Answer 1

Select algorithm (linear worst-case running time)

```
#include <algorithm>
#include <iostream>
#include <random>
#include <vector>
#include <list>
#include <set>
#include <chrono>
#include <numeric>
using namespace std;
using namespace std::chrono;
void swap(int *p, int *q)
{
  // helper function
  int x = *p;
  *p = *q;
  *q = x;
}
int deterministic_partition(int A[], int l, int r, int k){
  int z;
  for (z=l;z<r;z++){
    if (A[z]==k){
      break;
    }
  }
  swap(A[z],A[l]);
  int p=A[l];
  int i=l;
  for (int j=l+1;j<r+1;j++){
    if (A[j]<=p){
      i=i+1;
      swap(A[i],A[j]);
    }
  }
  swap(A[i],A[l]);
  // return the pivot position
  return i;
}
int search(int A[], int j)
{
  sort(A, A+j);
  if (j%2==1){
    return A[j/2];
  }
  else{
    return (A[j/2]+ A[(j/2-1)])/2;
  }
```

```
}


int select(int A[], int l, int r, int order_statistic)
{
  int number_of_elements=r - l + 1;
  if (order_statistic > 0 && order_statistic <= number_of_elements)
  {
    int num_groups=(number_of_elements+4)/5;
    int arr[num_groups];
    int tmp;
    for (tmp=0; tmp<number_of_elements/5; tmp++){
      arr[tmp] = search(A+l+tmp*5, 5);
    }
    if (tmp*5 < number_of_elements){
      arr[tmp] = search(A+l+tmp*5, number_of_elements%5);
      tmp=tmp+1;
    }
    int centre;
    if (tmp==1){
      centre=arr[tmp-1];
    }
    else{
      centre=select(arr, 0, tmp-1, tmp/2);
    }
    int val = deterministic_partition(A, l, r, centre);
    if (val-l > order_statistic-1){
      return select(A, l, val-1, order_statistic);
    }
    else if (val-l < order_statistic-1){
      return select(A, 1+val, r, l-val-1+order_statistic);
    }
    else {
      return A[val];
    }
  }

  else{
    cout <<"Index larger than size of array"<<endl;
    return 0;
  }
}

int main()
{
```

```
  // Create a array with 1-100 numbers
  int A[101];
  for (int i=0; i<101; ++i){
    A[i]=i+1;
  }
  cout <<"Original permutation is ";
  for (int i=0; i<101; ++i){
    cout<<A[i]<<" ";
  }
  cout<<endl;
  random_device rd;
  mt19937 g(rd());
  // Shuffle the array with 1-100 numbers
  shuffle(A, A+100, g);
  cout <<"Shuffled permutation is ";
  for (int i=0; i<101; ++i){
    cout<<A[i]<<" ";
  }
  cout<<endl;
  int number_of_elements = 100;
  // test for different cases
  int order_statistic = 3;
  cout << order_statistic<< "rd order statistic is ";
  cout<<select(A, 0, number_of_elements-1, order_statistic)<<endl;
  order_statistic = 20;
  cout << order_statistic<< "th order statistic is ";
  cout<<select(A, 0, number_of_elements-1, order_statistic)<<endl;
  order_statistic = 42;
  cout << order_statistic<< "nd order statistic is ";
  cout<<select(A, 0, number_of_elements-1, order_statistic)<<endl;
  order_statistic = 98;
  cout << order_statistic<< "th order statistic is ";
  cout<<select(A, 0, number_of_elements-1, order_statistic)<<endl;
  order_statistic = 80;
  cout << order_statistic<< "th order statistic is ";
  cout<<select(A, 0, number_of_elements-1, order_statistic)<<endl;
  return 0;
}
```

Output-

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

Shuffled permutation is 3 48 16 56 79 2 73 100 87 23 5 4 90 67 78 32 52 63 65 10
 97 8 55 70 80 13 15 72 76 19 94 82 11 58 39 17 21 14 68 71 89 40 92 62 57 1 46
 37 53 28 99 30 41 84 42 81 12 45 60 83 96 61 22 47 24 36 20 7 91 26 95 74 35 31
 44 66 9 77 75 34 38 27 50 33 51 59 85 6 18 93 86 25 88 69 29 49 98 54 43 64

3rd order statistic is 3
20th order statistic is 20
42nd order statistic is 42
98th order statistic is 98
80th order statistic is 80
```

Rand-Select (with linear expected running time)

```
#include <algorithm>
#include <iostream>
#include <random>
#include <vector>
#include <list>
#include <set>
#include <chrono>
#include <numeric>
using namespace std;
using namespace std::chrono;
void swap(int *p, int *q)
{
  // helper function
  int x = *p;
  *p = *q;
  *q = x;
}
int randomized_partition(int A[], int l, int r){
  // find random pivot
  srand(time(NULL));
  int random_ind = l + rand() % (r - l);
  // swap it with first  element
  swap(A[random_ind],A[l]);
  int p=A[l];
  int i=l;
  for (int j=l+1;j<r+1;j++){
    if (A[j]<=p){
      i=i+1;
      swap(A[i],A[j]);
    }
  }
  swap(A[i],A[l]);
  // return the pivot position
  return i;
}

int rand_select(int A[], int l, int r, int i){
  if (l==r){
    return A[l];  // exist condition
  }
  int q=randomized_partition(A,l,r);
  int k=q-l+1;
  if (k==i){
    return A[q]; // exist condition
  } // recurse on one side
  else if (i<k){
```

