04 - Graded Participation

2222 unread replies.2222 replies.

Your participation score for this module is based on posting an answer to **one** of the practice questions below to this discussion board.  Review the information on the [Graded Participation Information](https://ccsf.instructure.com/courses/50381/pages/graded-participation-information) for more details and the [Discussion Board Tips](https://ccsf.instructure.com/courses/50381/pages/discussion-board-tips) for details on using Canvas discussion boards.  For coding questions, make sure to read the [posting code to Canvas](https://ccsf.instructure.com/courses/50381/pages/posting-code-to-canvas) guide to make your answer look nice on the discussion board!

Note: we have a large amount of questions this week!  Pick your favorite one, and answer it in your reply to this discussion board.  Make sure to read the posts other students make, both for your question, and for other questions.  Post replies to other students' posts if you have questions or comments!

**Question 1**

Using Big-O notation, indicate the Order of the following for the worst case.  State any assumptions you make.

* 1. After arriving at a party with n people present, you shake hands with everyone there.
  2. Everyone in a room with n people shakes hands with everyone else in the room.
  3. You climb a flight of n stairs.
  4. You slide down the banister next to a staircase with n stairs.
  5. After entering an elevator in a building with n floors, you press a button to choose a floor.

**Question 2**

Using Big-O notation, indicate the Order of the following for the worst case.  State any assumptions you make.

* 1. Compute the sum of a list of n integers.
  2. Compute the sum of every other integer in a list of n integers.
  3. Display an element in an arbitrary position in a linked list of n integers.
  4. Display an element in an arbitrary position in an array-based list of n integers.

**Question 3**

Consider the code below.  In Big-O notation, what is the order of it?  Explain your answer.

for (int pass = 1; pass <= n; pass++)  
{  
  for (int i = 0; i < n; i++)  
  {  
    for (int count = 1; count < 10; count++)  
    {  
      task T // some task which takes t time units  
    }  
  }  
}

**Question 4**

1. Implement the function insertionSort().  Copying it from the slide is fine.
2. Modify the function to sort in descending order instead of ascending order.
3. Add a counter to the function to count how many array accesses it makes.   An array access is defined as any time you read to or write to the array you are sorting.
4. In big-O notation, what is the efficiency of insertionSort()?
5. Write a simple main() function which calls your function with at least three arrays of random values of various sizes.  How many array accesses are made for each array?

Paste your modified insertionSort(), your main(), and answers to questions 4 and 5 as your reply.

**Question 5**

Trace the merge sort algorithm as it sorts the following array into ascending order:

20 80 40 25 60 30

You can do this with pencil and paper, and then take a picture, and post it as a reply.  If you would like to do it electronically, that's fine too.

**Question 6**

Trace the quick sort algorithm as it does one partition step on the following array.  Assume your sort uses median-of-three pivot selection with positions 0, 2, and 5 (values 20, 40, 30 in the below).

20 80 40 25 60 30

You can do this with pencil and paper, and then take a picture, and post it as a reply.  If you would like to do it electronically, that's fine too.

**Question 7**

Find an array of six elements which makes the bubble sort demonstrate its worst case behavior.  Assume you are sorting into ascending order.

**Question 8**

Suppose you comment out the call to merge() from the merge sort algorithm, so you wind up with the below mystery function instead.  What will this new algorithm do?

mergeSort(theArray: ItemArray, first: integer, last: integer)  
{  
  if (first<last)  
  {  
    mid = (first+last) / 2  
      
    // recursively sort first half of array  
    mergeSort(theArray, first, mid)  
  
    // recursively sort second half of array  
    mergeSort(theArray, mid+1, last)  
  
    // merge sorted halves (first thru mid, and mid+1 thru last)  
    // merge(theArray, first, mid, last)  
  }  
}