PES University
End Semester Assessment (ESA) May 2016
UE14CS255: Design and Analysis of Algorithms Laboratory
Question Bank

Q. No.	Questions
1	Search for a student record by student's name in an array of student records using **Sequential Search** algorithm. Print the student record and the execution time. Sort the student records using **Selection Sort** algorithm. On the sorted array, search using **Binary Search** and print the student record in the array and the execution time.
2	Search for an integer in an array of integers using **Sequential Search** algorithm. Print the index of the integer in the array and the execution time. Sort the array using **Bubble Sort** algorithm. On the sorted array, search using **Binary Search** and print the index of the integer in the array and the execution time.
3	Sort an array of integers using **Insertion Sort** algorithm and print the sorted array and the execution time. Sort an array of integers using **Heap Sort** algorithm and print the sorted array and the execution time.
4	Sort an array of student records by student's CGPA using **Merge Sort**. Print the sorted array and the execution time. Now, sort the array using **Sorting by Distribution Counting** algorithm. Print the sorted array and the execution time.
5	Sort an array of student records by student's register number using **Quick Sort**. Print the sorted array and the execution time.
6	Sort an array of student records by student's name using **Merge Sort**. Print the sorted array and the execution time. Now, sort the array using **Bubble Sort** algorithm. Print the sorted array and the execution time.
7	In a long text of length n characters, search for a pattern of length m characters using **Brute Force String Matching** algorithm. Print the index of first occurarace and the number of single-character comparisons. Now, search for a pattern using **Horspool's** algorithm. Print the index of first occurarace and the number of single-character comparisons.
8	Insert a given sequence of integers into a **Binary Search Tree** and print the tree in **BFS**-order.
9	Solve **0/1 Knapsack Problem** using Dynamic Programming for a given list of n items with weight w_i and value v_i , and knapsack capacity W.
10	Find the transitive closure of a directed graph having n vertices using **Warshall's** algorithm.
11	Find all pairs shortest paths in a directed graph having n vertices using **Floyd's** algorithm.
12	Find single source shortest paths in a directed graph having n vertices using **Dijkstra's** algorithm.

Lab In-charge **Chairperson**

Dept. of CS&E

Q. No. **01**: Search for a student record by student's name in an array of student records using **Sequential Search** algorithm. Print the student record and the execution time. Sort the student records using **Selection Sort** algorithm. On the sorted array, search using **Binary Search** and print the student record in the array and the execution time.

Input: Input begins with n ($1 \le n \le 2^{20}$) of number of student records. The following n lines has a student record per line. The following line has a student's name (one alphabetic word without any spaces; $1 \le$ length of the name ≤ 20). A student record of a student has a unique register number (one alphanumeric word; $3 \le$ length of the word ≤ 10), his/her name (one alphabetic word without any spaces; $1 \le$ length of the name ≤ 20) and his/her CGPA with the precision up to two digits after decimal point ($00.00 \le$ CGPA ≤ 10.00). The three fields of a record are separated by a spaces.

Output: Print the first student record matched by Sequential Search (print "NOT FOUND" in case of unsuccessful search) in the first line and print the time taken (in milliseconds) in a new line. Print the student record searched by Binary Search (print "NOT FOUND" in case of unsuccessful search) in a new line and print the time taken (in milliseconds) in a new line.

Example: Input:
10
CS003 Vinay 10
CS005 Mouli 9.94
CS010 Gautham 9.94
CS020 Sneha 9.94
CS200 Mohit 9.93
CS012 Aarti 9.9
CS006 Darshan 9.88
CS002 Adithya 9.78
CS050 Karthik 9.71
CS001 Adithya 9.58
Adithya

Output:
CS002 Adithya 9.78
123.456 (Note: This value in milliseconds may be different for each execution)
CS001 Adithya 9.58 (Note: any student record with matching name)
12.345 (Note: This value in milliseconds may be different for each execution)

Q. No. **02**: Search for an integer in an array of integers using **Sequential Search** algorithm. Print the index (0-based) of the integer in the array and the execution time. Sort the array using **Bubble Sort** algorithm. On the sorted array, search using **Binary Search** and print the index of the integer in the array and the execution time.

