Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Data Structures and Algorithms Lab
Assignment 1

Problem 1

In this problem we will implement three simple sorting algorithms, namely 'insertion sort', 'selection sort' and 'merge sort' for sorting an array of n integers in the ascending order. Let $A[1, \ldots, n]$ denote the array of integers which is initially unsorted.

Insertion sort: After k iterations, contents of $A[1, \ldots, k]$ are in the sorted order. Initially $k = 0$. In iteration $k + 1$, element $A[k + 1]$ is inserted in the partially sorted sub-array $A[1, \ldots, k]$ by shifting elements in $A[1, \ldots, k]$ that are greater than $A[k + 1]$ to the right. Thus, $A[1, \ldots, k + 1]$ is sorted after $k + 1$ iterations. Continue for n iterations to get fully sorted array.

Selection sort: Find the minimum value in $A[1, \ldots, n]$ and swap it with $A[1]$. Repeat the same step now for the sub-array $A[2, \ldots, n]$. After n iterations, the resulting array is sorted.

Merge sort: If the array has only one element, return the array as sorted. Else divide the array into two smaller arrays: $A[1, \ldots, n/2]$ and $A[n/2 + 1, \ldots, n]$ and recursively call merge sort on both these arrays. After sorting the smaller arrays recursively, merge the two sorted arrays in $O(n)$ time.

1. Read about these sorting algorithms for more details.
2. Implement Insertion sort, Selection sort and Merge sort. Your program should read the input from a file, where each element is stored in a separate line. The output also should be written into a file in the same format.
3. Analyze theoretically the space and time complexity of your algorithms in terms of the number of elements to be sorted.
4. Create inputs of sizes $10, 100, 1000, 10000$ and 100000 and measure the running time of both sorting algorithms for these inputs. For this experiment, you can assume

that for a given n (say 1000), the input is $\{1, 2, 3, \ldots, 1000\}$ in an arbitrary order. You need to write a small program to create such inputs for different values of n. The objective is to create candidate inputs that allow a true performance comparison of the two sorting algorithms.

5. Compare the performance of the three sorting algorithms based on your theoretical analysis and experimental evaluation.

6. The above findings should be submitted either as a written report, or as a typesetted document. In any case, you should include a pdf file (scan it if you hand wrote the report) of the above findings in your submission.

Problem 2

In this problem, you have to implement an 'ArrayLinkedList' class which is an array based doubly linked list implementation. The list elements are integer values. The linked list will be maintained on a pre-allocated array of given size. Each array record, which also corresponds to list record, will contain as usual three fields, namely, (a) *key* which holds the element (b) *prev* pointer and (c) *next* pointer. Note that the pointer values are array indices.

Your implementation should maintain two lists, namely *active list* and *free list*. The active list correspond to the linked list and free list holds the array records that are free. When a new element is inserted into the list, a record is removed from the free list and inserted into the active list. Similarly, when an element is removed from the active list, it is inserted back into the free list. Initially all the array records belong to the free list.

The ArrayLinkedList class should support the following:

1. ArrayLinkedList(int arraysize) - Constructor that allocates an array of size given by the 'arraysize' parameter. Also, the active list is initialized to be nil and free list is initialized to contain all the array records.

2. Insert(int x, int v) - Insert the element v into the list immediately after the first occurrence of x in the list. It should return false if either x is not present in the list or if the list is already full.

3. Delete(int v) - Delete the first occurrence of v in the list if it exists. Returns false if v is not present in the list. This function should take $O(m)$ time where m is the size of active list.

4. Search(int v) - Returns true if v is present in the list. Search also should take $O(m)$ time.

5. CompactList() - Moves elements in active and free lists in such a way that the elements in active list occupy array positions $0, \ldots, m-1$, where m is the size of the active list, and free list elements occupy the remaining array positions. Your procedure should take only $O(m)$ time.

6. PrintList() - Prints the list elements.

7. PrintArray() - Print the 'key' value stored in the array indices $0, 1, 2, \ldots, N-1$, in the same order, where N is the array size.