CS19001: Programming and Data Structures Laboratory

Assignment No. 11A (Sorting and Searching)

Date: 08-November-2019

Problem Statement:

In the city of Wonderland, once the Mad-Hatter gifted Alice a magic-box in her birthday, which can be opened by dialing a fixed number k from its opening panel. Also, the Mad-Hatter gave Alice n sorted integers (in ascending order) within which one is k (but never conveyed Alice about k). The only information (hint) that Alice can seek from the opening display panel of the magic-box is as follows:

- If she dials the correct number, then the magic-box opens.
- If she dials a wrong number, then the display panel shows that the dialed number is more or less than k.

Based on this information, Alice went on trying new numbers, but opened the magic-box very efficiently. You (being computationally smarter than Alice) can immediately point out that *Binary-Search* might have been going through Alice's mind and that is the reason she cracked the opening key of magic-box efficiently. *Yes! you are spot on in solving the problem that Alice faced!*

But, your job is to write a C-program that will not perform exactly as the binary search, but a *variant* of this. Consider the variation of binary search where the sorted integer-array of size n is divided into two parts, but everytime by choosing the $(\frac{a}{b}n)$-th element instead of the middle element $(a < b)$. The choice of a and b are user inputs (You can easily make out that the standard binary search assumes $a = 1$ and $b = 2$ and partition in the middle). The algorithm you may use is given as follows:

- Compare k (the searched element) with the $(\frac{a}{b}n)$-th element
- If equal, k found – return
- If k is smaller, search left sub-array indexed from 0 to $(\frac{a}{b}n - 1)$
- If k is greater, search right sub-array indexed from $(\frac{a}{b}n + 1)$ to $(n - 1)$

Now, write a recursive C-function as,

```
int BipartitionSearch ( int A[ ], int k, int a, int b, int low, int high )
```

which takes as parameters a sorted array A of integers, two indices low and $high$ $(low \leq high)$ in A and the element to be searched for k. The function returns the index, i $(low \leq i \leq high)$, of A if k is found within the indices low and $high$ (both included) of A, otherwise it returns -1. Finally, write the main C-function that –

- reads from user an integer n $(n \leq 10^5)$ and then all n sorted integers (in ascending order) in an array;
- reads another integer k, which is the element being searched;
- reads the bi-partition ratio, that is a and b values $(a < b)$;
- checks whether k resides in the array or not, by using `BiPartitionSearch` function;
- prints the index where the element k resides in the array, otherwise print -1 in case it is not found.

Note: You do not have to sort the array. Just enter the numbers in sorted order directly from the keyboard. You may modify the function definitions as you require.

Example Inputs/Outputs:

Sample-1:

```
Enter Number of Elements: 10
Enter 10 Integer Elements in Ascending-Order (Sorted): 2 4 6 8 10 12 14 16 18 20
Enter the Element to be Searched: 18
Enter the Bi-Partition Ratio (in a/b format): 2/5

The Searched Element ( 18 ) resides in 8-th Index of the Sorted Array!
```

Sample-2:

```
Enter Number of Elements: 10
Enter 10 Integer Elements in Ascending-Order (Sorted): 1 3 5 7 9 11 13 15 17 19
Enter the Element to be Searched: 18
Enter the Bi-Partition Ratio (in a/b format): 1/3

The Searched Element ( 18 ) is NOT Found in the Sorted Array!
```

Submit a single C source file. Do not use global/static variables.

CS19001: Programming and Data Structures Laboratory

Assignment No. 11B (Sorting and Searching)

Date: 08-November-2019

Problem Statement:

You may consider another variant of binary search. *By this time, I guess you may be realizing what are the things that were going in Alice's mind while she was trying to open the magic-box.* Consider the variation of binary search where the sorted array of size n is divided into d parts ($d \leq n$), instead of two parts as conventionally done in binary search (or what you did in the previous assignment). The algorithm is as follows:

- Compare k (the element being searched for) with the $(\frac{1}{d}n)$-th element
- If equal, k found – return
- If k is smaller than $(\frac{1}{d}n)$-th element, search first sub-array indexed from 0 to $(\frac{1}{d}n - 1)$
- Repeat the following for all $i = 2, 3, \ldots, d-1$,
  - If k is greater than $\frac{i-1}{d}n$-th element, compare with $\frac{i}{d}n$-th element
  - If equal, k found – return
  - If k is smaller than $(\frac{i}{d}n)$-th element, search next sub-array indexed from $(\frac{i-1}{d}n + 1)$ to $(\frac{i}{d}n - 1)$
- If k is still greater than $(\frac{d-1}{d}n)$-th element, search last sub-array indexed from $(\frac{d-1}{d}n + 1)$ to $(n-1)$

Write a recursive C-function as,

```
int DarySearch ( int A[ ], int k, int d, int low, int high )
```

which takes as parameters a sorted array A of integers, two indices low and $high$ ($low \leq high$) in A and the element to be searched for k. The function returns the index, k ($low \leq k \leq high$), of A if k is found within the indices low and $high$ (both included) of A, otherwise it returns -1. Finally, write the main C-function that –

- reads from user an integer n ($n \leq 10^5$) and then all n sorted integers (in ascending order) in an array;
- reads another integer k, which is the element being searched;
- reads the expected number of partitions, d where $d \leq n$;
- checks whether k resides in the array or not, by using `DarySearch` function;
- prints the index where the element k resides in the array, otherwise print -1 in case it is not found.

Note: You do not have to sort the array. Just enter the numbers in sorted order directly from the keyboard. You may modify the function definitions as you require.

Example Inputs/Outputs:

Sample-1:

```
Enter Number of Elements: 12
Enter 12 Integer Elements in Ascending-Order (Sorted): 1 2 3 5 7 11 13 17 19 23 29 31
Enter the Element to be Searched: 5
Enter the Expected Number of Partitions: 4

The Searched Element ( 5 ) resides in 3-th Index of the Sorted Array!
```

Sample-2:

```
Enter Number of Elements: 12
Enter 12 Integer Elements in Ascending-Order (Sorted): 4 6 8 9 10 12 14 15 16 18 20 21
Enter the Element to be Searched: 5
Enter the Expected Number of Partitions: 3

The Searched Element ( 5 ) is NOT Found in the Sorted Array!
```

Submit a single C source file. Do not use global/static variables.