CSC 323 Algorithm Design and Analysis
Spring 2017
Instructor: Dr. Natarajan Meghanathan

Term Project

There are 4 teams of students (as chosen by the students themselves). The instructor created a pool of eight projects of similar type and comparable level of difficulty.
As shown in the video **https://youtu.be/dVvAab31qi8**, each team is randomly assigned two projects. A team can choose one of the two projects assigned to them and do as the term project.

Due dates:

March 28, 2017 (in-class @ 1 PM): A progress report due for each team on their chosen project.

April 18 and April 20, 2017 (Term Project Presentation, in-class starting from 1 PM): Each team will present their project for 25-30 minutes.

The instructor will use a random number program to decide which team presents on what day.
The presentation schedule will be announced during the week of March 28, 2017.

April 20, 2017 (Hardcopy of the final project report @ 1 PM): Each team will submit a printed copy of their final comprehensive project report.

The items to be included in the progress report, final report and the team presentation are listed as part of the individual project descriptions.

Teams and the Assigned Projects

Team 2: Jason Bruno, Jordan Hubbard, Justin McGuffee
Project # 5: Determining the Smallest Integer for which a Monotonically Decreasing Function goes below a Certain Value
Project # 1: Counting the Number of Inversions in an Array
****

Team 3: Bria McCutcheon, Kayla Johnson, Jaylen Boykin
Project # 4: Determining the Common Integers among Several Arrays
Project # 7: Determining the K-Closest Elements to a Given Value in an Array
****

Team 4: Alexander Arrington, Daniel Epps, Michale Wilson
Project # 8: Rearranging the Elements in an Array
Project # 2: Counting the Number of Integer Pairs in an Array that Add to a Target Value
****

Team 1: Darren McGee, Kayshaunna Williams, Elbert Buchanan
Project # 6: Determining the Unique Integers in an Array
Project # 3: Determining the Smallest and Second Smallest Integers in an Array using Iterative and Recursive Approaches
****

Project # 1: Counting the Number of Inversions in an Array

Your **objective** in this project is to develop a Θ(nlogn) algorithm to determine the inversion count of an array.

The number of inversions in an array is a measure of how far is the array from being sorted. For an array A[0, ..., n-1], we say there is an inversion involving two distinct indices i and j (where $0 \le i \le n-1$ and $0 \le j \le n-1$) if $i < j$, but $A[i] > A[j]$. The number of such pairs of indices i and j (wherein $i < j$ and $A[i] > A[j]$) in an array A is called the inversion count of the array.

For example: the following array has 4 inversions

0	1	2	3	4
10	1	4	10	5

Index Pairs i, j	Values A[i], A[j]	Inversion Yes or No
0, 1	10, 1	Yes
0, 2	10, 4	Yes
0, 3	10, 10	No
0, 4	10, 5	Yes
1, 2	1, 4	No
1, 3	1, 10	No
1, 4	1, 5	No
2, 3	4, 10	No
2, 4	4, 5	No
3, 4	10, 5	Yes

Note that if for any two distinct indices i and j, if A[i] = A[j], then there is NO inversion involving the indices i and j. In the example shown here, there is no inversion involving indices 0 and 3, as both A[0] and A[3] are 10.

We say there is an inversion for two distinct indices i and j only if $i < j$ and $A[i] > A[j]$.

Tasks: Your specific tasks are as follows:
(1) Provide a detailed description of the working of the algorithm along with an example.
(2) Provide a pseudo code for the algorithm.
(3) Analyze the theoretical run-time complexity of the algorithm and show that it is Θ(nlogn).
(4) Analyze the theoretical space complexity of the algorithm.
(5) Discuss the correctness of your algorithm
(6) Implement the algorithm
(7) Run simulations of your algorithm for array sizes of 10, 100, 1000 and 10000. You could fill the contents of the arrays with random integers ranging from 1 to 500. For each array size, run the simulations for 50 runs and average the results for the inversion and the execution time of the algorithm. Plot the results for the array size (n) vs. {average execution time and the theoretical run-time complexity that is supposed to be nlogn} and also plot the results for the array size vs. average inversion count.

What to submit in the progress report (hard copy)
Items 1 through 5.

What to submit in the final report (hard copy)
Items 1 through 5; For item 6 - provide the entire code along with clear documentation of each method; For item 7 - provide screenshots of the execution for each array size and print the average inversion count as well as the average execution time; show the Excel plots of the results; interpret the results.

