CS 11 Exercise 08 - Sorting and Searching

University of the Philippines Diliman

November 4, 2018

Introduction

- The terms "array(s)" and "list(s)" will be used interchangeably.
- For this exercise, you have to write five different computer programs.
- Make sure that your algorithm works given the sample input and output. You must also check if your algorithm can also handle input other than the ones given.
- Please remove any prompt messages (e.g. `Enter number:`) when getting input. Prompt messages will mess with your output, making your solution invalid.
- See the sample input and output to guide you on what and how your program must display output.
- Submit your solutions via HackerRank. The HackerRank site is posted in UVLe.
- Submit your solutions on or before Sunday, November 18 at 11:59pm.

1 Insertion Sort

Implement the insertion sort algorithm shown below and print the desired output as described in the output section below.

1.1 Algorithm

```
for i = 1 to length(A) inclusive do:
    /* select value to be inserted */
    valueToInsert = A[i]
    holePosition = i
    /*locate hole position for the element to be inserted*/
    while holePosition > 0 and A[holePosition-1] > valueToInsert do:
        A[holePosition] = A[holePosition-1]
        holePosition = holePosition -1
    end while
    /* insert the number at hole position */
    A[holePosition] = valueToInsert
end for
```

1.2 Input

Input starts with the number of test cases t, followed by t test cases. Each test case is a line containing integers delimited by a `"|"`. These integers constitute the array (or list) that you need to sort using the Insertion Sort algorithm.

1.2.1 Sample Input

```
5
70|46|727|250|398|904|199|821|536|346|9
1|840|20|60|975|39|906|202|10|291|341|29|121
654|95|775|526|932|438|32|407|
17|832|87|124|95|527|347|403|238|3|807|36|974|91|123
891|2|162|53|412|907|582|802|99
```

1.3 Output

Output all elements swapped on each nth swap where n is divisible by 3. Lastly, output the sorted list.

1.3.1 Sample Output

```
398 727
199 398
536 904
346 904
346 536
9 821
9 398
9 199
[9, 46, 70, 199, 250, 346, 398, 536, 727, 821, 904]
39 975
906 975
202 840
10 840
10 39
291 906
341 906
29 906
29 291
29 39
121 840
121 202
[1, 10, 20, 29, 39, 60, 121, 202, 291, 341, 840, 906, 975]
5 95
32 526
438 526
32 438
407 526
[5, 32, 32, 95, 407, 438, 526, 654]
95 832
347 832
43 537
43 95
238 537
3 537
3 124
3 43
36 832
36 347
36 95
74 832
74 347
74 95
```

```
91 807
91 238
123 832
123 347
[3, 17, 36, 43, 74, 87, 91, 95, 123, 124, 238, 347, 537, 807, 832]
53 891
42 162
97 162
99 891
99 162
123 582
43 802
43 123
43 53
[2, 42, 43, 53, 97, 99, 123, 162, 582, 802, 891]
```

2 Bubble Sort

Implement the Bubble sort algorithm shown below and print the desired output as described in the output section below.

2.1 Algorithm

```
for all elements of list
   if list[i] > list[i+1]
      swap(list[i], list[i+1])
   end if
end for
```

2.2 Input

Input starts with the number of test cases t, followed by t test cases. Each test case is a line containing integers delimited by a `"|"`. These integers constitute the array (or list) that you need to sort using the Bubble Sort algorithm.

```
5
70|46|727|250|398|904|199|821|536|346|9
1|840|20|60|975|39|906|202|10|291|341|29|121
654|95|775|526|932|438|32|407|
17|832|87|124|95|527|347|403|238|3|807|36|974|91|123
891|2|162|53|412|907|582|802|99
```

2.3 Output

Output the state of the list on the nth iteration where n is divisible by 4. Lastly, output the sorted list.

```
[46, 70, 199, 250, 398, 346, 9, 536, 727, 821, 904]
[46, 70, 9, 199, 250, 346, 398, 536, 727, 821, 904]
[9, 46, 70, 199, 250, 346, 398, 536, 727, 821, 904]
[1, 20, 39, 60, 10, 202, 291, 29, 121, 341, 840, 906, 975]
[1, 10, 20, 29, 39, 60, 121, 202, 291, 341, 840, 906, 975]
[1, 10, 20, 29, 39, 60, 121, 202, 291, 341, 840, 906, 975]
[1, 10, 20, 29, 39, 60, 121, 202, 291, 341, 840, 906, 975]
[5, 32, 32, 95, 407, 438, 526, 654]
[5, 32, 32, 95, 407, 438, 526, 654]
[17, 87, 95, 43, 124, 3, 238, 36, 74, 91, 123, 347, 537, 807, 832]
```

```
[17, 3, 43, 36, 74, 87, 91, 95, 123, 124, 238, 347, 537, 807, 832]
[3, 17, 36, 43, 74, 87, 91, 95, 123, 124, 238, 347, 537, 807, 832]
[3, 17, 36, 43, 74, 87, 91, 95, 123, 124, 238, 347, 537, 807, 832]
[2, 42, 53, 97, 99, 123, 43, 162, 582, 802, 891]
[2, 42, 43, 53, 97, 99, 123, 162, 582, 802, 891]
[2, 42, 43, 53, 97, 99, 123, 162, 582, 802, 891]
```

3 Selection Sort

Implement the selection sort algorithm shown below and print the desired output as described in the output section below.