Input: Input begins with n ($1 \leq n \leq 2^{20}$) of number of integers. The following n lines has an integer per line ($-2^{20} \leq$ integer $\leq 2^{20}$). The following line has an integer to be searched.

Output: Print the index (0-based) of the first integer matched by Sequential Search (print "NOT FOUND" in case of unsuccessful search) in the first line and print the time taken (in milliseconds) in a new line. Print the index of the integer searched by Binary Search (print "NOT FOUND" in case of unsuccessful search) in a new line and print the time taken (in milliseconds) in a new line.

Example: Input:
6
999999
1234
0
-999999
1234
-1234
1234

Output:
1
123.456 (Note: This value in milliseconds may be different for each execution)
4 (Note: index of any element with matching integer)
12.345 (Note: This value in milliseconds may be different for each execution)

Q. No. **03**: Sort an array of integers using **Insertion Sort** algorithm and print the sorted array and the execution time. Sort an array of integers using **Heap Sort** algorithm and print the sorted array and the execution time.

Input: Input begins with n ($1 \leq n \leq 2^{20}$) of number of integers. The following n lines has an integer per line ($-2^{20} \leq$ integer $\leq 2^{20}$).

Output: Print the sorted array one integer per line sorted by Insertion Sort. Print the time taken (in milliseconds) in a new line. Print the sorted array one integer per line sorted by Heap Sort. Print the time taken (in milliseconds) in a new line.

Example: Input:
6
999999
1234
0
-999999
1234
-1234

Output:
-999999
-1234
0
1234
1234
999999
123.456 (Note: This value in milliseconds may be different for each execution)
-999999
-1234
0
1234
1234
999999
12.345 (Note: This value in milliseconds may be different for each execution)

Q. No. **04**: Sort an array of student records by student's CGPA using **Merge Sort**. Print the sorted array and the execution time. Now, sort the array using **Sorting by Distribution Counting** algorithm. Print the sorted array and the execution time.

Input: Input begins with n ($1 \leq n \leq 2^{20}$) of number of student records. The following n lines has a student record per line. A student record of a student has a unique register number (one alphanumeric word; 3 ≤ length of the word ≤ 10), his/her name (one alphabetic word without any spaces; 1 ≤ length of the name ≤ 20) and his/her CGPA with the precision up to two digits after decimal point (00.00 ≤ CGPA ≤ 10.00). The three fields of a record are separated by a spaces.

Output: Print the sorted array of student records one record per line sorted by Merge Sort. Print the time taken (in milliseconds) in a new line. Starting from a new line, print the sorted array of student records one record per line sorted by Sorting by Distribution Counting. Print the time taken (in milliseconds) in a new line.

Example: Input:
6
CS050 Karthik 9.71
CS005 Mouli 9.94
CS003 Vinay 10
CS002 Adithya 9.78
CS001 Adithya 9.58
CS010 Gautham 9.94

Output:
CS003 Vinay 10
CS005 Mouli 9.94
CS010 Gautham 9.94
CS002 Adithya 9.78
CS050 Karthik 9.71
CS001 Adithya 9.58
123.456 (Note: This value in milliseconds may be different for each execution)
CS003 Vinay 10
CS005 Mouli 9.94
CS010 Gautham 9.94
CS002 Adithya 9.78
CS050 Karthik 9.71
CS001 Adithya 9.58
12.345 (Note: This value in milliseconds may be different for each execution)

Q. No. **05**: Sort an array of student records by student's register number using **Quick Sort**. Print the sorted array and the execution time.