What to present (team presentation)
Item 1 (an overview of the working along with an example); Items 2 through 5

For Items 6 and 7 - Show sample execution in class for two array sizes, and the printing of the inversion count and execution time. Also discuss the results for the array size vs. {average execution time and the theoretical run-time complexity} via the Excel plots.

Project # 2: Counting the Number of Integer Pairs in an Array that Add to a Target Value

Your **objective** in this project is to develop a Θ(nlogn) algorithm to determine the number of integer pairs in an array (of size n) that add to a target value (t).

You would create an array of 'n' random integers whose values range from 1 to 500. You also have a set T of target values, T = {500, ..., 1000}. Your task is to find the number of integer pairs (say, denoted [x, y]) in the array such that they add up to a particular target value t ∈ T (i.e., x + y = t). Note that x and y could be either distinct or the same values as long as they occupy two different locations (indices) in the array.

In the following example, I show the contents of an array of n = 7 random integers whose values range from 1 to 10. The set T of target values is T = {10, ..., 20}. The number of [x, y] pairs whose sum is equal to a target value in T is shown alongside.

0	1	2	3	4	5	6
10	3	5	10	7	4	9

Target Value, t	# {x, y} Pairs	{x, y} Pairs
10	1	{3, 7}
11	1	{7, 4}
12	2	{3, 9}; {5, 7}
13	3	{10, 3}; {3, 10}; {4, 9}
14	3	{10, 4}; {5, 9}; {10, 4}
15	2	{10, 5}; {5, 10}
16	1	{7, 9}
17	2	{10, 7}; {10, 7}
18	0	-
19	2	{10, 9}; {10, 9}
20	1	{10, 10}

For clarity, I have distinguished the two 10s in this example by representing the 10 at index 3 with an underscore (10).

Tasks: Your specific tasks are as follows:
(1) Provide a detailed description of the working of the algorithm along with an example.
(2) Provide a pseudo code for the algorithm.
(3) Analyze the theoretical run-time complexity of the algorithm and show that it is Θ(nlogn).
(4) Analyze the theoretical space complexity of the algorithm.
(5) Discuss the correctness of your algorithm
(6) Implement the algorithm
(7) Run simulations of your algorithm for array sizes of 100, 1000 and 10000. You could fill the contents of all of these arrays with random integers ranging from 1 to 500. For each array size, run the simulations for 50 runs and average the results for the number of {x, y} pairs with respect each of the target sum values in the set T = {500, ..., 1000}. Plot the results as follows:

(a) The integer target values in the set T = {500, ..., 1000} vs. the average number of {x, y} pairs that add to each of these target values
(b) The array size vs. {average execution time and the theoretical run-time complexity that is supposed to be nlogn}

What to submit in the progress report (hard copy): Items 1 through 5.

What to submit in the final report (hard copy)
Items 1 through 5; For item 6 - provide the entire code along with clear documentation of each method; For item 7 - provide screenshots of the execution for each array size and the printing of the distribution of

the number of {x, y} pairs that add to a sample/particular target value in the set T = {500, ..., 1000}. Also discuss the results for the array size vs. {average execution time and the theoretical run-time complexity}; plot the results in Excel; interpret the results.

What to present (team presentation)
Item 1 (an overview of the working along with an example); Items 2 through 5
For Items 6 and 7 - Show sample execution in class for two array sizes, and the printing of the distribution of the number of {x, y} pairs that add to each of the target values in the set T = {500, ..., 1000}. Also discuss the results for the array size vs. {average execution time and the theoretical run-time complexity} via the Excel plots.

Project # 3: Determining the Smallest and Second Smallest Integers in an Array using Iterative and Recursive Approaches

Given an array A of 'n' integers, your **objective** in this project is to determine the smallest and second smallest integers using an iterative $\Theta(n)$ algorithm as well as a recursive $\Theta(n)$ algorithm. The second smallest integer has to be strictly larger than the smallest integer (but not larger than any other integer). In other words, if there is more than one occurrence of the smallest integer in an array, the second smallest integer cannot be the same as the smallest integer in the array. For example, in the array A = {3, 1, 2, 4, 1, 5, 6}; the smallest integer is 1 and the second smallest integer is 2.