```
    return rand_select(A,l,q-1,i);
  }else{
    return rand_select(A,q+1,r,i-k);
  }
}

int main()
{
  // Create a array with 1-100 numbers
  int A[101];
  for (int i=0; i<100; ++i){
    A[i]=i+1;
  }
  cout <<"Original permutation is ";
  for (int i=0; i<100; ++i){
    cout<<A[i]<<" ";
  }
  cout<<endl;
  random_device rd;
  mt19937 g(rd());
  // Shuffle the array with 1-100 numbers
  shuffle(A, A+100, g);
  cout <<"Shuffled permutation is ";
  for (int i=0; i<100; ++i){
    cout<<A[i]<<" ";
  }
  cout<<endl;
  int number_of_elements = 100;
  // test for different cases
  int order_statistic = 3;
  cout << order_statistic<< "rd order statistic is ";
  cout<<rand_select(A, 0, number_of_elements-1, order_statistic)<<endl;
  order_statistic = 20;
  cout << order_statistic<< "th order statistic is ";
  cout<<rand_select(A, 0, number_of_elements-1, order_statistic)<<endl;
  order_statistic = 42;
  cout << order_statistic<< "nd order statistic is ";
  cout<<rand_select(A, 0, number_of_elements-1, order_statistic)<<endl;
  order_statistic = 98;
  cout << order_statistic<< "th order statistic is ";
  cout<<rand_select(A, 0, number_of_elements-1, order_statistic)<<endl;
  order_statistic = 80;
  cout << order_statistic<< "th order statistic is ";
  cout<<rand_select(A, 0, number_of_elements-1, order_statistic)<<endl;
  return 0;
}
```

Output-

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

Shuffled permutation is 26 76 50 8 4 51 55 9 63 86 90 87 60 53 19 68 23 20 18 29
88 80 74 89 59 49 2 32 52 61 56 42 70 11 84 28 77 83 58 21 40 95 98 47 1 81 57
96 85 65 67 14 91 62 30 44 24 66 7 6 41 73 12 45 22 48 97 16 100 17 72 71 54 10
75 15 25 92 13 78 99 46 37 35 82 3 33 38 94 34 27 69 93 36 31 39 5 79 43 64

3rd order statistic is 3
20th order statistic is 20
42nd order statistic is 42
98th order statistic is 98
80th order statistic is 80
```

Answer 2

```
#include <algorithm>
#include <iostream>
#include <random>
#include <vector>
#include <list>
#include <set>
#include <chrono>
#include <numeric>
using namespace std;
using namespace std::chrono;

void print_2D_vector(vector<vector<int>> const &v) {
  //helper function to print 2D vector
for (vector<int> row: v) {
    for (int val: row) {
      cout << val << " ";
}
    cout << '\n';
  }
}

vector<vector<int>> LCSHelper(string A,string B){
  int n=A.size();
  int m=B.size();
  vector<vector<int>> memo(n+1, vector<int>(m+1, 0));
  for (int i=0;i<n+1;i++){
    for (int j=0;j<m+1;j++){
      if ((i==0) or (j==0)){
        memo[i][j]=0;
      }
      else if (A[i-1]==B[j-1]){
        memo[i][j]=memo[i-1][j-1]+1;
      }
      else{
        memo[i][j]=max(memo[i-1][j],memo[i][j-1]);
      }
    }
  }
  return memo;
}
string reconstruct(string A,string B,vector<vector<int>> memo){
  // reconstruct the LCS string
  int n=A.size();
  int m=B.size();
  int row=n;
  int col=m;
```

```
  string LCS="";
  while(row>0 and col>0){
    if (A[row-1] == B[col-1]){
      LCS=A[row-1]+LCS;
      row=row-1;
      col=col-1;
    }else if (memo[row-1][col] > memo[row][col-1]){
      row=row-1;
    }else{
      col=col-1;
    }
  }
  return LCS;
}
void longestCommonSubsequence(string A,string B){
  vector<vector<int>> memo=LCSHelper(A,B);
  // memo stores the entries of length of LCS for different string sizes
  // Required Answer is the last element of this matrix
  cout << "Length of LCS is : "<<memo[A.size()][B.size()]<<endl;
  // Compute what is the LCS. reconstruct it
  string common_subseq=reconstruct(A,B,memo);
  // Prints the LCS
  cout << "LCS of C and D is  : "<<common_subseq<<endl;
}

int main(int argc, char const *argv[]) {
  // Test input
  string C = "ABCDGH";
  string D = "AEDFHR";
  cout << "String C is : "<<C<<endl;
  cout << "String D is : "<<D<<endl;
  // Call the longestCommonSubsequence function. This function prints the length
  // of LCS and also the LCS string
  longestCommonSubsequence(C,D);
  return 0;
}
```

Output

```
String C is : ABCDGH
String D is : AEDFHR
Length of LCS is : 3
LCS of C and D is : ADH
```