Input: Input begins with n ($1 \le n \le 2^{20}$) of number of student records. The following n lines has a student record per line. A student record of a student has a unique register number (one alphanumeric word; 3 ≤ length of the word ≤ 10), his/her name (one alphabetic word without any spaces; 1 ≤ length of the name ≤ 20) and his/her CGPA with the precision up to two digits after decimal point (00.00 ≤ CGPA ≤ 10.00). The three fields of a record are separated by a spaces.

Output: Print the sorted array of student records one record per line sorted by Quick Sort. Print the time taken (in milliseconds) in a new line.

Example: Input:
6
CS050 Karthik 9.71
CS005 Mouli 9.94
CS003 Vinay 10
CS002 Adithya 9.78
CS001 Adithya 9.58
CS010 Gautham 9.94

Output:
CS003 Vinay 10
CS005 Mouli 9.94
CS010 Gautham 9.94
CS002 Adithya 9.78
CS050 Karthik 9.71
CS001 Adithya 9.58
123.456 (Note: This value in milliseconds may be different for each execution)

Q. No. **06**: Sort an array of student records by student's name using **Merge Sort**. Print the sorted array and the execution time. Now, sort the array using **Bubble Sort** algorithm. Print the sorted array and the execution time.

Input: Input begins with n ($1 \le n \le 2^{20}$) of number of student records. The following n lines has a student record per line. A student record of a student has a unique register number (one alphanumeric word; 3 ≤ length of the word ≤ 10), his/her name (one alphabetic word without any spaces; 1 ≤ length of the name ≤ 20) and his/her CGPA with the precision up to two digits after decimal point (00.00 ≤ CGPA ≤ 10.00). The three fields of a record are separated by a spaces.

Output: Print the sorted array of student records one record per line sorted by Merge Sort. Print the time taken (in milliseconds) in a new line. Starting from a new line, print the sorted array of student records one record per line sorted by Bubble Sort. Print the time taken (in milliseconds) in a new line.

Example: Input:
6
CS050 Karthik 9.71
CS005 Mouli 9.94
CS003 Vinay 10
CS002 Adithya 9.78
CS001 Adithya 9.58
CS010 Gautham 9.94

Output:
CS003 Vinay 10
CS005 Mouli 9.94
CS010 Gautham 9.94
CS002 Adithya 9.78
CS050 Karthik 9.71
CS001 Adithya 9.58
12.345 (Note: This value in milliseconds may be different for each execution)
CS003 Vinay 10
CS005 Mouli 9.94
CS010 Gautham 9.94
CS002 Adithya 9.78
CS050 Karthik 9.71
CS001 Adithya 9.58
123.456 (Note: This value in milliseconds may be different for each execution)

Q. No. **07**: In a long text of length n characters, search for a pattern of length m characters using **Brute Force String Matching** algorithm. Print the index of first occurarace and the number of single-character comparisons. Now, search for a pattern using **Horspool's** algorithm. Print the index of first occurarace and the number of single-character comparisons.

Input: Input begins with a text ($1 \leq$ length of the text $\leq 2^{20}$) in a single line and the pattern to be searched in a new line. The text and pattern could have spaces as characters.

Output: Print the 0-based index ($0 \leq$ index $<$ length of the text) of the beginning the pattern in the text matched by the Brute Force Sting Matching algorithm in a new line. Print '-1' if the pattern is not found in the text. With a space as a seperation, print the number of single-character comparisons made whether they were matching or not. In a new line, print similar index and number of single-character comparisons made by the Horspool's algorithm.

Example: Input:
Discrete Mathematics and Logic is a prerequisite for Design and Analysis of Algorithms.
and Logic

Output:
21 31
21 12

Example: Input:
Discrete Mathematics and Logic is a prerequisite for Design and Analysis of Algorithms.
Discrete and

Output:
-1 85
-1 13

Q. No. **08**: Insert a given sequence of integers into a **Binary Search Tree** and print the tree in **BFS**-order.

Input: Input begins with n ($1 \le n \le 2^{20}$), which is the number of integers to be inserted into the BST. It is followed by n lines each having an integer to be inserted into the BST in the given order ($-2^{20} \le$ integer $\le 2^{20}$).

