CS 575
Homework 3
Total: 100 points

The written parts of assignment should be handed in person at the beginning of class on the submission date.

Written Part (50 points)

1. Sort the array of numbers 10, 7, 3, 8, 1, 9, 0 in ascending order using the **insertion sort algorithm**. Show the state of the array after each iteration of the algorithm. **(5 points)**

2. Use diagrams similar to the one shown in the slides and in your textbook to show the various stages of the **merge sort algorithm** for the following array of numbers: 13, 57, 39, 85, 70, 22, 64, 48 **(5 points)**

3. Use Strassen's method to compute the product of the following two matrices **(10 points)**

A = 2, 3   B = 5, 6

      4, 5   1, 3

4. Illustrate all the steps of the partition function in quicksort on the array A =[13, 9, 5, 7, 3, 1, 10, 6, 11, 2] **(10 points)**

5. Given a sorted list of distinct integers A[1], A[2],...,A[n], you want to find out whether there is an index i for which A[i] = i. Give an algorithm that runs in time O(log n).

For example, A = [-1, -2, 3, 5, 6, 7] does have such an index, i.e., A[3] = 3. But A =[0, 1, 2, 5, 6, 7] doesn't have such an index.

You need to write the pseudocode, prove correctness and analyze the running time. **(15 points)**

6. You are given a loaded die. The probability of obtaining a particular face of the die is inversely proportional to the number on that face. What is the probability that you will get a number greater than 3 if you roll the die? **(5 points)**

Programming Part

What to hand in?

1. Submit code with in-line documentation. **You can write in the assignment in C, C++ or Java.**

2. Run your code on your local machine as well as on 'remote'. A readme file outlining how your code should be run.

Problem 1. (15 points)

In the first part of the problem you will implement the simple selection sort algorithm. You can use an array to store the elements.

a. As you are aware the selection sort algorithm runs in $O(n^2)$ time. Implement your algorithm in a separate function called SelectionSort().

Sample Test Cases

Input: 4, 6, 8, 15, 20, 22, 10, 3, 9, 2

Output: 2, 3, 4, 6, 8, 9, 10, 15, 20, 22

b. In the second part of the assignment you will modify the selection sort algorithm to obtain the first k smallest elements of the array in sorted order (the value of k will be entered by the user). Your algorithm must run in O(nk) time.

Sample Test Case 1

Input: 4, 6, 8, 15, 20, 22, 10, 3, 9, 2

k = 4

Output: 2, 3, 4, 6

Sample Test Case 2

Input: 4, 6, 8, 15, 20, 22, 10, 3, 9, 2

k = 6

Output: 2, 3, 4, 6, 8, 9

c. Use the implementation in part 2 to determine the median value of the input array. You can assume that the values in the array are distinct. If the array is of even size, then the median is the average of the two middle values is the array.

Problem 2. (20 points)

In this problem you will implement a sorting algorithm, which has not been covered in class. We will call this sorting algorithm CoolSort(). We will take advantage of insertion sort for designing this sorting algorithm. The description of the sorting algorithm is as follows.

We will first choose a decreasing sequence of numbers that ends at 1. For example, let us consider the sequence of step sizes H = 5, 3, 1. You can choose any decreasing sequence. Note that the first element of the sequence is less than the number of elements in the array.

For each H, sort sub-arrays that start at arbitrary element and include every H^{th} element using insertion sort. For example, consider the following array

a = [62 83 18 53 07 17 95 86 47 69 25 28].

An example run of CoolSort with gaps 5, 3 and 1 is shown below.

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}
Input	62	83	18	53	07	17	95	86	47	69	25	28

H = 5   17 28 18 47 07 25 83 86 53 69 62 95

H = 3   17 07 18 47 28 25 69 62 53 83 86 95

H = 1   07 17 18 25 28 47 53 62 69 83 86 95

The first pass, 5-sorting, performs insertion sort on separate subarrays (a_1, a_6, a_{11}), (a_2, a_7, a_{12}), (a_3, a_8), (a_4, a_9), (a_5, a_{10}). For instance, it changes the subarray (a_1, a_6, a_{11}) from (62, 17, 25) to (17, 25, 62). The next pass, 3-sorting, performs insertion sort on the subarrays (a_1, a_4, a_7, a_{10}), (a_2, a_5, a_8, a_{11}), (a_3, a_6, a_9, a_{12}). The last pass, 1-sorting, is an ordinary insertion sort of the entire array (a_1,..., a_{12}).

As the example illustrates, the subarrays that CoolSort operates on are initially short; later they are longer but almost ordered.

Though unintuitive, it can be shown that the above algorithm has a runtime of O($N^{3/2}$) in comparison to selection sort which has a runtime of O(N^2). That is why this algorithm is cool!

Sample Test Case 1

Input = [2, 5, 6, 4, 10, 9, 8, 1, 10, 5] and **H** = [5, 3,1]

Output = [1, 2, 4, 5, 5, 6, 8, 9, 10, 10]

Sample Test Case 2

Input = [2, 5, 9, 4, 10, 7, 8, 1, 11, 5] and **H** = [5, 2,1]

Output = [1, 2, 4, 5, 5, 7, 8, 9, 10, 11]

Problem 3. (15 points)

Implement the algorithm you devised in Question 5 and show that it works