Tasks: Your specific tasks are as follows:
(1) Provide a detailed description of an iterative $\Theta(n)$ algorithm to determine the smallest and second smallest integers in an array along with an example to illustrate the working of the algorithm on an array of 8-10 integers.
(2) Provide a detailed description of a recursive $\Theta(n)$ algorithm to determine the smallest and second smallest integers in an array along with an example to illustrate the working of the algorithm on an array of 8-10 integers.
(3) A pseudo code for the iterative algorithm.
(4) A pseudo code for the recursive algorithm.
(5) Analyze the theoretical time-complexity of the iterative algorithm and show that it is $\Theta(n)$ for an array of 'n' integers.
(6) Analyze the theoretical time-complexity of the recursive algorithm and show that is $\Theta(n)$ for an array of 'n' integers.
(7) Analyze the space complexity of the iterative and recursive algorithms.
(8) Implement the iterative and recursive algorithms.
(9) Run simulations of the iterative and recursive algorithms for the following values of the array size n = 10, 100, 1000, 10000 and 100000. You could fill the contents of all of these arrays with random integers ranging from 1 to 500. For each array size, run 50 simulations and average the results for the execution time. Plot the results for the array size vs. {average execution time for the iterative algorithm, average execution time for the recursive algorithm}.

What to submit in the progress report (hard copy): Items 1 through 7.

What to submit in the final report (hard copy)
Items 1 through 7; For item 8 - provide the entire code along with clear documentation of each method; For item 9 - provide screenshots of the execution for each array size, the printing of the execution time and the Excel plots of the results; interpret the results.

What to present (team presentation)
Items 1 and 2 (an overview of the working along with an example); Items 3 through 7
For Items 8 and 9 - Show sample execution in class for two array sizes, and the printing of the smallest and second smallest integers of the array. Also discuss the results for the array size vs. {average execution time for the iterative algorithm, average execution time for the recursive algorithm} via the Excel plots.

Project # 4: Determining the Common Integers among Several Arrays

In this project, your **objective** will be to design and implement a $\Theta(mn)$ algorithm for determining the common integers among 'm' arrays, each of size 'n'. Treat an integer occurring at a particular index as a unique entity; this way, if an integer occurs more than once in an array, each incidence is treated as a unique entity. If the same integer occurs in all the 'm' arrays more than once, you need to determine the minimum number of occurrences of the integer across all the 'm' arrays and consider those occurrences as the common integers.

For example, consider the three arrays (m = 3) of n = 10 integers each as shown below. I indicate each incidence of an integer with a unique suffix (this way you can distinguish an integer occurring more than once within an array); the common integers (present in all the three of them) are: $\{12_1, 12_2, 78_1, 10_1\}$.

Array 1: $\{25_1, 12_1, 85_1, 14_1, 12_2, 63_1, 35_1, 22_1, 10_1, 78_1\}$
Array 2: $\{12_1, 55_1, 78_1, 63_1, 55_2, 12_2, 47_1, 10_1, 62_1, 78_2\}$
Array 3: $\{15_1, 42_1, 78_1, 12_1, 12_2, 10_1, 78_2, 22_1, 10_2, 10_3\}$

We have two common occurrences of integer 12 (denoted 12_1 and 12_2) along one with one common occurrence of integers 78 and 10 (denoted 78_1 and 10_1).

Note that integer 10 is present thrice in Array 3, but only once in Arrays 1 and 2. Hence, we conclude integer 10 to be present only once across all the three arrays. The same applies for integer 78. Note that 63 is present once in both Arrays 1 and 2, but is not present even once in Array 3; so, integer 63 is not considered to be common across all the three arrays.

Tasks: Your specific tasks are as follows:
(1) Provide a detailed description of the working of the algorithm along with an example.
(2) Provide a pseudo code for the algorithm.
(3) Analyze the theoretical run-time complexity of the algorithm and show that it is $\Theta(mn)$ for 'm' arrays, each of size 'n'.
(4) Analyze the theoretical space complexity of the algorithm.
(5) Discuss the correctness of your algorithm
(6) Implement the algorithm
(7) Run simulations of your algorithm for the number of arrays (m) to be 4, 6 and 8. In each case: the number of integers in each array (n) is 200. You could fill the contents of all of these arrays with random integers ranging from 1 to 500. For each value of m: conduct the simulations for 50 runs and average the results for the execution time. Plot the results for the number of arrays vs. average execution time.

