

CSCI 241 Data Structures Project 1

Objectives

The goals of this project include:

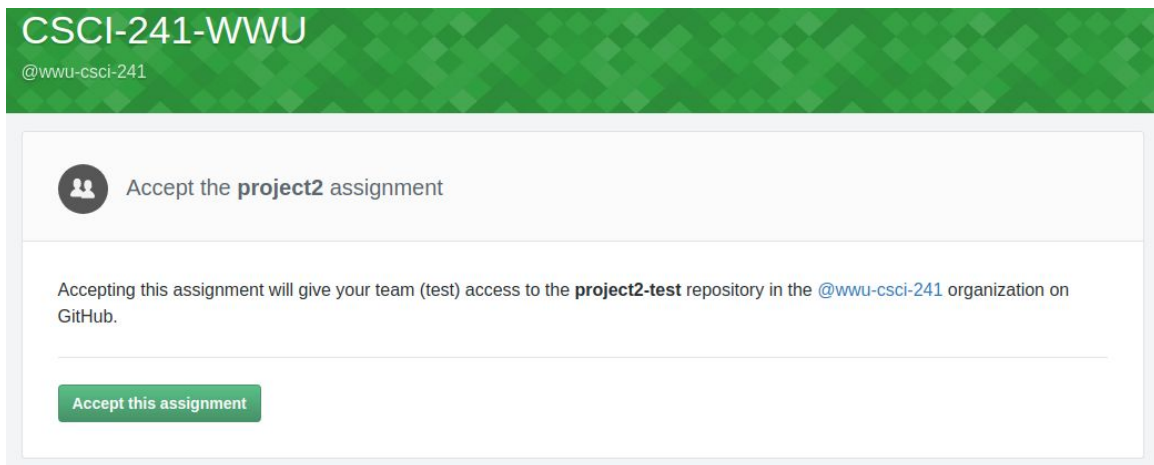
1. Use the implemented sorting algorithms from lab2 and lab3
2. Compare their running time

Part I Set Up a Team

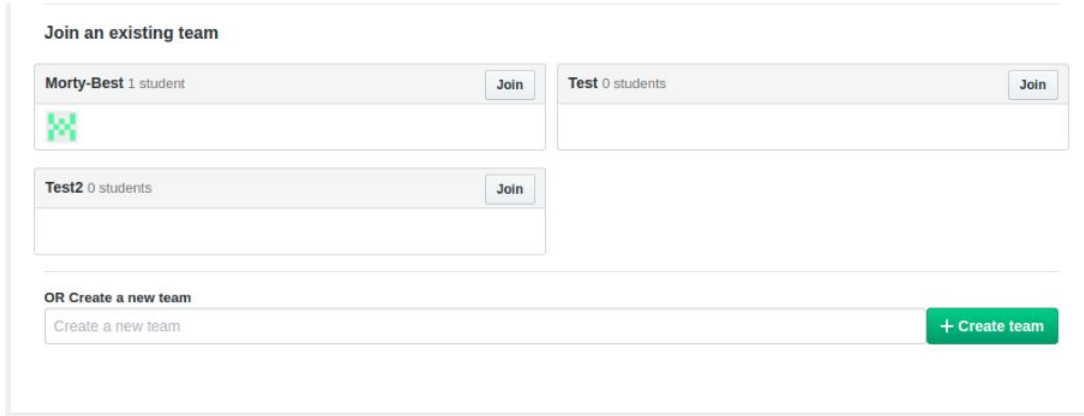
In this project, you will be working with other students as a team. A team can have maximum 3 members, and one member has to take the role as team leader. Team leader will

- initiate meetings
- make sure every member takes a fair share of tasks
- lead the team to beat the deadline.

If you are a team leader, you will need to click this link (https://classroom.github.com/g/K4rKir_w) to initialize a repository for your team. You will be asked to give a name to your team. In the following example, you will see that the team is named “Morty-Best”.



If you are a team member, you will need to confirm that your team leader has done the above step. After that, you will need to click this link (https://classroom.github.com/g/K4rKir_w) and choose your team among many teams, and click Join. In the following example, we will join the team named “Morty-Best”.



Note that the above steps are based on the assumption that you have joined wwu-csci-145 GitHub organization. If you never did this before, you may want to do it now.