Output: Print the n integers in the sequence as per the BFS-order traversal on the BST, one per line.

Example:
Input:
4
0
999999
-999999
1234

Output:
0
-999999
999999
1234

Q. No. **09**: Solve 0/1 **Knapsack Problem** using Dynamic Programming for a given list of n items with weight w_i and value v_i , and knapsack capacity W.

Input: Input begins with n ($1 \leq n \leq 1000$) in a single line, which is the number of items. It is followed by weights of n items seperated by spaces in a single line (weights are integers; $1 \leq$ weight ≤ 100). It is followed by values of n items seperated by spaces in a single line (values are integers; $1 \leq$ values ≤ 100). The last line has the capacity of the Knapsack, W.

Output: Print the solution of the Knapsack Problem in a single line.

Example: Input:
4
1 1 4 2
1 2 10 2
5

Output:
12

Example: Input:
5
1 2 4 3 5
10 5 41 30 52
8

Output:
82

Q. No. **10**: Find the transitive closure of a directed graph having n vertices using **Warshall's** algorithm.

Input: Input begins with n of number of vertices (1 ≤ n ≤ 1000). The following n lines has the adjacency matrix of the directed graph where the elements of a row are seperated by spaces (an element is either 0 or 1).

Output: Print the transitive closure matrix printed one row per line and space separated elements within a row.

Example: Input:
4
0 0 0 1
1 0 0 0
0 0 0 1
0 1 0 0

Output:
1 1 0 1
1 1 0 1
1 1 0 1
1 1 0 1

Q. No. 11: Find all pairs shortest paths in a directed graph having n vertices using **Floyd's** algorithm.

Input: Input begins with n of number of vertices of the given directed graph ($1 \leq n \leq 1000$). The following n lines has the cost adjacency matrix of the directed graph where the elements of a row are seperated by spaces ($1 \leq$ edge cost ≤ 1000). For a pair of vertices with no edge between them, a weight of 1000000 would be given, which is practically infinity for us (you can be sure that no other edges and the shortest paths would be of length 1000000 or more). And, cost of self loops are always zero.

Output: Print the distance matrix printed one row per line and space separated elements within a row.

Example: Input:
4
0 1000000 3 1000000
2 0 1000000 1000000
1000000 7 0 1
6 1000000 1000000 0

Output:
0 10 3 4
2 0 5 6
7 7 0 1
6 16 9 0

Example: Input:
5
0 3 7 1000000 1000000
2 0 6 8 1000000
9 2 0 1000000 1
2 3 4 0 1000000
1000000 2 1 3 0

Output:
0 3 7 11 8
2 0 6 8 7
4 2 0 4 1
2 3 4 0 5
4 2 1 3 0

Q. No. **12**: Find single source shortest paths in a directed graph having n vertices using **Dijkstra's** algorithm.

Input: Input begins with n of number of vertices of the given directed graph (1 ≤ n ≤ 1000). The following n lines has the cost adjacency matrix of the directed graph where the elements of a row are seperated by spaces (1 ≤ edge cost ≤ 1000). For a pair of vertices with no edge between them, a weight of 1000000 would be given, which is practically infinity for us (you can be sure that no other edges and the shortest paths would be of length 1000000 or more). And, cost of self loops are always zero. Following the cost adjacency matrix, the new line has the 0-based index of the source vertex and destination vertex seperated by a space.

Output: Print the shortest distance and the shortest path (sequence of vertices of the shortest path).

Example: Input:
4
0 1000000 3 1000000
2 0 1000000 1000000
1000000 7 0 1
6 1000000 1000000 0
2
0

Output:
7
2 3 0

Example: Input:
5
0 3 1000000 7 1000000
3 0 4 2 1000000
1000000 4 0 5 6
7 2 5 0 4
1000000 1000000 6 4 0
0
4

Output:
9
0 1 3 4