What to submit in the progress report (hard copy): Items 1 through 5.

What to submit in the final report (hard copy)
Items 1 through 5; For item 6 - provide the entire code along with clear documentation of each method; For item 7 - provide screenshots of the execution for each value of the number of arrays and the printing of the common integers across all the arrays as well as submit the Excel plots of the results for the number of arrays vs. average execution time. Interpret the results of the Excel plots.

What to present (team presentation)
Item 1 (an overview of the working along with an example); Items 2 through 5
For Items 6 and 7 - Show sample execution in class for all the three values for the number of arrays and the printing of the common integers in all the arrays as well as the execution time. Also discuss the results for the array size vs. {average execution time} via the Excel plots.

Project # 5: Determining the Smallest Integer for which a Monotonically Decreasing Function goes below a Certain Value

In this project, you are given a monotonically decreasing function f(n), where n is a positive integer (n > 0). Your **objective** is to develop a Θ(logn) algorithm that would determine the smallest value of n for which f(n) would be less than a target value 't'. For all demo and validation purposes of your code, you could assume f(n) to be $2/n^2$ and the target value t to be 0.001.

In the following chart, we show the values of $f(n) = 2/n^2$ for different values of n and we identify the minimum value of n for which f(n) would be less than target values for t = 0.1, 0.05 and 0.01.

n	**f(*n*) = $2/n^2$**	***n***	**f(*n*) = $2/n^2$**
1	2	9	0.0247
2	0.5	10	0.02
3	0.222	11	0.0165
4	0.125	12	0.0139
5	0.08	13	0.0118
6	0.0556	14	0.0102
7	0.0408	15	0.0089
8	0.0313	16	0.0078

Target value, t	Min. Value of 'n'
0.1	5
0.05	7
0.01	15

Tasks: Your specific tasks are as follows:
(1) Provide a detailed description of the working of the algorithm along with an example (assume t = 0.001).
(2) Provide a pseudo code for the algorithm and identify the basic operation.
(3) Analyze the theoretical run-time complexity of the algorithm with respect to the basic operation identified in (2) and show that it is Θ(logn) where n is the minimum value for which f(n) goes below a target value.
(4) Analyze the theoretical space complexity of the algorithm.
(5) Discuss the correctness of your algorithm
(6) Implement the algorithm
(7) Run simulations of your algorithm with $f(n) = 2/n^2$ and find the minimum value of n for which f(n) goes below the target values of t = 0.1, 0.01, 0.001, 0.0001 and 0.00001. Plot the results for the target value vs. {the number of times the basic operation is executed; the theoretical run-time complexity that is supposed to be logn where n is the minimum value for which f(n) goes below the target value of t}.

What to submit in the progress report (hard copy): Items 1 through 5.

What to submit in the final report (hard copy)
Items 1 through 5; For item 6 - provide the entire code along with clear documentation of each method; For item 7 - provide screenshots of the execution for each of the five target values and printing the minimum value of 'n' and the number of times the basic operation is executed as well as submit the Excel plots of the results for the target value vs. {the number of times the basic operation is executed; the theoretical run-time complexity that is supposed to be logn where n is the minimum value for which f(n) goes below the target value of t }. Interpret the results of the Excel plots.

What to present (team presentation)
Item 1 (an overview of the working along with an example); Items 2 through 5
For Items 6 and 7 - Show sample execution in class for all the five target values and the printing of the minimum value of 'n' and the number of times the basic operation is executed. Also discuss the results for the target value vs. the number of times the basic operation is executed via the Excel plots.

Project # 6: Determining the Unique Integers in an Array

In this project, your **objective** is to design and implement a Θ(nlogn) algorithm as well as a Θ(n) algorithm to determine the unique integers in an array of size 'n'.

An integer appearing more than once in an input array is included only once in the output array. Also, the integers should be included in the output array in the order they appeared in the input array. In other words, if two distinct integers A[i] and A[j] appeared at indices i and j respectively (such that i < j) for the first time in the input array A, then the two integers should appear at indices i' and j' respectively such that i' < j' in the output array. For example, if the input array is: {4, 2, 6, 4, 3, 5, 5}, then the output array should be {4, 2, 6, 3, 5}. Note that 4 and 5 appeared for the first time at indices 0 and 5 respectively in the input array and they appear appropriately at indices 0 and 4 in the output array. Likewise, integers 6 and 3 appeared respectively at indices 2 and 4 in the input array and they appear appropriately at indices 2 and 3 in the output array.

