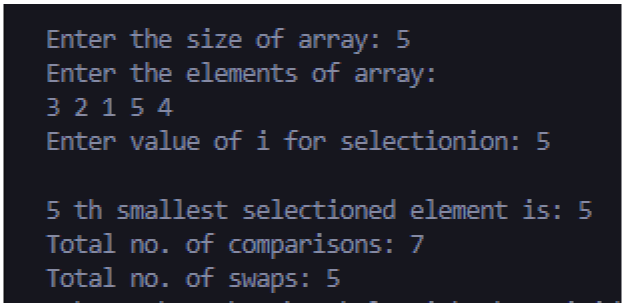


```
Enter the number of characters: 5
Enter character 1: a
Enter frequency for character a: 2
Enter character 2: c
Enter frequency for character c: 3
Enter character 3: f
Enter frequency for character f: 1
Enter character 4: g
Enter frequency for character g: 5
Enter character 5: a
Enter frequency for character a: 4
Huffman Codes:
'c': 00
'f': 010
'a': 011
'a': 10
'g': 11
```

```
#include <stdio.h>

#include <math.h>

int comparison = 0, swap = 0;

int partition(int a[50], int l, int r)

{

    int pivot = a[l];

    int i = l, j;

    for(j = i + 1; j <= r; j++)

    {

                            comparison++;

        if(a[j] <= pivot)

        {

            i++;

                                          swap++;

            int temp = a[i];

            a[i] = a[j];

            a[j] = temp;

        }

    }

              swap++;

    int temp = a[l];

    a[l] = a[i];

    a[i] = temp;

    return i;

}
```

```
int selection(int a[], int p, int r, int i)
{
        comparison++;
        if(p == r) return a[p];
        int q = partition(a, p, r);
        int k = q - p + 1;
        if(i == k) return a[q];
        if(i < k) return selection(a, p, q - 1, i);
        if(i > k) return selection(a, q + 1, r, i - k);
}

int main()
{
        int a[10], num, i, l, r, result;
        printf("Enter the size of array: ");
        scanf("%d", &num);

        printf("Enter the elements of array: \n");
        for(i = 0; i < num; i++)
        {
                scanf("%d", &a[i]);
        }
        l = 0;
        r = num - 1;
        printf("Enter value of i for selectionion: ");
        scanf("%d", &i);


        result = selection(a, l, r, i);

        printf("\n%d th smallest selectioned element is: %d", i, result);

        printf("\nTotal no. of comparisons: %d", comparison);

        printf("\nTotal no. of swaps: %d", swap);

}
```

OUTPUT :

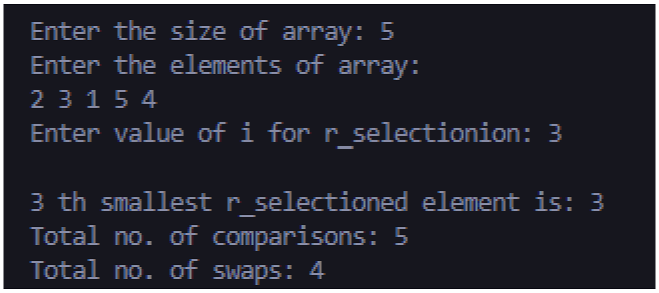


```
Enter the size of array: 5
Enter the elements of array:
3 2 1 5 4
Enter value of i for selectionion: 5

5 th smallest selectioned element is: 5
Total no. of comparisons: 7
Total no. of swaps: 5
```

```
#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <time.h>

int comparison = 0, swap = 0;


int partition(int a[50], int l, int r)

{

    int pivot = a[l];

    int i = l, j;

    for(j = i + 1; j <= r; j++)

    {

                            comparison++;

        if(a[j] <= pivot)

        {

            i++;

                                          swap++;

            int temp = a[i];

            a[i] = a[j];

            a[j] = temp;

        }

    }

              swap++;

    int temp = a[l];

    a[l] = a[i];
    a[i] = temp;
    return i;
}
int randomized_partition(int a[50], int p, int r)
{
            // Generate random number within range of 'p' and 'r'
    int k = (rand() % (r - p + 1)) + p;
            swap++;
    int temp = a[p];
    a[p] = a[k];
    a[k] = temp;
    return partition(a, p, r);
}
int r_selection(int a[], int p, int r, int i)
{
            comparison++;
            if(p == r) return a[p];
            int q = randomized_partition(a, p, r);
            int k = q - p + 1;
            if(i == k) return a[q];
            if(i < k) return r_selection(a, p, q - 1, i);
            if(i > k) return r_selection(a, q + 1, r, i - k);
}

int main()
```

```
{
        int a[10], num, i, l, r, result;
        printf("Enter the size of array: ");
        scanf("%d", &num);

        printf("Enter the elements of array: \n");
        for(i = 0; i < num; i++)
        {
                scanf("%d", &a[i]);
        }
        l = 0;
        r = num - 1;
        printf("Enter value of i for r_selectionion: ");
        scanf("%d", &i);

        result = r_selection(a, l, r, i);
        printf("\n%d th smallest r_selectioned element is: %d", i, result);
        printf("\nTotal no. of comparisons: %d", comparison);
        printf("\nTotal no. of swaps: %d", swap);
}
```