3.1 Algorithm

```
for i = 1 to n - 1
    /* assume current element is minimum*/
    index_min = i
    /* find the minimum element */
    for j = i+1 to n
        if list[j] < list[index_min] then
            index_min = j;
        end if
    end for
    /* swap the minimum element with the current element*/
    if index_min != i  then
        swap list[index_min] and list[i]
    end if
end for
```

3.2 Input

Input starts with the number of test cases t, followed by t test cases. Each test case is a line containing integers delimited by a `"|"`. These integers constitute the array (or list) that you need to sort using the Selection Sort algorithm.

```
70|46|727|250|398|904|199|821|536|346|9
1|840|20|60|975|39|906|202|10|291|341|29|121
654|95|775|526|932|438|32|407|
```

```
17|832|87|124|95|527|347|403|238|3|807|36|974|91|123
891|2|162|53|412|907|582|802|99
```

3.3 Output

Output the state of the list after each swap. Lastly, output the sorted list.

```
[9, 46, 727, 250, 398, 904, 199, 821, 536, 346, 70]
[9, 46, 70, 250, 398, 904, 199, 821, 536, 346, 727]
[9, 46, 70, 199, 398, 904, 250, 821, 536, 346, 727]
[9, 46, 70, 199, 250, 904, 398, 821, 536, 346, 727]
[9, 46, 70, 199, 250, 346, 398, 821, 536, 904, 727]
[9, 46, 70, 199, 250, 346, 398, 536, 821, 904, 727]
[9, 46, 70, 199, 250, 346, 398, 536, 727, 904, 821]
[9, 46, 70, 199, 250, 346, 398, 536, 727, 821, 904]
[9, 46, 70, 199, 250, 346, 398, 536, 727, 821, 904]
[1, 10, 20, 60, 975, 39, 906, 202, 840, 291, 341, 29, 121]
[1, 10, 20, 29, 975, 39, 906, 202, 840, 291, 341, 60, 121]
[1, 10, 20, 29, 39, 975, 906, 202, 840, 291, 341, 60, 121]
[1, 10, 20, 29, 39, 60, 906, 202, 840, 291, 341, 975, 121]
[1, 10, 20, 29, 39, 60, 121, 202, 840, 291, 341, 975, 906]
[1, 10, 20, 29, 39, 60, 121, 202, 291, 840, 341, 975, 906]
[1, 10, 20, 29, 39, 60, 121, 202, 291, 341, 840, 975, 906]
[1, 10, 20, 29, 39, 60, 121, 202, 291, 341, 840, 906, 975]
[1, 10, 20, 29, 39, 60, 121, 202, 291, 341, 840, 906, 975]
[5, 95, 654, 526, 32, 438, 32, 407]
[5, 32, 654, 526, 95, 438, 32, 407]
[5, 32, 32, 526, 95, 438, 654, 407]
[5, 32, 32, 95, 526, 438, 654, 407]
[5, 32, 32, 95, 407, 438, 654, 526]
[5, 32, 32, 95, 407, 438, 526, 654]
[5, 32, 32, 95, 407, 438, 526, 654]
[3, 832, 87, 124, 95, 537, 347, 43, 238, 17, 807, 36, 74, 91, 123]
[3, 17, 87, 124, 95, 537, 347, 43, 238, 832, 807, 36, 74, 91, 123]
[3, 17, 36, 124, 95, 537, 347, 43, 238, 832, 807, 87, 74, 91, 123]
[3, 17, 36, 43, 95, 537, 347, 124, 238, 832, 807, 87, 74, 91, 123]
[3, 17, 36, 43, 74, 537, 347, 124, 238, 832, 807, 87, 95, 91, 123]
[3, 17, 36, 43, 74, 87, 347, 124, 238, 832, 807, 537, 95, 91, 123]
[3, 17, 36, 43, 74, 87, 91, 124, 238, 832, 807, 537, 95, 347, 123]
[3, 17, 36, 43, 74, 87, 91, 95, 238, 832, 807, 537, 124, 347, 123]
[3, 17, 36, 43, 74, 87, 91, 95, 123, 832, 807, 537, 124, 347, 238]
```

```
[3, 17, 36, 43, 74, 87, 91, 95, 123, 124, 807, 537, 832, 347, 238]
[3, 17, 36, 43, 74, 87, 91, 95, 123, 124, 238, 537, 832, 347, 807]
[3, 17, 36, 43, 74, 87, 91, 95, 123, 124, 238, 347, 832, 537, 807]
[3, 17, 36, 43, 74, 87, 91, 95, 123, 124, 238, 347, 537, 832, 807]
[3, 17, 36, 43, 74, 87, 91, 95, 123, 124, 238, 347, 537, 807, 832]
[3, 17, 36, 43, 74, 87, 91, 95, 123, 124, 238, 347, 537, 807, 832]
[2, 891, 162, 53, 42, 97, 582, 802, 99, 123, 43]
[2, 42, 162, 53, 891, 97, 582, 802, 99, 123, 43]
[2, 42, 43, 53, 891, 97, 582, 802, 99, 123, 162]
[2, 42, 43, 53, 97, 891, 582, 802, 99, 123, 162]
[2, 42, 43, 53, 97, 99, 582, 802, 891, 123, 162]
[2, 42, 43, 53, 97, 99, 123, 802, 891, 582, 162]
[2, 42, 43, 53, 97, 99, 123, 162, 891, 582, 802]
[2, 42, 43, 53, 97, 99, 123, 162, 582, 891, 802]
[2, 42, 43, 53, 97, 99, 123, 162, 582, 802, 891]
[2, 42, 43, 53, 97, 99, 123, 162, 582, 802, 891]
```