Tasks: Your specific tasks are as follows:
(1) Provide a detailed description of the Θ(nlogn) algorithm along with an example to illustrate the working of the algorithm on an array of 8-10 integers.
(2) Provide a detailed description of the Θ(n) algorithm along with an example to illustrate the working of the algorithm on an array of 8-10 integers.
(3) A pseudo code for the Θ(nlogn) algorithm.
(4) A pseudo code for the Θ(n) algorithm.
(5) Analyze the theoretical time-complexity of the algorithm of (1) and (3) and show that it is Θ(nlogn) for an array of 'n' integers.
(6) Analyze the theoretical time-complexity of the algorithm of (2) and (4) and show that it is Θ(n) for an array of 'n' integers.
(7) Analyze the space complexity of the Θ(nlogn) and Θ(n) algorithms.
(8) Implement the Θ(nlogn) and Θ(n) algorithms.
(9) Run simulations of the Θ(nlogn) and Θ(n) algorithms for the following values of the array size n = 10, 100, 1000 and 10000. You could fill the contents of all of these arrays with random integers ranging from 1 to 500. For each array size, run 50 simulations and average the results for the execution time. Plot the results for the array size vs. {average execution time for the Θ(nlogn) algorithm, average execution time for the Θ(n) algorithm}.

What to submit in the progress report (hard copy): Items 1 through 7.

What to submit in the final report (hard copy)
Items 1 through 7; For item 8 - provide the entire code along with clear documentation of each method; For item 9 - provide screenshots of the execution for each array size and the printing of the output array with unique integers. Also discuss the results for the array size vs. {average execution time for the Θ(nlogn) algorithm, average execution time for the Θ(n) algorithm} via the Excel plots; interpret the results.

What to present (team presentation)
Items 1 and 2 (an overview of the working along with an example); Items 3 through 7
For Items 8 and 9 - Show sample execution in class for array sizes of n = 10 and 100 and the printing of the output array with unique integers. Also discuss the results for the array size vs. {average execution time for the Θ(nlogn) algorithm, average execution time for the Θ(n) algorithm} via the Excel plots.

Project # 7: Determining the K-Closest Elements to a Given Value in an Array

In this project, your objective will be to design and implement a Θ(nlogn) algorithm to determine the 'K' closest elements to a target value X in an array of 'n' integers.

For example, consider an array A = {4, 1, 9, 7, 8, 3, 10, 2}. The three (K = 3) closest elements to integer 6 (X = 6) in this array are: {4, 7, 8}.

If the target value X is in the array itself, then X is not to be considered as one of the K-Closest elements. For example, consider an array A = {4, 1, 9, 7, 8, 3, 10, 2}. The three (K = 3) closest elements to a target integer 7 (X = 7) in this array are: {4, 8, 9}. Note that {8, 9, 10} is also a valid answer in this case. As long as you print one of the valid answers, it is sufficient.

In the above example (with {4, 8, 9} as the 3-closest values to '7'), the average of the 3-closest values is also 7, and the absolute difference between the target value of 7 and the average of the 3-closest values to 7 is 0. The 4-closest values to 7 are: {8, 9, 4, 10} and their average is 7.75; the absolute difference between the target value of 7 and the average of the 4-closest values to 7 is 0.75.

Tasks: Your specific tasks are as follows:
(1) Provide a detailed description of the working of the algorithm along with an example for an array of size 10 integers.
(2) Provide a pseudo code for the algorithm.
(3) Analyze the theoretical run-time complexity of the algorithm and show that it is Θ(nlogn) where n is the number of integers in the array.
(4) Analyze the theoretical space complexity of the algorithm.
(5) Discuss the correctness of your algorithm
(6) Implement the algorithm
(7) Run simulations of your algorithm with array sizes of n = 100, 1000 and 10000. You could fill the arrays with integers randomly chosen from the range [1, ..., 500]. For each array size, vary the K values from 3 to 10. For each array size and K, conduct simulations for 50 runs (for each run, create an array for the given size and choose a random value for the target X in the range 1, ..., 500) and determine the average of the K-closest values to X as well as determine the overall average of the difference between the target X values and the average of the K-closest values to X (across the 50 simulation runs and the target X values chosen for a given array size and K).
Also, determine the average execution time for each value of the array size and the different values of K.