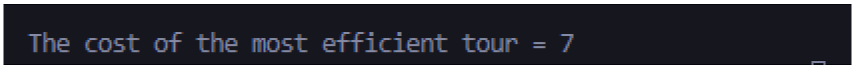
OUTPUT :



```
Enter the size of array: 5
Enter the elements of array:
2 3 1 5 4
Enter value of i for r_selectionion: 3

3 th smallest r_selectioned element is: 3
Total no. of comparisons: 5
Total no. of swaps: 4
```

```
#include <stdio.h>

#define n 4                                         // No. of nodes

#define MAX 1000000

int dist[n + 1][n + 1] = {

    { 0, 0, 0, 0, 0 },

              { 0, 0, 6, 1, 3 },

    { 0, 4, 0, 2, 1 },

              { 0, 1, 2, 0, 8 },

    { 0, 3, 1, 7, 0 },

};

// Memoization for top-down recursion

int memo[n + 1][1 << (n + 1)];

int min(int a, int b) {

    return a < b ? a : b;

}

int fun(int i, int mask) {

    // Base case

    if (mask == ((1 << i) | 3)) return dist[1][i];

    // Memoization

    if (memo[i][mask] != 0) return memo[i][mask];

    int res = MAX; // Result of this sub-problem

    for (int j = 1; j <= n; j++)

        if ((mask & (1 << j)) && j != i && j != 1)

            res = min(res, fun(j, mask & (~(1 << i))) + dist[j][i]);

    return memo[i][mask] = res;
}

int main() {

    int ans = MAX;

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

        ans = min(ans, fun(i, (1 << (n + 1)) - 1) + dist[i][1]);

    printf("The cost of the most efficient tour = %d", ans);

    return 0;

}
```

OUTPUT :

