CS146: Spring 2023
Homework 2: Divide and Conquer
Due Friday, February 17, at 11:59PM
100 points

For this assignment you will implement two divide and conquer algorithms: merge sort and the maximum subarray (we will cover this algorithm the week of February 6).. For merge sort, you will just implement the algorithm on an array of ints. I have provided the pseudo code for both algorithms below. You will need to handle cases of all sizes, not just powers of 2.

Merge Sort

```
mergeSort(arr,start,end)
  if start>=end
      return
  mid = midpoint of arr
  mergeSort(arr,start,mid)
  mergeSort(arr,mid+1,end)

  //merge the two parts
  merge(arr,start,mid,end)

merge(arr,start,mid,end)
      leftSize = mid - start + 1;
      rightSize = end - mid;

      left[0..leftSize]
      right[0..rightSize]

      for leftIndex=0 to leftSize
            left[leftIndex] = arr[start+leftIndex]
      for rightIndex=0 to rightSize
            right[rightIndex] = arr[mid+rightIndex-1]
      left[leftSize] = MAX
      right[rightSize] = MAX

      leftIndex = 0
      rightIndex = 0

      for mergeIndex=start to end
            if left[leftIndex]≤right[rightIndex]
                  arr[mergeIndex] = left[leftIndex]
                  leftIndex = leftIndex+1
            else
                  arr[mergeIndex] = right[rightIndex]
                  rightIndex = rightIndex+1
```

Maximum Subarray

```
findMaxSubarry(arr,low,high)
      if high==low
            return (low,high,arr[low])
      else
            mid = mid point of arr
            (l-low,l-high,l-sum) = findMaxSubarray(arr,low,mid)
            (r-low,r-high,r-sum) = findMaxSubarray(arr,mid+1,high)
            (c-low,c-high,c-sum) = findMaxCrossing(arr,low,mid,high)
            if l-sum ≥ r-sum and l-sum ≥ c-sum
                  return (l-low,l-high,l-sum)
            else if r-sum ≥l-sum and r-sum ≥ c-sum
                  return (r-low,r-high,r-sum)
            else
                  return (c-low,c-high,c-sum)

findMaxCrossing(arr,low,mid,high)
      l-sum = MIN
      sum = 0
      for i=mid downto low
            sum = sum arr[i]
            if sum > l-sum
                  l-sum = sum
                  max-left = i
      r-sum = MIN
      sum = 0
      for j=mid+1 to high
            sum = sum + arr[j]
            if sum > r-sum
                  r-sum = sum
                  max-right = j
      return (max-left,max-right,l-sum+r-sum);
```

Getting Started

You are free to use whatever IDE or programming environment you prefer, but most instructions will assume that you are using the Eclipse IDE. Create a workspace for homework 2 and in that workspace create a project called hw2_proj, containing a package called divideandconquer. Copy the starter files (Triple.java,LinkedList.java, MergeSorterTest.java, and MaxSubFinderTest.java) into the package. MergeSorterTest.java and MaxSubFinderTest are JUnit 4 test files. You need to add JUnit 4 to your build path, see the JUnit Setup section for instructions. If you are unfamiliar with JUnit tests, please see Lab 5 from CS46B that I added to the Canvas Resources.

Triple.java

This file contains a generic Triple class. You will use this to return the starting point, ending point, and sum for the MaxSubFinder class. We use classes like this when we want to have a container that holds different types. **Do not modify this class.**

LinkedList.java

This file contains a simple doubly linked list class. You will use this class when implementing the getMaxSubList method in the MaxSubFinder class. Please read through the class as it provides some helpful methods including a middle method that returns the Node in the middle of the list. **Do not modify this class.**

Implement Merge Sort (40 points)

Create a class called MergeSorter in the `divideandconquer` package. This class will implement merge sort on an array of ints. Implement the following method with the exact signature below. You will need to create private helper methods that do most of the work.

```
public static void mergeSort(int[] arr)
```

This method sorts the int[] arr using the merge sort algorithm described in the pseudocode above.

Commenting your Code (10 points)

It is important to understand how your entire program works. For the method below (and any helper methods you create) write a comment indicating the purpose of the section of code. For methods that are short, a comment above the method will suffice, but for longer methods such as when you implement the merge method, you should comment on different parts in more detail (like how the different components of the merge method are working).

