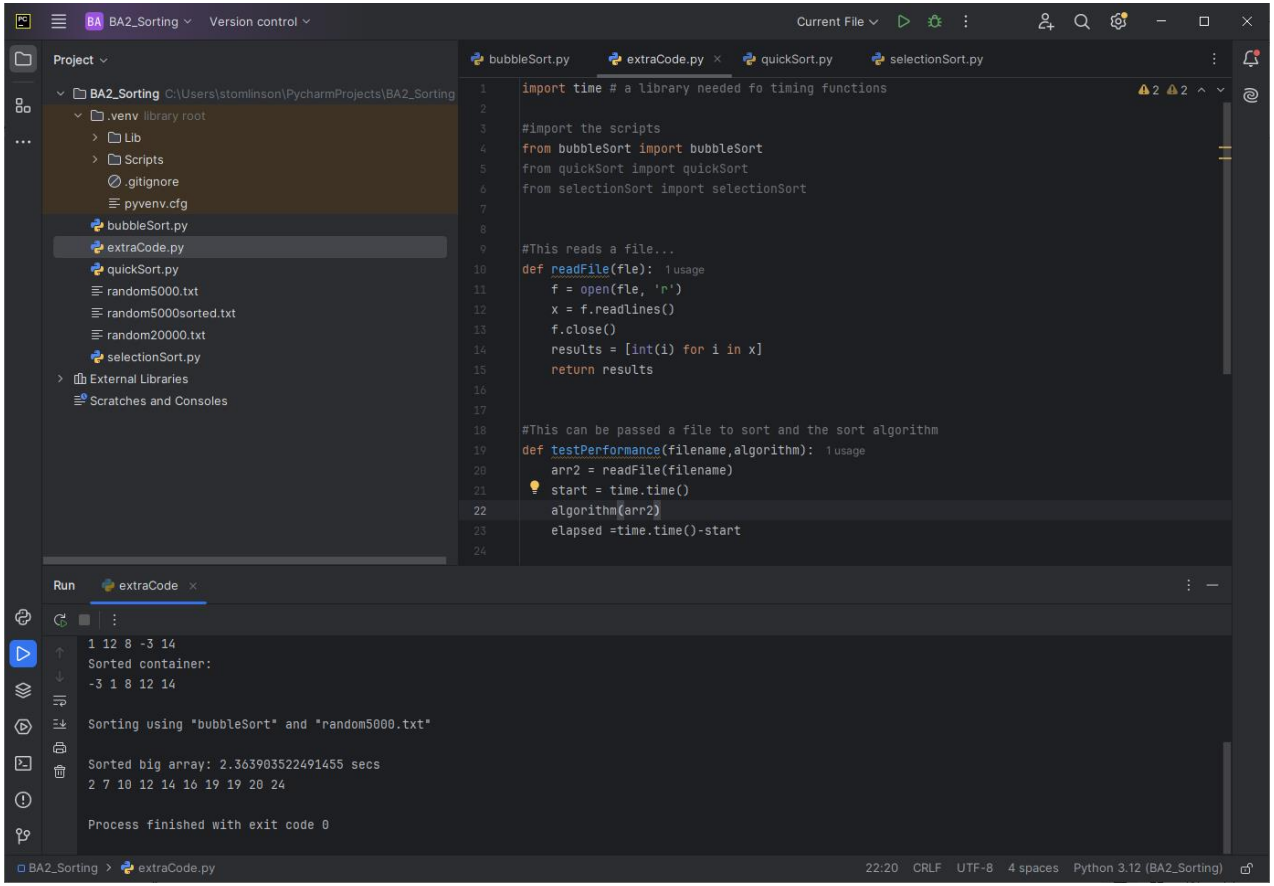
Implementing Algorithms

Bioinformatics Algorithms Week2

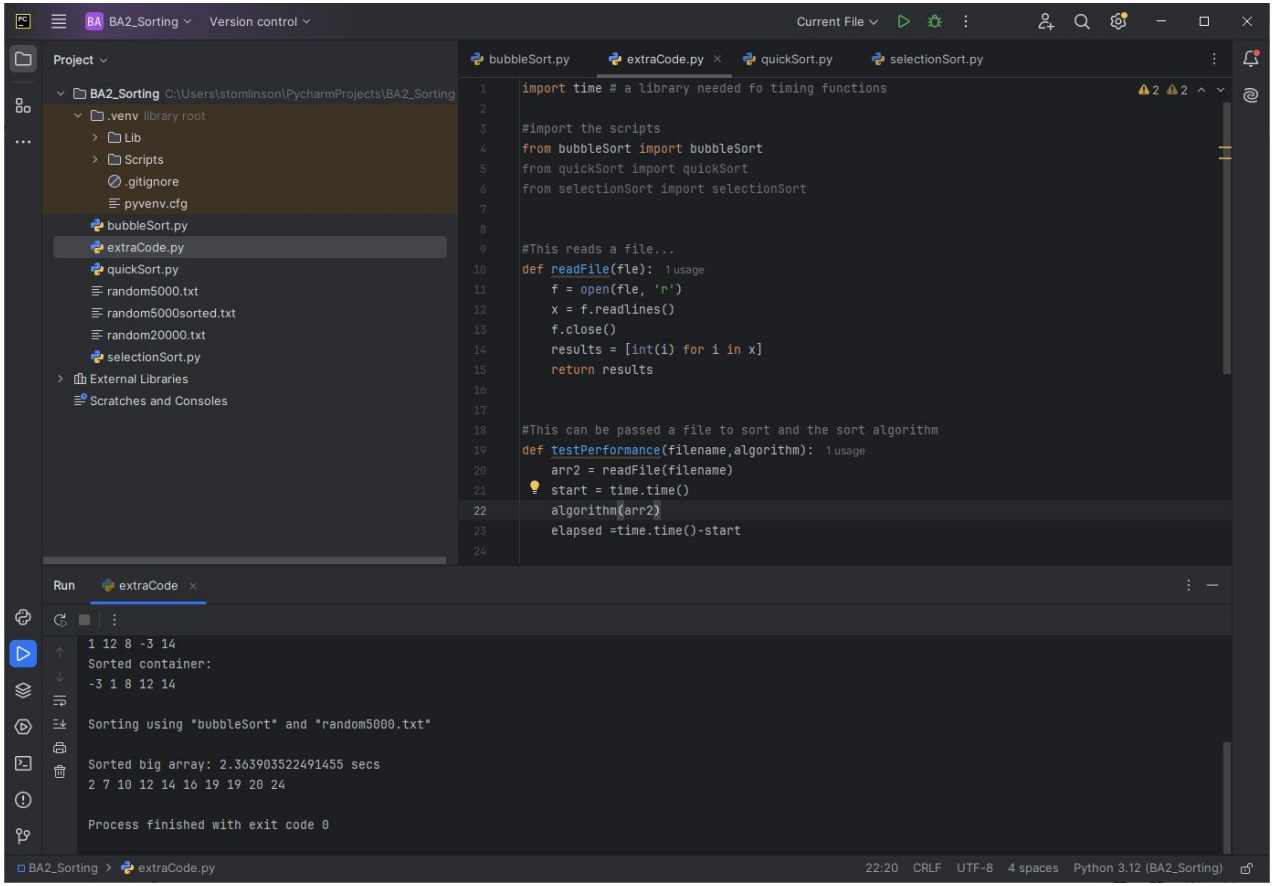
Simon Tomlinson

Use PyCharm



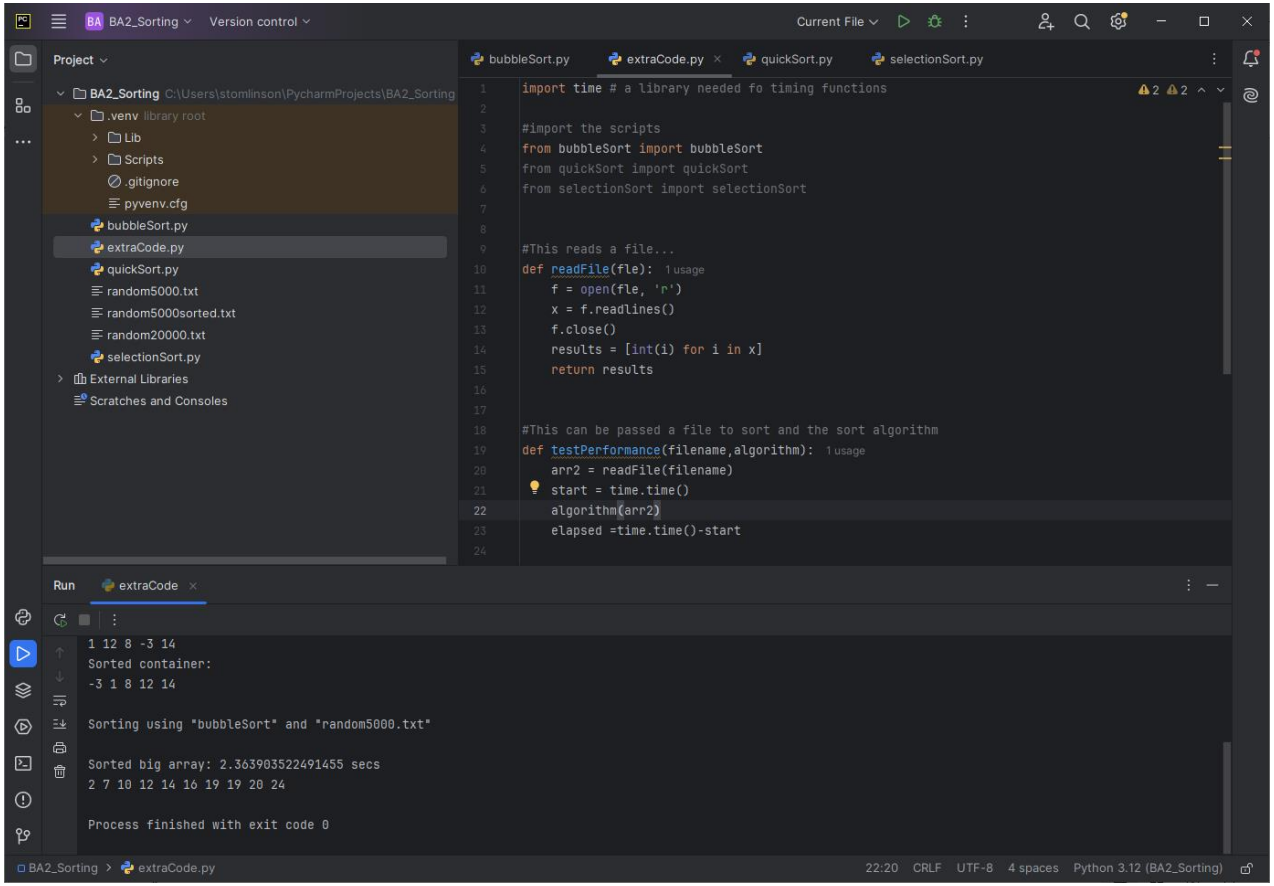
- Use Pycharm available on the teaching machines

Use PyCharm



- Download the class files from Learn and unzip the archive
- Create a new Project in Pycharm (here called BA2_Sorting)
- Select all the archive files and copy and paste the files into the folder (within the IDE). Files will then copy to the project folder

Use PyCharm



- Open one of the files and then click the green run icon to run the code
- The code that runs is the code that is selected in the edit window
- “extraCode.py” contains hints and extra code for this class…

Timing Execution

- Use extraCode.py to calculate execution time for each sort algorithm (you need to complete the code)
- For this purpose two large files are provided (random20000.txt, random5000.txt) together with a pre-sorted file (random5000sorted.txt).