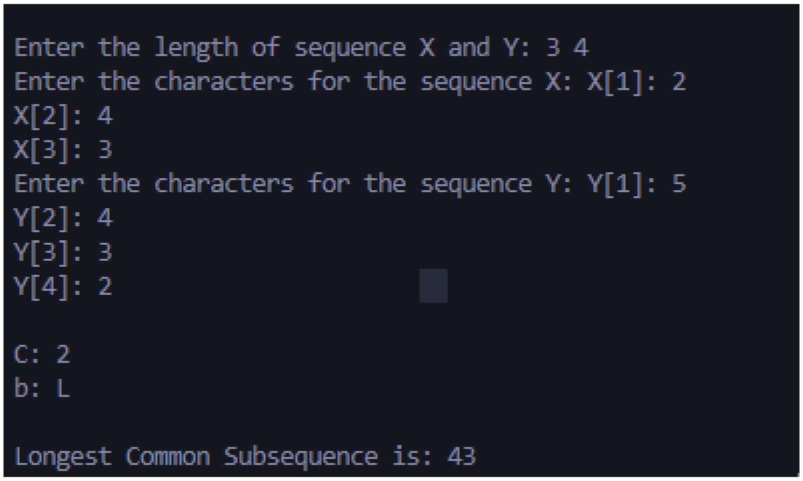


```
The cost of the most efficient tour = 7
```

```
#include<stdio.h>


void print_LCS(int m, char b[][m], char X[], int i, int j)

{

  if(i == 0 || j == 0)

      return;

  if(b[i][j] == 'C'){

      print_LCS(m, b, X, i - 1, j - 1);

      printf("%c", X[i - 1]);

  }

  else if(b[i][j] == 'U')

      print_LCS(m, b, X, i - 1, j);

  else

      print_LCS(m, b, X, i, j - 1);

}


int main()

{

   int m, n;

   printf("Enter the length of sequence X and Y: ");

   scanf("%d %d", &m, &n);

   char X[m], Y[n];

   printf("Enter the characters for the sequence X: ");

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

      printf("X[%d]: ", i + 1);
```

```
    scanf("%s", &X[i]);
}
printf("Enter the characters for the sequence Y: ");
for(int i = 0; i < n; i++){
    printf("Y[%d]: ", i + 1);
    scanf("%s", &Y[i]);
}
// Calculation for LCS_length
char b[m + 1][n + 1];
int c[m + 1][n + 1];
for(int i = 0; i <= m; i++){
    c[i][0] = 0;
}
for(int j = 0; j <= n; j++){
    c[0][j] = 0;
}
for(int i = 1; i <= m; i++){
    for(int j = 1; j <= n; j++){
        if(X[i - 1] == Y[j - 1]){
            c[i][j] = c[i - 1][j - 1] + 1;
            b[i][j] = 'C';
        }
        else if(c[i - 1][j] >= c[i][j - 1]){
            c[i][j] = c[i - 1][j];
            b[i][j] = 'U';
            }
            else{
                c[i][j] = c[i][j - 1];
                b[i][j] = 'L';
            }
        }
    }
    //print b and c table
    printf("\nC: %d\n", c[m][n]);
    printf("b: %c\n", b[m][n]);
              printf("\nLongest Common Subsequence is: ");
    print_LCS(n + 1, b, X, m, n);
    return 0;
}
```

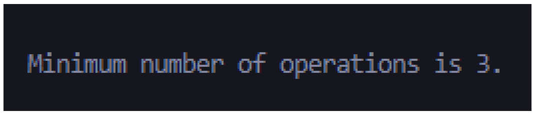
OUTPUT :



```
Enter the length of sequence X and Y: 3 4
Enter the characters for the sequence X: X[1]: 2
X[2]: 4
X[3]: 3
Enter the characters for the sequence Y: Y[1]: 5
Y[2]: 4
Y[3]: 3
Y[4]: 2

C: 2
b: L

Longest Common Subsequence is: 43
```

```c
// Implementing string editing algorithm(leveshtein algorithm) in C
#include <string.h>
#include <stdio.h>

static int distance (char * word1, int len1,char * word2,int len2)
{
    int i, j, matrix[len1 + 1][len2 + 1];
    for (i = 0; i <= len1; i++) {
        matrix[i][0] = i;
    }
    for (i = 0; i <= len2; i++) {
        matrix[0][i] = i;
    }
    for (i = 1; i <= len1; i++) {
        char c1, c2;
        c1 = word1[i-1];
        for (j = 1; j <= len2; j++) {
            c2 = word2[j-1];
            if (c1 == c2) {
                matrix[i][j] = matrix[i-1][j-1];
            }
            else {
                int delete = matrix[i-1][j] + 1;
                int insert = matrix[i][j-1] + 1;
                int replace = matrix[i-1][j-1] + 1;
        int minimum = delete;

        if (insert < minimum) minimum = insert;

        if (replace < minimum) minimum = replace;

        matrix[i][j] = minimum;

      }

    }

  }

  return matrix[len1][len2];

}


int main ()

{

  char * word1, * word2;

  word1 = "suraj";

  word2 = "kumal";

  int len1 = strlen (word1);

  int len2 = strlen (word2);

  int d = distance (word1, len1, word2, len2);

  printf ("\nMinimum number of operations is %d.\n",d);

  return 0;

}
```

OUTPUT :



```
Minimum number of operations is 3.
```

NAGARJUNA COLLEGE OF INFORMATION TECHNOLOGY

DESIGN AND ANALYSIS OF ALGORITHM

PREPARED BY:

SURAJ KUMAL

ROLL NO: 32

SUBMITTED TO:

BHIM RAWAT