Testing Your MergeSorter (5 points)

I have given you a very limited set of JUnit tests to help determine if your implementation is correct. You need to add at least 5 more JUnit tests for the mergeSort method that test typical and edge cases. You get ½ point for creating the test case and ½ point for passing the test case. You are welcome to create more than five JUnit tests, but as long as you create and pass five JUnit tests, you will receive full credit for this part of the assignment.

Maximum Subarray using Divide and Conquer (60 points)

Create a class called MaxSubFinder in the `divideandconquer` package. This class will implement the maximum subarray problem on both an array and a linked list using the pseudocode described above. Implement the following methods with the exact signatures below. You will need to create private helper methods to do most of the work. **You can assume that the array and list are not empty.**

`public static Triple<Integer,Integer,Integer> getMaxSubarray(int[] arr)`

This method returns a triple that represents the maximum subarray. The first element of the triple is the index in arr where the maximum subarray starts, the middle element of the triple is where the index in the arr where maximum subarray ends, and the last element of the triple is the maximum subarray sum.

`public static Triple<Node,Node,Integer> getMaxSubList(LinkedList list)`

This method returns a triple that represents the maximum sub list. The first element of the triple is the list node where the maximum sublist starts, the middle element of the triple is the list node where the maximum sublist ends, and the last element of the triple is the maximum sublist sum.

Commenting your Code (10 points)

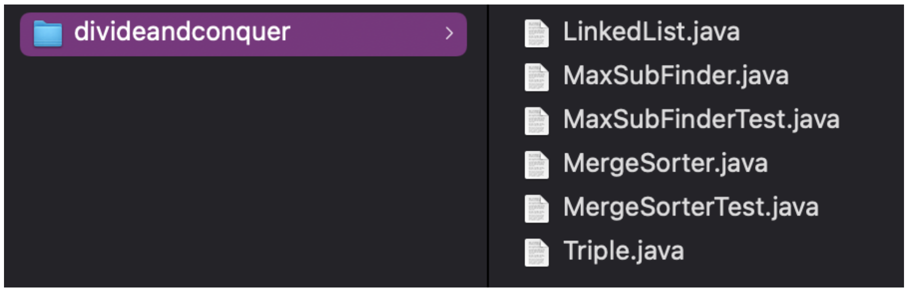
It is important to understand how your entire program works. For the methods below (and any helper methods you create) write a comment indicating the purpose of the section of code. For methods that are short, a comment above the method will suffice, but for longer sections of code such as when you need to complete the maximum crossing you should provide details within the method.

Testing Your MaxSubFinder (5 points)

I have given you a very limited set of JUnit tests to help determine if your implementation is correct. You need to add at least 5 more JUnit tests for the getMaxSubArray method and the getMaxSubList method that test both typical and edge cases. You get ½ point for creating the test case and ½ point for passing the test case. You are welcome to create more than five JUnit tests, but as long as you create and pass five JUnit tests, you will receive full credit for this part of the assignment.

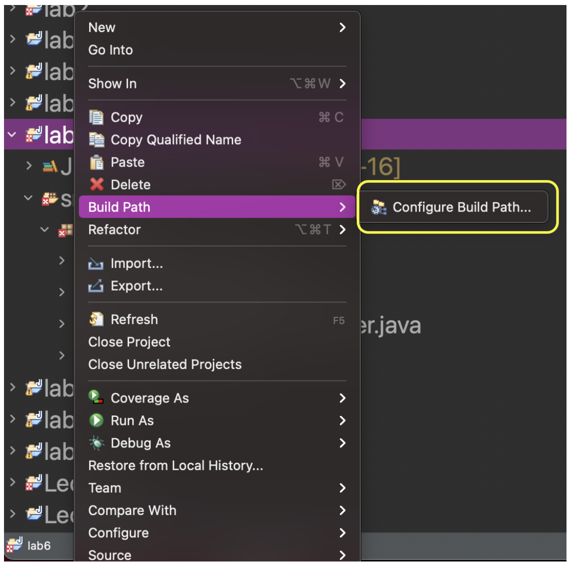
Submitting

Export your project and upload. You can export your project as either a jar or a zip file. Make sure that the file you submit contains all your .java files in the package. If you upload a file that is missing any of the java source files you will get an automatic deduction of 10 points. Basically when I unjar or unzip your file I should get a structure like this…

