BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI HYDERABAD CAMPUS

Data Structures and Algorithms (CS F211 / IS F211)

Homework Assignment – 5

1. **Bidirectional bubble sort** (BBS) is a slightly improved version of bubble sort (BS) to keep track of "turtles": The first rightward pass will shift the largest element to its correct place at the end, and the following leftward pass will shift the smallest element to its correct place at the beginning. The second complete pass will shift the second largest and second smallest elements to their correct places, and so on. After *i* passes, the first *i* and the last *i* elements in the list are in their correct positions, and do not need to be checked. By shortening the part of the list that is sorted each time, the number of operations can be halved.
   Ex. Sort {2,3,4,5,1} using BBS
   1st rightward pass: {2 3 4 1 5}
   1st Leftward pass: {1 2 3 4 5}
   While BBS needs only two passes, Bubble sort would need four passes (convince yourself by tracing the above example).

   Write a program to generate 10, 10000, 25000, and 50000 numbers randomly (use srand() and rand()) and sort them using the following algorithms. Assume the integers are bound in the range [0, 10000000] for all these questions. Use malloc() to store your array of numbers. Measure the time taken to run each of the algorithm and print it (use time() and clock() functions). You are free to define the number and size of the bucket in case of bucket sort. Also print unsorted vs sorted output for the first case with only 10 numbers for each algorithm. **Note: don't ask for any input from user.**
   a. Bubble sort
   b. Bidirectional bubble sort
   c. Insertion sort

2. **Comb sort** (CS) is still another variation of bubble sort (BS): The basic idea of comb sort is that the gap between two elements that are compared in the inner for loop of BS can be much more than 1. The inner loop of bubble sort, which does the actual *swap*, is modified such that gap between swapped elements goes down (for each iteration of outer loop) in steps of a "shrink factor" *k (=1.3)*: [n/k, n/k^2, n/k^3, ..., 1].

   The gap starts out as the length of the list *n* being sorted divided by the shrink factor *k* (= 1.3) and one pass of the aforementioned modified bubble sort is applied with that gap. Then the gap is divided by the shrink factor again, the list is sorted with this new gap, and the process repeats until the gap is 1. At this point, comb sort continues using a gap of 1 (as in BS) until the list is fully sorted.

   Write a program to generate 10, 10000, 50000, 100000, and 500000 numbers randomly (use srand() and rand()) and sort them using the following algorithms.

Assume the integers are bound in the range [0, 10000000] for all these questions. Use malloc() to store your array of numbers. Measure the time taken to run each of the algorithm and print it. Also print unsorted vs sorted output for the first case with only 10 numbers for each. **Note: don't ask for any input from user.**

a. Comb sort
b. Merge sort
c. Quick sort

3. A majority element in an array A[] of size n is an element that appears more than n/2 times (and hence there is at most one such element). Write a function which takes an array and emits the majority element (if it exists), otherwise prints NONE in O(n) as follows:
   Example:

   Input1:
   9
   3 3 4 2 4 4 2 4 4
   Output1: 4
   Input2:
   9
   3 3 3 4 2 4 4 2 4
   Output2: NONE

4. Let A be an array of n distinct positive integers. If i < j and A[i] > A[j] then the pair (i, j) is called an inversion of A. Given n and an array A your task is to find the number of inversions of A in O(n logn). Also print all pairs of inversions.
   (Note : You cannot use global variables.)

5. A contractor is given a job to renovate the old building. Each day's work has been decided and the according to work, each day **distinct** number of workers are hired for that day. The contractor is interested in knowing that for each day, how many days in future are there when number of workers required is less than the number of workers on the current day. You should solve this problem in O(n log n)
   Input Format:
   N
   N **distinct** numbers separated by space each number representing the no. of workers required for that day.
   Constraints: N ≤ 1000 and noOfWorkers[i] ≤ 100
   output: a line contains N integer separated by a space, where ith integer represents the number of days of future, when no. of workers is less than the current day.
   Input :
   7
   5 8 1 6 3 2 7
   output : 3 5 0 2 1 0 0
   Explanation: For day-1, 5 workers are there and in future only on day-3, day-5, day6 no of workers are 1, 3, 2 < 5 hence 3 in output for day-1 and similarly for other days.

6. Given two arrays A1[] and A2[], sort A1 in such a way that the relative order among the elements will be same as those are in A2. For the elements not present in A2, append them at last in sorted order. Your program should handle all cases like number of elements in A2[] may be more or less compared to A1[]. A2[] may have some elements which may not be there in A1[] and vice versa is also possible.
   Example:
   Input: A1[] = {2, 1, 2, 5, 7, 1, 9, 3, 6, 8, 8}
   A2[] = {2, 1, 8, 3}
   Output: A1[] = {2, 2, 1, 1, 8, 8, 3, 5, 6, 7, 9}

7. Write an algorithm called Longest-Repeated-Substring (T) that takes a string T representing some text, and finds the longest string that occurs at least twice in T. If no such substring exist, then the algorithm returns "No substring found"
   Input: a b c a c a n a g r t m m t t k k c a c a n a g r a g c a d d c c
   Output: c a c a n a g r

8. Design an efficient in-place algorithm called Partition-Even-Odd (Arr) that partitions an array **Arr** in even and odd numbers. The algorithm must terminate with **Arr** containing all its even elements preceding all its odd elements.
   For example: Arr = 7, 17, 74, 21, 7, 9, 26, 10
   The result might be Arr = 74, 26, 10, 21, 7, 9, 17, 7

9. Write an in-place partition algorithm called Modulo-Partition (**A**) that takes an array **A** of **n** numbers and changes **A** in such a way that (1) the final content of **A** is a permutation of the initial content of **A**, and (2) all the values that are equivalent to **0 mod 10** precede all the values equivalent to **1 mod 10**, which precede all the values equivalent to **2 mod 10**, etc.
   For example, A = 7, 62, 5, 57, 12, 39, 5, 8, 16, 48
   A = 12, 62, 5, 5, 16, 57, 7, 8, 48, 39

10. Implement a dictionary data structure that supports longest prefix matching. Specifically, write the following two algorithms:
    - Build-Dictionary(W) takes a list W of n strings and builds the dictionary.
    - Longest-Prefix(k) takes a string k and returns the longest prefix of k found in the dictionary, or NULL if none exists. Find methods to improve time complexity of Longest-Prefix(k).
      For example, assuming the dictionary was built with strings, "luna", "lunatic", "a", "al", "algo", "an", "anto", then if k is "algorithms", then Longest-Prefix(k) should return "algo", or if k is "anarchy" then Longest-Prefix(k) should return "an", or if k is "kugano" then Longest-Prefix(k) should return NULL.

-----------------------------------XXXXXXXXXXXXXXXXXXXXXXXXXXX-----------------------------------