Question 1. Order statistics: Write codes for Rand-Select (with linear expected running time) and Select (with linear worst-case running time). Test your two programs with an input array that is a random permutation of A = {1, 2, 3, …, 99, 100}.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <algorithm>
#include <iostream>

using namespace std;

void Insertion_Sort(int A[], int initial, int end)  //Cite from
assignment 1.
{
    for(int i = initial + 1; i <= end; i++)
    {
        if(A[i - 1] > A[i])
        {
            int tmp = A[i];
            int r = i;
            while(r > initial && A[r - 1] > tmp)
            {
                A[r] = A[r - 1];
                r = r - 1;
            }
            A[r] = tmp;
        }
    }
}

int trackMid(int A[], int initial, int end)
{
    int i = 0,num1 = 0;
    if(initial == end) return A[initial];
    for(i = initial; i < end - 5; i = i + 5)
    {
        Insertion_Sort(A, i, i + 4);
        num1 = i - initial;
        swap(A[initial + num1 / 5], A[i + 2]);
    }
    int num2 = end - i + 1;
```

```
    if(num2 > 0)
    {
        Insertion_Sort(A, i, i + num2 - 1);
        num1 = i - initial;
        swap(A[initial + num2 / 5], A[i + num2 / 2]);
    }
    num1 = num1 / 5;
    if(num1 == initial) return A[initial];
    return trackMid(A, initial, initial + num1);
}

int trackMid_Index(int A[], int initial, int end, int num)
{
    for(int i = initial; i <= end; i++)
        if(A[i] == num)
            return i;
    return -1;
}

int sort_partition(int A[], int initial, int end, int mid_index)
{
    swap(A[mid_index], A[initial]);
    int i = initial;
    int j = end;
    int pivot = A[initial];
    while(i < j)
    {
        while(A[j] >= pivot && i < j)
            j = j - 1;
        A[i] = A[j];
        while(A[i] <= pivot && i < j)
            i = i + 1;
        A[j] = A[i];
    }
    A[i] = pivot;
    return i;
}

int rand_select_worst(int A[], int initial, int end, int k)
{
    int num = trackMid(A, initial, end);
    int mid_index =  trackMid_Index(A, initial, end, num);
    int i = sort_partition(A, initial, end, mid_index);
    int new_pivot = i - initial + 1;
```

```
    if(new_pivot == k) return A[i];
    if(new_pivot > k)  return rand_select_worst(A, initial, i - 1, k);
    return rand_select_worst(A, i + 1, end, k - new_pivot);
}

int get_pivot_expected(int A[], int initial, int end)
{
    int pivot = A[initial];
    while (initial < end) {
        while (initial < end && A[end] >= pivot) {
            --end;
        }
        A[initial] = A[end];
        while (initial < end && A[initial] <= pivot) {
            ++initial;
        }
        A[end] = A[initial];
    }
    A[initial] = pivot;
    return initial;
}

void rand_select_expected(int A[], int initial, int end, int k){
    int index;
    if(end < initial)return;
    index = get_pivot_expected(A, initial, end);
    if(index == k - 1){
        printf("Use the rand_select_expected, the 50th element
is %d.\n",A[index]);
        return;
    }
    if(index > k - 1){
        rand_select_expected(A, initial, index - 1, k);
    }
    if(index < k - 1){
        rand_select_expected(A, index + 1, end, k);
    }
}

int main()
{
    int A1[100], A2[100];
    int A[100] = {0};
    int A_index = 0;
```

```
    srand(time(NULL));
    for(int i = 1; i <= 100; i++)
    {
        do{
            A_index = rand() % 100; //generate a random index 0-99 and
insert i if A[A_index] is empty.
        }
        while (A[A_index] != 0);
        A[A_index] = i;
    }
    for(int i = 0; i < 100; i++)
    {
        A1[i] = A[i];
        A2[i] = A[i];
    }
    rand_select_expected(A1,0,99,50);
    printf("Use the rand_select_worst, the 30th element
is %d.\n",rand_select_worst(A2,0,99,30));
    return 0;
}
```

Results:

```
PS D:\Code> cd "d:\Code\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Use the rand_select_expected, the 50th element is 50.
Use the rand_select_worst, the 30th element is 30.
```

Question 2. Dynamic Programming of LCS: Write codes for the longest common subsequence.

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

char string1[20], string2[20];
char record[21][21] = {0}; //Records array.
char flag[21][21] = {0};    ///Flags array to track the same characters.

void LCS(int n, int m)
{
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= m; j++)
        {
            if(string1[i - 1] == string2[j - 1])
```

```
            {
                record[i][j] = record[i - 1][j - 1] + 1;
                flag[i][j] = 1;  //if flag == 1 then catch the same
element.
            }
            else if(record[i][j - 1] > record[i - 1][j])  //compare the
right element and the above element.
            {
                record[i][j] = record[i][j - 1];
                flag[i][j] = -1;
            }
            else
            {
                record[i][j] = record[i - 1][j];
                flag[i][j] = 2;
            }
        }
    }
}

void track_LCS(int n, int m)
{
    char result[20];
    int k = 0;    //To preserve the result.
    while(n > 0 && m > 0)
    {
        if(flag[n][m] == 1)
        {
            result[k]=string1[n - 1];
            k++;
            n--;
            m--;
        }
        else if(flag[n][m] == -1)
            m--;
        else if(flag[n][m] == 2)
            n--;
    }
    for(n = k - 1; n >= 0; n--)
        printf("%c",result[n]);
}

int main()
{
```

```
    strcpy(string1,"northeastern");
    strcpy(string2,"northwestern");
    LCS(strlen(string1),strlen(string2));
    printf("%d\n",record[strlen(string1)][strlen(string2)]);
    track_LCS(strlen(string1),strlen(string2));
    return 0;
}
```

Results:

```
PS D:\Code> cd "d:\Code\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
11
northestern
```