(a) For each array size, n: plot the distribution of {K} vs. {overall average of the {difference between the target X values and the average of the K-closest values to the target X values} }
(b) For each array size, n: plot the distribution of {K} vs. average execution time.

What to submit in the progress report (hard copy): Items 1 through 5.

What to submit in the final report (hard copy)
Items 1 through 5; For item 6 - provide the entire code along with clear documentation of each method; For item 7 - provide screenshots of the execution of the algorithm for all the three values of the array sizes (n = 100, 1000 and 10000) and K values of 3 to 10: show the printing of the overall average of the {difference between the target X values and the average of the K-closest values to the target X values} as well as the average execution time of the algorithm. Plot the results (a) and (b) of Item 7 in Excel and interpret them.

What to present (team presentation)
Item 1 (an overview of the working along with an example); Items 2 through 5

For Items 6 and 7 - Show sample execution in class for array sizes of 10 and 100 and the printing of the overall average of the difference values for K values of 3 and 10 as well as the average execution time of the algorithm. Also discuss the results using the Excel plots (a) and (b) as listed under Item 7.

Project # 8: Rearranging the Elements in an Array

Your **objective** in this project is as follows: Given a randomly generated array of negative and positive integers, design and implement two independent algorithms to rearrange the elements of the array such that it has all the negative integers followed by all the positive integers. Note that '0' is to be considered a positive integer. Your two algorithms should be as follows:
(Alg-1): The space-complexity of the algorithm should be Θ(1): i.e., the algorithm cannot use additional space proportional to the size of the input array; but, a few temporary variables could be used. In other words, all modifications (i.e., rearrangement of the elements) should be done on the input array itself. Though there is no restriction on the time-complexity of the algorithm, it should be as low as possible.
(Alg-2): The time-complexity of the algorithm should be Θ(n). Though there is no restriction on the space-complexity of the algorithm, it should be as low as possible.

An example to illustrate the problem is given below.

0	1	2	3	4	5	6	7	8	9	10
50	78	-21	0	-69	0	22	-44	25	-10	17

Input Array

0	1	2	3	4	5	6	7	8	9	10
-10	-44	-21	-69	0	22	0	25	78	17	50

Rearranged Array

Tasks: Your specific tasks are as follows:
(1) Provide a detailed description of Alg-1: the Θ(1) space-complexity algorithm along with an example to illustrate the working of the algorithm on an array of 8-10 integers (mix of positive and negative).
(2) Provide a detailed description of Alg-2: the Θ(n) time-complexity algorithm along with an example to illustrate the working of the algorithm on an array of 8-10 integers (mix of positive and negative).
(3) A pseudo code for Alg-1: the Θ(1) space-complexity algorithm.
(4) A pseudo code for Alg-2: the Θ(n) time-complexity algorithm.
(5) Analyze the theoretical time-complexity of Alg-1 of (1) and (3) for an array of 'n' integers.
(6) Analyze the theoretical time-complexity of Alg-2 of (2) and (4) for an array of 'n' integers.
(7) Analyze the space complexity of Alg-1 and Alg-2.
(8) Implement Alg-1 and Alg-2.
(9) Run simulations of Alg-1 and Alg-2 for the following values of the array size n = 10, 100, 1000 and 10000. You could fill the contents of all of these arrays with random integers ranging from -500 to 500. For each array size, run 50 simulations and average the results for the execution time. Plot the results for the array size vs. {average execution time for algorithm Alg-1, average execution time for algorithm Alg-2}.

What to submit in the progress report (hard copy): Items 1 through 7.

What to submit in the final report (hard copy)
Items 1 through 7; For item 8 - provide the entire code along with clear documentation of each method; For item 9 - provide screenshots of the execution for each array size and print the output rearranged array with a sequence of negative integers followed by positive integers. Also, discuss the results for the array size vs. {average execution time for Alg-1, average execution time for Alg-2} via the Excel plots; interpret the results.

What to present (team presentation)
Items 1 and 2 (an overview of the working along with an example); Items 3 through 7
For Items 8 and 9 - Show sample execution in class for array sizes of n = 10 and 100 and the printing of the output rearranged array with a sequence of negative integers followed by positive integers. Also, discuss the results for the array size vs. {average execution time for Alg-1, average execution time for Alg-2} via the Excel plots.