|  |  |  |
| --- | --- | --- |
| **American International University-Bangladesh**  **Faculty of Science and Information Technology**  **Department of Computer Science**  **CSC 2211: Algorithms [A,P,K]** | | |
|  | **Selection Sort:**  Suppose A is an array of N values. We want to sort A in ascending order. That is, A [1] should be the smallest and A[N] should be the largest.  The idea of Selection Sort is that we repeatedly find the smallest element in the unsorted part of the array and swap it with the first element in the unsorted part of the array. |  |
|  | **Insertion Sort:**  Suppose A is an array of N values. We want to sort A in ascending order.  Insertion Sort is an algorithm to do this as follows: We traverse the array and insert each element into the sorted part of the list where it belongs. This usually involves pushing down the larger elements in the sorted part. |  |
|  | **Bubble Sort:**  Suppose A is an array of N values. We want to sort A in ascending order.  Bubble Sort is a simple-minded algorithm based on the idea that we look at the list, and wherever we find two consecutive elements out of order, we swap them. We do this as follows: We repeatedly traverse the unsorted part of the array, comparing consecutive elements, and we interchange them when they are out of order. The name of the algorithm refers to the fact that the largest element "sinks" to the bottom and the smaller elements "float" to the top.  **procedure bubbleSort(A : list of sortable items )**  **n = length(A)**  **repeat**  **swapped = false**  **for i = 1 to n-1 inclusive do**  **/\* if this pair is out of order \*/**  **if A[i-1] > A[i] then**  **/\* swap them and remember something changed \*/**  **swap( A[i-1], A[i] )**  **swapped = true**  **end if**  **end for**  **until not swapped**  **end procedure** |  |
|  | Consider the following searching problem:  **Input:** A sequence of n numbers **A** = <a1, a2, . . . , an> and a value **V**.  **Output:** An index **i** such that **V = A[i]** or the special value nil if **V** does not appear in **A**.  Write a C/C++ code for linear search, which scans through the sequence, looking for **V**.”   |  |  | | --- | --- | | **Sample Input:**  Enter the size of array:  6  Enter the elements of array:  1,5,3,7,9,5  Enter the value you want to search?  7  **Output:**  Found at index number 3. | **Sample Input:**  Enter the size of array:  6  Enter the elements of array:  1,5,3,7,9,5  Enter the value you want to search?  4  **Output:**  nil | |  |
|  | For the following activity selection problem:  **Input:** Set S of n activities, a1, a2, . . . , an .  si = start time of activity i.  fi = finish time of activity i.  **Output:** Subset A of maximum number of compatible activities.  **NB:** Two activities are compatible, if their intervals don’t overlap.  D:\McGraw-Hill Projects\Cormen\algorithms\greedy_activity_selector.gif  **Sample Input:**  Enter the size of array:  6  Enter the elements of Start time array:  1 3 0 5 8 5  Enter the elements of Finish time array:  2 4 6 7 9 9  **Sample Output:**  Following activities are selected  0 1 3 4 |  |
|  | | |
|  | **Instruction for submitting assignment:**   1. First of all implement all the problem using C/C++. 2. For each problem name the file by its problem number. (**Ex: for problem 1 the file name will be 1.cpp or 1.c**) 3. Make a **ZIP Folder** named your **student Id (Ex: 14-26557-2)** containing all the code. 4. **Upload the ZIP folder** in the portals under the Lab performance\assignment folder. 5. Make a **handwritten copy** of your code in the following format (Applicable for each code):   *a. Assertions*  *b. Code Implementations*  *c. Complexity Analysis*   1. Submit both **Soft copy and hard copy** within due time. 2. Submission deadline: **28-02-2019** |  |