|  |
| --- |
| **Declaration of Original Work**  By submitting the answer script of this assignment to the subject lecturer through Moodle Centralized Group, you hereby declare that the work in the answer sheet is completely your own work. No part of the answer sheet is taken from other people’s work without giving them credit. All references have been clearly cited.  You understand that an infringement of this declaration leaves you subject to disciplinary actions such as mark deduction, disqualification or even expulsion by the College.  If necessary, students may be invited to provide more information on their submission. |

*(Please refer to the relevant section(s) on plagiarism of the Student Handbook.)*

**Instructions to Students:**

1. Please refer to assignment specification for the submission method
2. Show all your work clearly and neatly. Marks will be deducted for untidy work.

**Answer ALL questions.**

**Answer for Question 1**

1. Code captionText

   Description automatically generated
2. Code captionGraphical user interface, application

   Description automatically generated
3. Code captionText

   Description automatically generated

**Answer for Question 2**

1. Code captionText

   Description automatically generated



1. Code captionText

   Description automatically generated
2. Code captionText

   Description automatically generated

Program execution result:

Text

Description automatically generated

More test cases:

Text

Description automatically generated

Text

Description automatically generatedText

Description automatically generated

Printing information before and after sort: Text

Description automatically generated

Result:

Text

Description automatically generated

**Answer for Question 3**

1. In the aspect of efficiency, bubble sort has the average time complexity of O(n2), but quick sort has the average time complexity of O(n log n), which is faster than bubble sort.

The quick sort applies the idea of pivot, partition, and recursion, it based on partitioning of array of data into smaller arrays. But in bubble sort, it partially sorts some continuous elements in each iteration, we usually called them as bubbles.

The quick sort requires extra coding effort on implementation, because of the partitioning. But bubble sort does not, it only required 2 for-loop to directly sort the data array.

1. Quick sort is based on recursion

Partitioning pseudo code:

function partitioning( List, starting, ending){

x = List[starting]

i = starting

for j = starting + 1 to ending{

if List[j] < x then{

i = i + 1

swap(List[i], List[j])

}

}

swap(List[i], List[starting])

return i

}

Sorting pseudo code:

function quicksort(List, starting, ending){

if starting >= ending

return

else{

pivot = partitioning(List, starting, ending)

quicksort(List, left, pivot - 1)

quicksort(List, pivot + 1, ending)

}

}

Demonstration of the quick sort algorithm:

unsorted = [1,9,2,8,7,3]

quicksort(unsorted, 0, 5)

1st pass:

pivot = partitioning(unsorted, 0, 5)

list == [1, 9, 2, 8, 7, 3]

pivot = 0

quicksort(unsorted, 0, -1) 🡺 return

quicksort(unsorted, 1, 5) 🡺 1st inner quicksort() (right hand side):

pivot = partitioning(unsorted, 1, 5)

list == [1, 3, 2, 8, 7, 9]

pivot = 5

quicksort(unsorted, 1, 4) 🡺 2nd inner quicksort() (left hand side):

pivot = partitioning(unsorted, 1, 4)

list == [1, 2, 3, 8, 7, 9]

pivot = 2

quicksort(unsorted, 1, 1) 🡺 return

quicksort(unsorted, 3, 4) 🡺 3rd inner quicksort() (right hand side):

pivot = partitioning(unsorted, 3, 4)

list == [1, 2, 3, 7, 8, 9]

pivot = 4

quicksort(unsorted, 3, 3) 🡺 return

quicksort(unsorted, 5, 4) 🡺 return

**(as above 2 inner quicksort() is returned, the list is sorted)**

quicksort(unsorted, 6, 5) 🡺 return

**Answer for Question 4**

Value a to f: 20, 1, 19, 95, 56, 60

1. Initial state:

20, 1, 19, 95, 56, 60

1st pass:

1, 20, 19, 95, 56, 60 🡸 swapped 20, 1

1, 19, 20, 95, 56, 60 🡸 swapped 20, 19

1, 19, 20, 95, 56, 60 🡸 20 < 95, no swap

1, 19, 20, 56, 95, 60 🡸 swapped 95, 56

1, 19, 20, 56, 60, 95 🡸 swapped 95, 60

2nd pass, no swap:

1, 19, 20, 56, 60, 95 🡸 1<19, no swap

1, 19, 20, 56, 60, 95 🡸 19<20, no swap

1, 19, 20, 56, 60, 95 🡸 20<56, no swap

1, 19, 20, 56, 60, 95 🡸 56<60, no swap

1, 19, 20, 56, 60, 95 🡸 60<95, no swap

The sorting is early terminated in the case.

Reason:

In the 2nd pass, there is no swap between, 1 and 19, 19 and 20, 20 and 56, 56 and 60. Therefore, all elements in the remaining sub-list are sorted, which also means the whole list is already sorted (because the largest number is already at the last), so early termination exists.

1. Sort 20, 1, 19, 95, 56, 60

Start with 20

20

insert 1 🡺 1,20

\_\_,20 🡺 1,20

insert 19 🡺 1, 19, 20

1, \_\_,20 🡺 1, 19, 20

insert 95 🡺 1, 19, 20, 95

1, 19, 20, \_\_ 🡺 1, 19, 20, 95

insert 56 🡺 1, 19, 20, 56, 95

1, 19, 20, \_\_, 95 🡺 1, 19, 20, 56, 95

insert 60 🡺 1, 19, 20, 56, 60, 95

1, 19, 20, 56, \_\_, 95 🡺 1, 19, 20, 56, 60, 95