Assignment 2- Searching and Sorting

About the Assignment

This assignment is based on Sorting and Searching. Implement ALL questions. Your program MUST take input from text files, which is formatted as specified below, and/or with each question.

Measure the running time of your algorithms (use the time measuring options available, in the programming language you are using to implement the algorithms). Compare the variation of running time of each algorithm, corresponding to the variation in the size of the input.

Each question should be tested against input files that contain various number of data items. For Insertion sort, Merge sort, Quick Sort and heap Sort the number of data items should be of the order of 10^i, where i = 1, 2, 3, …, 8.

Common Input Format Searching

The first line of the input file contains a positive integer **n**, the number of data items to be sorted.

Then, **n** lines follow, each line containing exactly one data item.(In sorted order for Binary search)

Next line contains the number to be searched for

Sorting

The first line of the input file contains a positive integer **n**, the number of data items to be sorted.

Then, **n** lines follow, each line containing exactly one data item.

Common Output Format

The (sorted) output must contain exactly **n+1** lines.

The first **n** lines must contain exactly one data item.

The last line must contain a single line, that prints the measured running time of the sorting algorithm, in the format as in the example given here: "**Running time: 2.013 sec**".

Sample Input

Shall be provided with each question.

Searching

1. LINSEARCH

Given an array of **n** integers, write a program to search whether a particular integer, say **k**, is present in the array. If present, return the index of the same. Otherwise, return -1. Use linear search.

Input: An array of **n** integers and **k** the integer to be searched for.

Output: If k is present in the array return the index of k ; otherwise return -1.

2. BINSEARCH

Given a sorted array of **n** integers, write a program to search whether a particular integer, say **k**, is present in the array. If present, return the index of the same. Otherwise, return -1. Use binary search.

Input: An array on **n** integers sorted in ascending order and **k** the integer to be searched for.

Output: If k is present in the array return the index of k ; otherwise return -1.

Sorting

1. INSERTION SORT

a) Implement the insertion sort algorithm, to sort **n real numbers**.

Input: An array on **n** real numbers, $<a_1,a_2,\ldots\ldots a_n>$.

Output: The **n** real numbers in sorted order. $<a_{1'},a_{2'},\ldots\ldots a_{n'}>$ such that $a_1 \le a_2 \le \ldots\ldots \le a_n$.

<u>Sample Input</u>

```
6
71.2
-4.56789
4
0.00
-0.8907
45.66
```

2. SELECTION SORT

Given a list of **n words**, implement the selection sort algorithm, to sort them in lexicographical order. The words contain only lower case English letters (**a** – **z**).

Input: An array on **n** words consisting of only lower case English letters . Output: The n words sorted in lexicographical order.

Sample Input

6

nitc sorting

datastructures

sort select

insert

3. QUICK SORT

Implement the quick sort algorithm, to sort **n integers**.

Input: An array on **n** real numbers <a_1,a_2,……a_n> .

Output: The n real numbers in sorted order. <$a_{1'}$,$a_{2'}$,……$a_{n'}$> such that $a_1 \le a_2 \le \ldots \le a_n$.

Sample Input

6

712

-45

456765

0

-8907

4566

4. RADIX SORT

Implement the radix sort algorithm, to sort **n** non-negative **hexadecimal numbers**. The numbers can range from **0000 0000** to **FFFF FFFF**. The numbers must be sorted in hexadecimal format itself and not after conversion to integer format.

Input: An array on **n** hexadecimal numbers, <a_1,a_2,……a_n> .

Output: The n hexadecimal numbers in sorted order. <$a_{1'}$,$a_{2'}$,……$a_{n'}$> such that $a_1 \le a_2 \le \ldots \le a_n$.

Sample Input

6

BEE

CAFEF1FA

00000000

7CD

101248

A5A5A5A5

5. MERGE SORT

Implement the merge sort algorithm to sort **n integers.**

Input: An array on **n** integers, $<a_1,a_2,\ldots\ldots a_n>$.

Output: The n integers in sorted order. $<a_{1'},a_{2'},\ldots\ldots a_{n'}>$ such that $a_1 \leq a_2 \leq \ldots\ldots \leq a_n$.

Sample Input

6

712

-45

456765

0

-8907

4566

6. HEAP SORT

Implement the heap sort algorithm to sort **n integers**.

Input: An array on **n** integers, $<a_1,a_2,\ldots\ldots a_n>$.

Output: The n integers in sorted order. $<a_{1'},a_{2'},\ldots\ldots a_{n'}>$ such that $a_1 \leq a_2 \leq \ldots\ldots \leq a_n$.

Sample Input

6

712

-45

456765

0

-8907

4566