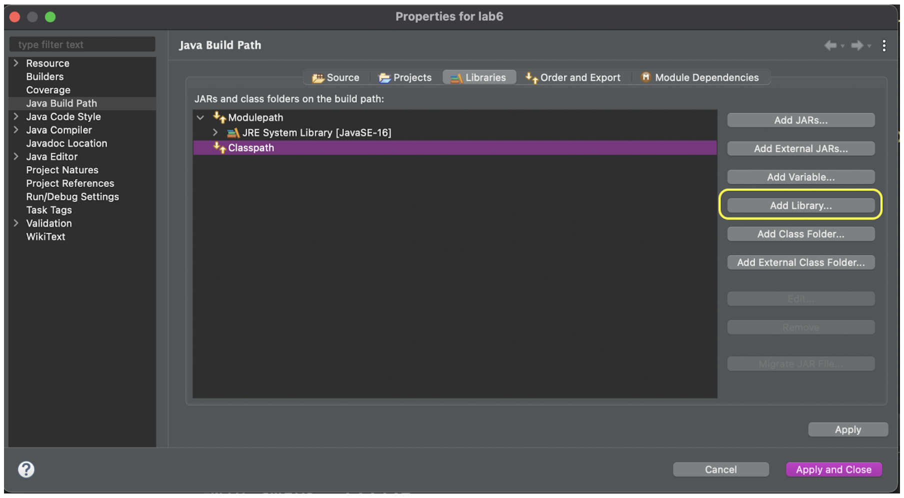


JUnit Setup

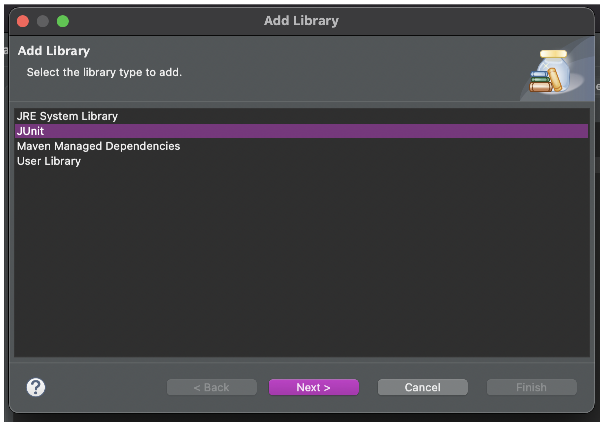
These instructions are for the Eclipse IDE. While you are welcome to use any IDE or programming environment, I will only provide instructions for Eclipse. To add the JUnit library to your build path, right-click on your homework project, look for Build Path and click on the Configure Build Path option.



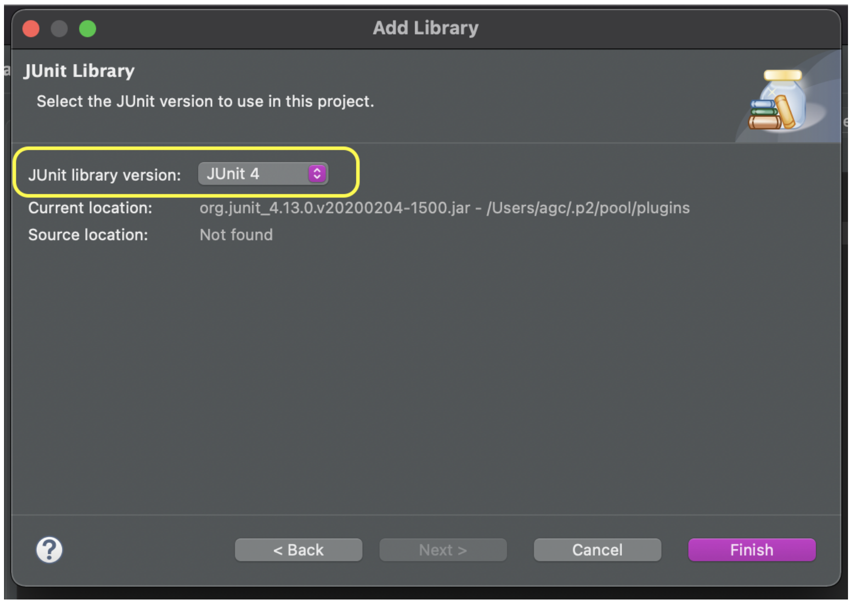
Highlight Classpath and click Add Library



