CSE100 Algorithm Design and Analysis — Lab #2
Spring semester 2013 — Miguel Á. Carreira-Perpiñán

Searching in sequences

Description In this assignment you are requested to implement different algorithms for searching elements in an array.

1. The first algorithm to implement is the linear search algorithm, whose pseudocode follows

```
Require: $a$: array of elements, of type $T$
Require: $a_s$: element to find, of type $T$
1: $N \leftarrow length(a)$
2: for $i \leftarrow$ 1 to $N$ do
3:    if $a_i = a_s$ then
4:       return $i$
5:    end if
6: end for
7: return -1
```

   Testcase number to use with grade_me: 1

2. The second algorithm to implement is the binary search algorithm. You can assume that the input sequence is already sorted in ascending order.

```
Require: $a$: sorted array of elements, of type $T$
Require: $a_s$: element to find, of type $T$
1: $l \leftarrow 1$
2: $r \leftarrow length(a)$
3: while $l \leq r$ do
4:    $m \leftarrow \lceil (r+l)/2 \rceil$
5:    if $a_m = a_s$ then
6:       return $m$
7:    else if $a_m > a_s$ then
8:       $r \leftarrow m - 1$
9:    else
10:      $l \leftarrow m + 1$
11:   end if
12: end while
13: return -1
```

Testcase number to use with grade_me: 2

Note that you can also implement this algorithm recursively as described in the lecture.

Input structure The input may contain multiple cases. The sequences and the element to search are integers (i.e. you can safely store them into `int` variables). Each case starts with a number which is the number of integers in the sequence. The following number is the element to search. Then the elements of the sequence follow, one per line. A sequence of length 0 indicates that the cases are over, and you should not process it. You can assume that the input is correctly structured (i.e. no data are missing). In the second testcase the sequences are sorted.

Output structure All the searching algorithms must return -1 if a_s is not in the sequence, or its position (i.e. array index) if it is contained. You can assume that all the numbers in the sequence are distinct.

Examples of input and output

Testcase 1: input

```
4
3
4543
63457
43
6896
0
```

Testcase 1: output

```
-1
```

Testcase 2: input

```
4
3
1
2
3
4
0
```

Testcase 2: output

```
2
```

Suggestions If you need to round floating point numbers to integers, you can use the functions `round` or `lround` declared in the header file `math.h`. See the `man` pages for their use. You should also feel free to write your own rounding function(s).
The algorithms presented show pseudo-code. As such, you should think about what the algorithms are trying to achieve before implementing them. Simply following the given pseudo-code will likely result in failure to pass the testcases.

Optional question The different algorithms to implement have different complexities. If you run all of them over the same test cases you would observe significant differences if the sequence to search is long. The function `gettimeofday` can be used to measure time, with a resolution up to the microsecond. Using the man page, figure out how it works and measure the time spent by the different algorithms to solve the testcases contained in the second set. Print the result (i.e. the time) to the file `stderr`.

Your solutions Before leaving the lab, submit a zipped tar archive of your program through the assignments page of UCMCROPS. Please use your UCMNetID as the filename for the zipped tar archive. Be careful since UCMCROPS strictly enforces the assignment deadlines (deadlines will be every lab date at either 4:20pm or 7:20pm depending on your lab session).