Homework 3 (5pt.)
Submission instruction:
Submit one single pdf file for this homework including both coding problems and analysis problems.
For coding problems, copy and paste your codes. Report your results.
For analysis problems, either type or hand-write and scan.

Question 1 (3 pt.) 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} (reuse of your Homework 2).

Code attached at the end.
Rand-Select with linear expected running time results:

```
bash-4.2$ g++ -std=c++11 expected rand-select.cpp -o rand && ./rand
Original Array: 55 93 31 45 84 18 39 89 90 10 66 85 14 28 41 53 29 87 21 15 42 47 100 9 6 98 96 50 49 80 36 70 64 95 57 83 48 69 11 2 65 61 76 91 86 68 78 16 20 5
6 52 38 94 44 59 26 32 97 71 82 99 34 73 25 12 7 54 74 79 30 22 35 27 51 62 33 37 67 4 88 24 75 46 8 72 5 23 19 81 63 58 43 60 17 13 1 92 40 3 77
Smallest Element: 1
```

Rand-Select with linear wort-case running time results:

```
bash-4.2$ g++ -std=c++11 worst-case rand-select.cpp -o rand && ./rand
Original Array: 61 74 59 75 90 5 76 38 50 7 86 30 35 94 39 63 40 62 79 46 81 77 92 34 71 26 17 33 21 80 28 6 9 54 93 87 1 42 47 88 55 58 68 72 25 4 96 57 49 15 45
 60 32 48 98 84 27 22 70 85 97 2 56 13 69 37 41 52 18 31 8 20 14 73 43 29 66 78 95 16 53 65 67 100 23 51 82 99 83 91 12 11 19 89 3 10 44 64 36 24
Smallest Element: 1
```