4 Linear Search

Implement the linear search algorithm shown below and print the desired output as described in the output section below.

4.1 Algorithm

```
for each item in the list
    if item matches value
        return the item's location
    end if
end for
```

4.2 Input

The input contains four parts:

1. A list of words, one word per line.
2. A line containing XXXXXX, which signals the end of the list.
3. One or more words to be searched in the list, one on each line.
4. A line containing XXXXXX, which signals the end of the input.

4.2.1 Sample Input

```
subdued
brainy
adamant
claim
strong
brick
reduce
look
whimsical
uncovered
impolite
strong
advise
towering
wriggle
grab
grandfather
racial
```

```
attach
shop
XXXXXX
parsimonious
heavenly
strong
identify
grab
best
wriggle
chilly
suggestion
brainy
XXXXXX
```

4.3 Output

Output the final status of the search (found or not found) and the total number of comparisons made in the list of words.

4.3.1 Sample Output

```
not found 20
not found 20
found 5
not found 20
found 16
not found 20
found 15
not found 20
not found 20
found 2
```

5 Binary Search

Implement the binary search algorithm shown below and print the desired output as described in the output section below.

5.1 Algorithm

```
A = sorted array
n = size of array
x = value to be searched
Set lowerBound = 1
Set upperBound = n
while x not found
  if upperBound < lowerBound
     EXIT: x does not exist.
  set midPoint = lowerBound + (upperBound-lowerBound) / 2
  if A[midPoint] < x
     set lowerBound = midPoint + 1
  if A[midPoint] > x
     set upperBound = midPoint - 1
  if A[midPoint] = x
     EXIT: x found at location midPoint
end while
```

5.2 Input

The input contains four parts:

1. A list of words, one word per line.
2. A line containing XXXXXX, which signals the end of the list.
3. One or more words to be searched in the list, one on each line.
4. A line containing XXXXXX, which signals the end of the input.

5.2.1 Sample Input

```
subdued
brainy
adamant
claim
strong
brick
reduce
```

```
look
whimsical
uncovered
impolite
strong
advise
towering
wriggle
grab
grandfather
racial
attach
shop
XXXXXX
parsimonious
heavenly
strong
identify
grab
best
wriggle
chilly
suggestion
brainy
XXXXXX
```

5.3 Output

Output in a file the matches made and final status of the search (found or not found) in listofwords.txt

5.3.1 Sample Output

```
look parsimonious
strong parsimonious
reduce parsimonious
racial parsimonious
not found
look heavenly
brick heavenly
grab heavenly
grandfather heavenly
```

```
impolite heavenly
not found
look strong
strong strong
found
look identify
brick identify
grab identify
grandfather identify
impolite identify
not found
look grab
brick grab
grab grab
found
look best
brick best
advise best
attach best
brainy best
not found
look wriggle
strong wriggle
uncovered wriggle
whimsical wriggle
wriggle wriggle
found
look chilly
brick chilly
grab chilly
claim chilly
not found
look suggestion
strong suggestion
uncovered suggestion
subdued suggestion
towering suggestion
not found
look brainy
brick brainy
advise brainy
attach brainy
brainy brainy
found
```