Submit Files - Assignment 9

Top of Form

|  |  |
| --- | --- |
| [[https://online.pcc.edu/d2l/img/0/Shared.Main.actHide.png?v=10.6.1.2687-197](javascript://)Hide Folder Information](javascript://) | |
| Folder |  |
| Assignment 9 | |
| Instructions |  |
| Overview --------  For this assignment you will create a program to read in a sequence of numbers and then sort them.  As with the last assignment, this will take some time. There are lots of details to figure out. Give yourself plenty of time to work on it.  Details -------  This program will start by asking the user to enter a file name. Then the program will open the file for reading. The program will read all of the data from the file and store the values in an array. There will be 100 lines in the file and each line will contain a single integer. Once all of the integers are stored in the array, the program will use one of the n-squared sorts discussed in chapter 10 (bubble, selection, insertion) to sort all of the data. Then the program will output the values in ascending order by printing one number per line.  Example input -------------  You can use the following as example input to your program for testing. Copy the following text into a file and use that for testing.  ------- 6476 1882 7027 8906 4487 1957 6398 6410 6630 5006 6722 311 8833 9727 9946 9825 596 4828 9530 7578 3192 2407 4403 5515 6541 4307 8015 5916 9061 6207 7819 1000 1710 5963 6635 294 3457 5768 5440 3252 4476 8191 6247 5730 155 5502 3697 5499 7350 9898 9903 1118 3822 8105 560 3625 9076 167 5612 5157 2754 6848 6369 5144 7095 8165 2796 725 5626 3191 8183 3305 6262 7599 3513 902 4198 1597 8216 8245 3012 3689 5387 2125 2088 9651 2469 8064 3175 2446 9036 8656 7176 3413 9832 1432 4189 2532 2554 8138 -------  The result from this should be:  155 167 294 311 560 596 725 902 1000 1118 1432 1597 1710 1882 1957  2088 2125 2407 2446 2469 2532 2554 2754 2796 3012 3175 3191 3192 3252 3305 3413 3457 3513 3625 3689 3697 3822 4189 4198 4307 4403 4476 4487 4828 5006 5144 5157 5387 5440 5499 5502 5515 5612 5626 5730 5768 5916 5963 6207 6247 6262 6369 6398 6410 6476 6541 6630 6635 6722 6848 7027 7095 7176 7350 7578 7599 7819 8015 8064 8105 8138 8165 8183 8191 8216 8245 8656 8833 8906 9036 9061 9076 9530 9651 9727 9825 9832 9898 9903 9946  Turning in the assignment -------------------------  When turning in the assignment you should turn in a zipfile of the folder containing your project. An easy way to get to this is to right-click on the file tab (in the editor pane) and select "Open Containing Folder". Then navigate up to the parent directory that contains the \*.sln file. This is the folder that you need to zip up and turn in.  By default this will contain a bunch of other files that Visual Studio uses during compilation. This will make the zip file very large. One easy way to reduce this size is to remove the \*.sdf file (it is in the same folder as the \*.sln file). This is a generated file and can be safely removed before creating the zip file.  NOTES: | |

Overview  
--------  
  
The final chapter for this term covers "Applications of Arrays".  This  
is an introduction to topics that you will learn in detail in the  
following terms.  Because it will be handled in more detail later we  
will only be highlighting it here.  
  
Sequential Searching  
--------------------  
  
The basic idea of a sequential search is to start at the beginning of  
an array and look at each element one at a time.  For example, if I  
have an array of integers I can do the following to find 17.  
  
bool found = false;  
for (int i=0;i<SIZE;i++) {  
  if (arr[i] == 17) {  
    found = true;  
  }  
}  
  
Notice that I am using a for loop to look at each element one at a  
time.  That is, I'm looking at them in sequence.  
  
Sequential searches are simple and effective but they can be slow.  If  
you have a list of 1,000,000 elements it will take, on average,  
500,000 lookups to find the element.  
  
A Faster Search  
---------------  
  
A faster way of searching is the Binary Search.  A binary search  
requires that you start with an ordered lists.  Then you look at the  
middle element of the list.  If the target value is less than the  
middle element, continue searching in the first half.  Otherwise  
continue in the second half.  Repeat this until you find the element  
or run oiut of elements.  
  
A binary search is a lot faster than a sequential search.  In fact,  
for 1,000,000 elements, the average number of lookups is 20.  
  
Sorting  
-------  
  
In Computer Science the term "sorting" refers to putting things in  
order.  That is, sorting a list of numbers means putting them in  
ascending or descending order.  Just as with searching, there are a  
number of algorithms available for sorting.  Some are slow and some  
are fast.  In this chapter we cover the category of searches known as  
"n-squared" searches.   
  
Bubble Sort  
-----------  
  
The bubble sort is based on a simple principle: compare two adjacent  
elements.  The they are out of order with respect to each other then  
swap them.  Continue doing this until everything is in order.  
  
The code looks something like this:  
  
for (int i=0; i<len; i++) {  
  for (int j=0; j<len; j++) {  
    if (arr[i] > arr[j]) {  
      swap(arr[i],arr[j]);  
    }  
  }  
}  
  
Selection Sort  
--------------  
  
The basic idea behind the selection sort is to find the smallest  
element of the list and move it to the front of the list.  Then shrink  
the list by one and repeat.  Continue doing this until you read the  
end of the list.  
  
The code looks something like this:  
  
for (int i=0; i<len; i++) {  
  int smallest = arr[i];  
  int smallestIdx = i;  
  
  for (int j=i; j<len; j++) {  
    if (arr[j] < smallest) {  
      smallest = arr[j];  
      smallestIdx = j;  
    }  
  }  
  
  swap(arr[i],arr[smallestIdx]);  
}  
  
It is a bit more complicated than the bubble sort but it is more  
efficient.  The bubble sort will move elements multiple times.  The  
selection sort will move elements around a lot fewer times.  
  
Insertion Sort  
--------------  
  
The insertion sort is slightly more complicated.  It tries to reduce  
the number of comparisons even more.  Since this one is more  
complicated it is best to look in the book and see the example there.  
  
The n-squared Sorts  
-------------------  
  
All of these sorting algorithms are n-squared.  That means that for a  
list of n elements, it will take approximately n\*n operations to sort  
the list.  For example, a list of 1,000 elements will require about  
1,000,000 operations for sorting.  This is not good performance so in  
general people avoid using the n-squared sorts.  There are faster  
sorts that you will learn about later.    
  
However, it's not a total waste to learn about the n-squared sorts.  
These are very good "training wheels" for learning about the higher  
level algorithms.Bottom of Form