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

Code attached at the end.
LCS results:

```
bash-4.2$ g++ -std=c++11 lcs.cpp -o lcs && ./lcs

X: A B C B D A B
Y: B D C A B A

C Table:
0 0 0 0 0 0 0
0 0 0 0 1 1 1
0 1 1 1 1 2 2
0 1 1 2 2 2 2
0 1 1 2 2 3 3
0 1 2 2 2 3 3
0 1 2 2 3 3 4
0 1 2 2 3 4 4


B Table:
2 2 2 1 3 1
1 3 3 2 1 3
2 2 1 3 2 2
1 2 2 2 1 3
2 1 2 2 2 2
2 2 2 1 2 1
1 2 2 2 1 2

Possible LCS: B C B A

bash-4.2$
```

```cpp
// Rand-Select with linear expected running time //
// 5 November 2019                               //
// Author: Anna DeVries                          //

#include <iostream>
#include <stdlib.h>

// Rearranges the subarray A[p...r] in place
int partition(int A[], int p, int r){
    // Initalize variables
    int x = A[r];       // pivot point
    int i = p - 1;
    int j,hold;

    // Rearranges
    for(j = p; j < r; j++){
        if(A[j] <= x){
            i = i + 1;
            std::swap(A[i], A[j]);
        }
    }

    std::swap(A[i + 1],A[r]);
    return i + 1;
}

// Randomizes pivot point
int randomized_partition(int A[], int p, int r){
    // Initalize variables
    srand(time(NULL));
    int i = (rand() % (r - p)) + p;

    std::swap(A[r],A[i]);
    return partition(A, p, r);
}

// Randomized Selection in expected linear time
int rand_select(int A[], int p, int r, int i){
    // Initialize variables
    int q, k;
    if(p == r){
        return A[p];
    }
    q = randomized_partition(A, p, r);

    k = q - p + 1;
    if(i == k){
        return A[q];
    }
    else if(i < k){
        return rand_select(A, p, q - 1, i);
    }
    else{
        return rand_select(A, q + 1, r, i - k);
    }
}

// Randomizes order of array
int randomize(int A[], int n){
    srand(time(NULL));
    for(int i = n - 1; i > 0; i--){
        int j = rand() % (i + 1);
        int hold = A[i];
        A[i] = A[j];
        A[j] = hold;
    }
}

int main(){
```

```
70      // Initialize array A[1,2,...,99,100] as a random permutation
71      int A[100];
72      int n, p, r, j, i;
73
74      for(int i = 0; i < 100; i++){
75          A[i] = i+1;
76      }
77      n = sizeof(A)/sizeof(A[0]);
78      i = 1;
79      p = 0;
80      r = n - 1;
81      randomize(A, n);
82
83      // Print original array
84      std::cout << "Original Array: ";
85      for(j = 0; j < n; j++){
86          std::cout << A[j] << " ";
87      }
88      std::cout << std::endl;
89
90      // Randomized selection algorithm
91      std::cout << "Smallest Element: " << rand_select(A, p, r, i) << std::endl;
92  }
```

```cpp
// Rand-Select with linear worst-case running time //
// 5 November 2019                                 //
// Author: Anna DeVries                            //

#include <iostream>
#include <stdlib.h>
#include <algorithm>

// Rearranges the subarray A[p...r] in place
int partition(int A[], int p, int r, int x){
    // Initalize variables
    int i = p - 1;
    int j, hold;

    // Rearranges
    for(j = p; j < r; j++){
        if(A[j] <= x){
            i = i + 1;
            std::swap(A[i], A[j]);
        }
    }

    std::swap(A[i + 1],A[r]);
    return i + 1;
}

// Returns median value
int median(int A[],int n){
    std::sort(A, A + n);
    return A[n / 2];
}

// Randomized Selection in worst-case linear time
int rand_select(int A[], int p, int r, int i){
    // Initalize varaibles
    int j, k, q, x, y;
    int B[(r - p + 5) / 5];

    // Divide the n elements into n/5 groups of 5 elements each
    for(j = 0; j < ((r - p + 1) / 5); j++){
        B[j] = median(A + p + j * 5, 5);
    }

    // Recursively select the median x of the n/5 medians
    if(j * 5 < (r - p + 1)){
        B[j] = median(A + p + (j * 5), (r - p + 1) % 5);
        j++;
    }
    if(j == 1){
        x = B[j - 1];
    }
    else{
        x = rand_select(B, 0, j - 1, j / 2);
    }

    // Partition with x as pivot
    q = partition(A, p, r, x);

    k = q - p + 1;
    if(i == k){
        return A[q];
    }
    else if(i < k){
        return rand_select(A, p, q - 1, i);
    }
    else{
        return rand_select(A, q + 1, r, i - k);
    }
}
```

```
70
71      // Randomizes order of array
72  int randomize(int A[], int n){
73      srand(time(NULL));
74      for(int i = n - 1; i > 0; i--){
75          int j = rand() % (i + 1);
76          int hold = A[i];
77          A[i] = A[j];
78          A[j] = hold;
79      }
80  }
81
82  int main(){
83      // Initialize array A[1,2,...,99,100] as a random permutation
84      int n, p, r, j, i;
85      int A[100];
86
87      for(int i = 0; i < 100; i++){
88          A[i] = i+1;
89      }
90      n = sizeof(A)/sizeof(A[0]);
91      i = 1;
92      p = 0;
93      r = n - 1;
94      randomize(A, n);
95
96      // Print original array
97      std::cout << "Original Array: ";
98      for(j = 0; j < n; j++){
99          std::cout << A[j] << " ";
100     }
101     std::cout << std::endl;
102
103     // Randomized selection algorithm
104     std::cout << "Smallest Element: " << rand_select(A, p, r, i) << std::endl;
105 }
```

```cpp
// Dynamic programming of LCS //
// 5 November 2019            //
// Author: Anna DeVries       //

#include <iostream>
#include <stdlib.h>

// Initialize b table as global variable, such as b[m][n].
int b[7][6];

// Allows b table to store integers but users to call them with words.
enum DIRECTIONS{
    DIAGONAL=1,
    UP=2,
    LEFT=3
};

// Recursively prints out an LCS of X and Y in the proper, forward order.
void print_lcs(int X[], int i, int j){
    // Initalize variables
    int n, m;

    // If tables are empty, return
    if(i == 0 || j == 0){
        // Check last element for an arrow reference to c[0][j] || c[i][0]. If arrow is "up-left", print last X value.
        if(b[i][j] == DIAGONAL){
            std::cout << (char)X[i] << " ";
        }
        return;
    }

    // If last value in b table is "up-left", remove last row and column from table.
    // This value produces a LCS value
    if(b[i][j] == DIAGONAL){
        print_lcs(X, i - 1, j-1);
        std::cout << (char)X[i] << " ";
    }
    // If last value in b table is "up", remove last row from table.
    else if(b[i][j] == UP){
        print_lcs(X, i - 1, j);
    }
    // If last value in b table is "left", remove last column from table.
    else{
        print_lcs(X, i, j - 1);
    }
}

// Constructs b and c tables for solving LCS.
void lcs_length(int X[], int Y[], int m, int n){
    // Initialize variables
    int i, j, k;

    // Print original arrays.
    std::cout << std::endl;
    std::cout << "X: ";
    for(j = 0; j < m; j++){
        std::cout << (char)X[j] << " ";
    }
    std::cout << std::endl;

    std::cout << "Y: ";
    for(j = 0; j < n; j++){
        std::cout << (char)Y[j] << " ";
    }
    std::cout << std::endl;

    // Initialize tables.
    int c[m + 1][n + 1];
```

```
    for(i = 0; i < m; i++){
        for(j = 0; j < n; j++){
            b[i][j] = 0;
        }
    }
    for(i = 0; i <= m; i++){
        for(j = 0; j <= n; j++){
            c[i][j] = 0;
        }
    }

    // Fill values of table c and b in row-major order.
    for(i = 1; i <= m; i++){
        for(j = 1; j <= n; j++){
            if(X[i - 1] == Y[j - 1]){
                c[i][j] = c[i - 1][j - 1] + 1;
                b[i - 1][j - 1] = DIAGONAL;
            }
            else if(c[i - 1][j] >= c[i][j-1]){
                c[i][j] = c[i - 1][j];
                b[i - 1][j - 1] = UP;
            }
            else{
                c[i][j] = c[i][j-1];
                b[i - 1][j - 1] = LEFT;
            }
        }
    }

    // Print c table
    std::cout << std::endl;
    std::cout << "C Table:" << std::endl;
    for(i = 0; i <= m; i ++){
        for(j = 0; j <= n; j++){
            std::cout << c[i][j] << " ";
        }
        std::cout << std::endl;
    }
    std::cout << std::endl;

    // Print b table
    std::cout << std::endl;
    std::cout << "B Table:" << std::endl;
    for(i = 0; i < m; i ++){
        for(j = 0; j < n; j++){
            std::cout << b[i][j] << " ";
        }
        std::cout << std::endl;
    }
    std::cout << std::endl;

    // Find LCS
    std::cout << "Possible LCS: ";
    print_lcs(X, m-1, n-1);
    std::cout << std::endl;
    std::cout << std::endl;

    return;
}

int main(){
    // Initialize array A[] and B[]
    int X[] = {'A','B','C','B','D','A','B'};
    int Y[] = {'B','D','C','A','B','A'};
    int m = sizeof(X)/sizeof(X[0]);
    int n = sizeof(Y)/sizeof(Y[0]);

    lcs_length(X, Y, m, n);

```

```
138         return 0;
139     }
```