Select JUnit and click Next

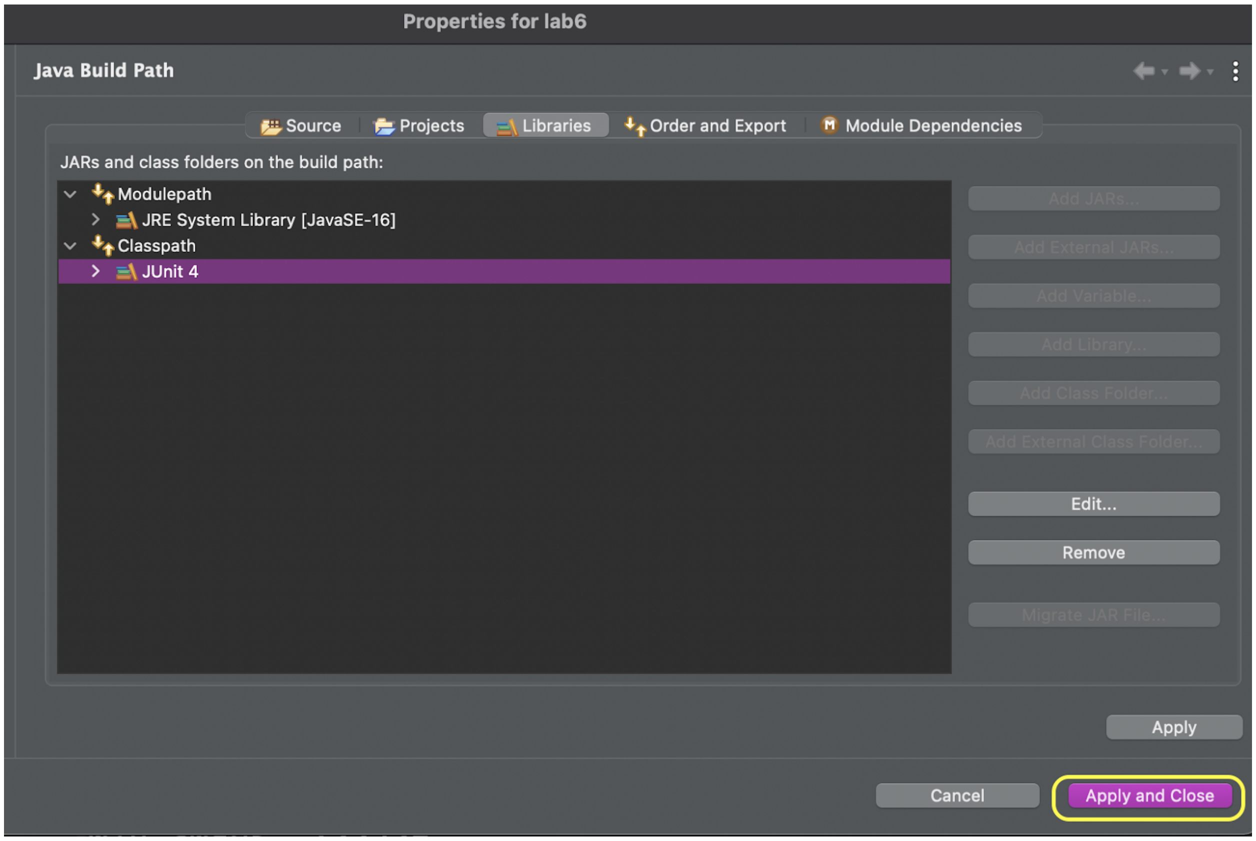


Make sure you select JUnit 4. Then click Finish.



JUnit library version: JUnit 4
Current location: org.junit_4.13.0.v20200204-1500.jar - /Users/agc/.p2/pool/plugins
Source location: Not found

Your build path should look something like this. Click Apply and Close.


Properties for lab6

Java Build Path

JARs and class folders on the build path:

- Modulepath
  - JRE System Library [JavaSE-16]
- Classpath
  - JUnit 4

That should have gotten rid of any compilation errors you had when you originally copied over the files.