Programming Design In-class Practices

Self-defined Data Types (in C)

Ling-Chieh Kung

Department of Information Management
National Taiwan University

Problem 1: bubble sort and insertion sort

- Implement bubble sort and insertion sort (to sort integers from small to large).
- Write two functions:

```
void bubbleSort(const int unsorted[], int sorted[], int len);
void insertionSort(const int unsorted[], int sorted[], int len);
```

  - **`unsorted`** is the original array, **`sorted`** is to store the sorted outcome after the invocation, and **`len`** is the length of the two arrays.
- Input:
  - First integer: n. Then n integers x_1, x_2, …, and x_n.
  - Separated by white spaces.
- Output:
  - The n integers sorted from small to large.
  - Separated by white spaces.

Input:
`7 7 5 3 3 9 6 7`

Output:
`3 3 5 6 7 7 9`

Problem 2: point sorting

- Given a set of points (x_i, y_i), $i = 1, \ldots, n$, sort them in the following way:
  - (x_i, y_i) is in front of (x_j, y_j) if $x_i^2 + y_i^2 < x_j^2 + y_j^2$.
  - If there is a tie, put the one with the smaller x to be in the front.
- You may want to use the structure **`Point`** defined in class.
- Write a function:

```
void vectorSort(const Point unsorted[], int sortedIndices[], int len);
```

  - **`unsorted`** is the original array of unsorted vectors, **`sortedIndices`** is to store the indices of vectors in the sorted manner, and **`len`** is the length of the two arrays.

Problem 2: point sorting

- Input:
  - Line 1: an integer n.
  - Line 2: n integers x_1, x_2, …, and x_n. Separated by white spaces.
  - Line 3: n integers y_1, y_2, …, and y_n. Separated by white spaces.
  - $1 \leq n \leq 100$, $-1000 \leq x_i \leq 1000$, and $-1000 \leq y_i \leq 1000$.
  - No two points are identical.
- Output:
  - n integers as the sorted indices, from the first to the last.
  - Separated by white spaces.

Problem 2: point sorting

- Sample input/output:

Input:
```
3
9 4 5
2 8 7
```

Output:
```
3 2 1
```

Input:
```
3
9 4 5
2 9 9
```

Output:
```
1 2 3
```

Input:
```
4
9 4 5 7
2 8 7 3
```

Output:
```
4 3 2 1
```

Problem 3: random number generation

- Write a function to generate some random integers that are uniformly distributed within 0 and 10,000.

```
void setRN(int rn[], int len);
```

  - **rn** is the array to store random integers, and **len** is the number of random numbers (i.e., the length of **rn**).
- Test the function by yourself.

Problem 4: checking randomness

- Given a set of values, we may create a frequency distribution for it.
  - We need to know the number of bins.
- For example:
  - Values: 9, 2, 3, 16, 12, 21, 2, 26, 32, 14, 17, 23.
  - Number of bins: 4 (0-9, 10-19, 20-29, 30-39).
  - Results:

Bin	Frequency
[0, 10)	4
[10, 20)	4
[20, 30)	3
[30, 40)	1

Problem 4: checking randomness

- Write a function to generate 10,000 random integers that are uniformly distributed within 0 and 10,000.

```
void setRN(int rn[], int len);
```

- Write another function to generate a frequency distribution (to be used to generate a histogram).

```
void hist(const int value[], int len, int freq[], int binCnt);
```

  - **`value`** contains the values to be distributed to the bins (classes/intervals), **`len`** is the number of values, **`freq`** is to store the frequencies of all bins, and **`binCnt`** is the number of bins.
  - You may assume that all values are nonnegative. The first bin should cover 0. All bins should have the same size. All numbers should be covers by your bins. You may make your own choice of bin size.

Problem 5: sampling w/o replacement

- How to generate n integers from 1 to n randomly with no repetition?
  - This is to **sample** n items out of n items **without replacement**.
- A naïve algorithm:
  - In iteration i, first generate a temporary random number r_i'.
  - If r_i' is different from all random numbers r_1, r_2, …, and r_{i-1}, set it to be the ith random number r_i. Otherwise, generate another r_i' and do it again.
- Try to implement this and see how **inefficient** this algorithm is!
- Would you design a better algorithm?