Part II Implementation of Sorting Algorithm

Implement (or take advantage of your code from lab2 and lab3 if you are certain that there are no errors) four sorting algorithms under a class named “SortingPack” with the following **static methods**:

- Insertion sort: Shall be named as “insertionSort”; accepts an integer array as the only parameter
`public static int[] insertionSort(int[] unsortedArr)`
- Quick sort: Shall be named as “quickSort”; accepts an integer array as the only parameter
`public static int[] quickSort(int[] unsortedArr)`
- Merge sort: Shall be named as “mergeSort”; accepts an integer array as the only parameter
`public static int[] mergeSort(int[] unsortedArr)`
- Heap sorting: Shall be named as “heapSort”; accepts an integer array as the only parameter. Given that this method will need to use the data structure of a heap, you will need to implement heap as a separate helper class (or reuse your code from lab3). The naming of the helper class and its methods is at your discretion.

`public static int[] heapSort(int[] unsortedArr)`

The above class names and method names should be the same as specified above.

Time your sorting algorithms and compare them

Create a testing class named “SortingTest”. Under this class, there needs to be two more static method besides the main method:

1. *randomArray*: This method return a random unsorted integer array. The bound of any generated random integer should be [0, 10000] (inclusive). This method will accept an integer parameter *n*, which specifies the length of the returned array:

```
int[] arr = randomArray(5); // possibly [3, 1, 2, 7, 10]
int[] brr = randomArray(3); // possibly [840, 0, 25]
```

2. *sortingTime*: This method returns the time needed to sort an array. This method takes two parameters

`String sortingAlgorithm, int[] arr`

`sortingAlgorithm` specifies the sorting algorithm to use. The four sorting algorithms you implemented in the last section should all be recognized by their method names (`insertionSort`, `quickSort`, `mergeSort`, `heapSort`). `arr` specifies the array to sort.

For the main method under `SortingTest`:

Time how long it takes to sort random unsorted integer arrays at different length for every implemented sorting method. The length of the arrays should start with 100, increment by 10, and end at 1000. **To reduce the influence of randomness, time counting needs to be conducted for 3 times at a given length *n* for each method, and the average should be calculated and treated as the execution time for the given *n*.** An example could be:

```
// you do not want to code as the following:
// round 1
arr = randomArray(100);
int time_insertionSort_1 = sortingTime(arr, insertionSort);
int time_quickSort_1 = sortingTime(arr, quickSort);
int time_mergeSort_1 = sortingTime(arr, mergeSort);
int time_heapSort_1 = sortingTime(arr, heapSort);
// round 2 and 3
// ...
// calculation
int time_insertionSort = (time_insertionSort_1 + time_insertionSort_2
+ time_insertionSort_3);
```

The timing data need to be saved in a CSV file. You will need to learn how to use File IO of Java for this task. Here is a link to get your started:

<https://www.seas.upenn.edu/~cis1xx/resources/java/fileIO/introToFileIO.html>. The file needs to be named as “time.csv”.

length	insertionSort	quickSort	mergeSort	heapSort
100	39.12431	21.7446	40.1192	30.18278
110	54.12492	56.3111	45.1231	37.3282
120	69.12553	90.8776	50.127	44.47362
130	84.12614	125.4441	55.1309	51.61904
140	99.12675	160.0106	60.1348	58.76446

To make the comparison among sorting algorithms fair for any given length n , you always want to make sure the randomly generated unsorted array stays the same for all sorting algorithms per round. One way of doing this is to deep copy the generated array.

Update / Upload Files for Your Project

Write a description for this project and discuss about your results in the README file. **Please be as brief as possible.**

Make a plot using Excel or any other softwares you like based on the data from your CSV file. Only one plot is in need: x-axis would be the array length, and y-axis would be the time; four curve lines in different colors need be drawn. Generate a jpg named “plot.jpg”. You will include this plot in your README file to strengthen your discussion.

It is beneficial if you can learn some Markdown language: <https://www.markdowntutorial.com/> .

The time.csv and plot.jpg should be included at the 1st level of your repo:

```

|— build.gradle
|— gradle
|— gradlew
|— libs
|— gradlew.bat
|— settings.gradle
|— README.md
|— src
|   |— main
|   |   |— java
|   |   |   |— yourJavaFiles
|   |— test
|   |   |— java
|   |   |   |— yourJavaTestingFiles
|— time.csv
|— plot.jpg

```

Submission and Grading

You should open your browser and double check if such changes are reflected in your Github repo. Programs that don't compile will risk getting a zero. Late assignments are not accepted. Please use comments to explain your program or any sections that are possibly difficult to understand. **10% of the total lab points would be deducted if no comments can be found in your java files. 10% of the total point would be deducted if there is no README file.**