Data Structures Lab
Session 4

Course: Data Structures (CS2001)

Note:
- Lab manual cover following below elementary sorting algorithms
  {Bubble, insertion, selection, shell sort}
- Maintain discipline during the lab.
- Just raise hand if you have any problem.
- Completing all tasks of each lab is compulsory.
- Get your lab checked at the end of the session.

Task-1:

Given an array of strings arr[]. Sort given strings using Bubble Sort and display the sorted array.

Key Points:

1. Bubble Sort, the two successive strings arr[i] and arr[i+1] are exchanged whenever arr[i]> arr[i+1]. The larger values sink to the bottom and hence called sinking sort. At the end of each pass, smaller values gradually "bubble" their way upward to the top and hence called bubble sort.

Task-2:

Develop C++ solution such that day month and year are taken as input for 5 records and perform Sorting Dates based on year using Selection Sort.

Key Points:

```
void selectionSort(int *array, int size) {
Find the smallest element in the array and exchange it with the element in the first position.
Find the second smallest element in the array and exchange it with the element in the second position.
Continue this process until done.
}
```

Task-3:

Develop an implementation of insertion sort that moves larger items to the right one position rather than doing full exchanges.

Key Points:

```
void insertionSort (int *array, int size) {
    Choose the second element in the array and place it in order with respect to the first
    element.
    Choose the third element in the array and place it in order with respect to the first two
    elements.
    Continue this process until done.
    Insertion of an element among those previously considered consists of moving larger
    elements one position to the right and then inserting the element into the vacated
    position
}
```

Task 4:

Given an array **arr[ ]** of length **N** consisting cost of **N** toys and an integer **K** the amount with you. The task is to find maximum number of toys you can buy with **K** amount.

Example 1:
Input:
N = 7
K = 50
arr[] = {1, 12, 5, 111, 200, 1000, 10}
Output: 4
Explaination: The costs of the toys
you can buy are 1, 12, 5 and 10.
Example 2:
Input:
N = 3
K = 100
arr[] = {20, 30, 50}
Output: 3
Explaination: You can buy all toys

Task 5:

Given an unsorted array arr[0..n-1] of size n, find the minimum length subarray arr[…] such that sorting this subarray makes the whole array sorted.
Examples:
1) If the input array is [10, 12, 20, 30, 25, 40, 32, 31, 35, 50, 60], your program should be able to find that the subarray lies between the indexes 3 and 8.
2) If the input array is [0, 1, 15, 25, 6, 7, 30, 40, 50], your program should be able to find that the subarray lies between the indexes 2 and 5.

Key Points:

Traverse the array from left to right tracking the max, saving the last found index value of 'j' which will be less than max
Traverse the array from right to left tracking the min, saving the last found index value of 'i' which will be greater than max
Sort the segment array from index i to j

Task 6:

A clerk at a shipping company is charged with the task of rearranging a number of large crates in order of the time they are to be shipped out. Thus, the cost of compares is very low relative to the cost of exchanges (move the crates). The warehouse is nearly full: there is extra space sufficient to hold any one of the crates, but not two. Which sorting method should the clerk use?

Discussion:

- Selection sort builds the final diagonal by exchanging the next-smallest element into position. The main cost of selection sorts is the comparisons required to find that element. Comparisons are not depicted in this representation except for a delay to make the time take for a comparison comparable to the cost of an exchange. The algorithm is slow at the beginning (because it has to scan through most of the array to find the next minimum) and fast at the end (because it has to scan through only a few elements).Selection sort is easy to implement; there is little that can go wrong with it. However, the method requires O(N^2) comparisons and so it should only be used on small files. There is an important exception to this rule. When sorting files with large records and small keys, the cost of exchanging records controls the running time. In such cases, selection sort requires O(N) time since the number of exchanges is at most N.

- Bubble sort works like selection sort, but its cost is explicit in this representation because it uses exchanges (data movement) to move the minimum element from right to left, one position at a time.

- Insertion sort maintains a ordered subarray that appears as a diagonal of black dots that moves from left to right sweeping up each new element that it encounters. Each new element is inserted into the diagonal. Insertion sort is an O(N^2) method both in the average case and in the worst case. For this reason, it is most effectively used on files with roughly N < 20. However, in the special case of an almost sorted file, insertion sort requires only linear time.

- Time complexity of shell sort is O(n^2). Shell sort is a highly efficient sorting algorithm and is based on insertion sort algorithm. This algorithm avoids large shifts as in case of insertion sort, if the smaller value is to the far right and has to be moved to the far left. Shell algorithm uses insertion sort on a widely spread elements, first to sort them and then sorts the less widely spaced elements. This spacing is termed as **interval**. This interval is calculated based on Knuth's formula as −

```
Knuth's Formula
h = h * 3 + 1
where −
h is interval with initial value 1
```

Reference:

For algorithms:
http://web.mit.edu/1.124/LectureNotes/sorting.html
For complexities and comparison
https://www.geeksforgeeks.org/analysis-of-different-sorting-techniques/
Algorithm visualizer
https://visualgo.net/en