- Chunks of code are provided to load these files into memory and then time the execution of an algorithm

For each algorithm…

- Rearrange the code so you can use it to time the execution of each algorithm
- Time how long it takes to load different length files
- Work out how long it takes to ‘sort’ the pre-sorted file
- Repeat the timings several times- do they change each time & if so why?
- Can you explain runtime performance in terms of algorithm design?

Extra Code Questions…

```
def bubbleSort(arr):
    swapped = True
    n = len(arr)
    x = 0

    # Traverse through all array elements
    while bool(swapped):
        swapped = False

        # Last I elements are already in place
        for j in range(0, n - 1 -x):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        x=x+1
```

- What is the purpose of the `swapped` variable?
- What is the purpose of the `x` variable?

Extra Code Questions

```
def partition(arr, left, right):
  i = left
  j = right
  tmp = 0
  pivot = arr[int((i + j) / 2)]

  while i <= j:
        while arr[i] < pivot:
            i+=1
        while arr[j] > pivot:
            j-=1
        if i <= j:
            arr[i], arr[j] = arr[j], arr[i]
            i+=1
            j-=1
  return i
```

```
def quickSort(arr, left, right):

      index = partition(arr, left, right)
      if left < index -1:
            quickSort(arr, left, index - 1)
      if  index < right:
            quickSort(arr, index, right)

def quickSortS(arr):
      quickSort(arr,0,len(arr)-1)
```

- Does the pivot need to be in the middle of the array?
- Why define quickSortS ??

Extra Code Questions

```
def testPerformance(filename,algorithm):
    arr2 = readFile(filename)
    start = time.time()
    algorithm(arr2)
    elapsed =time.time()-start

    #how to print the elapsed time
    print("")
    print("Sorted big array: %s secs" % elapsed)
    for i in range(10):
           print(arr2[i],end=" ")
    print("")
    return elapsed
```

- Would it make sense to start the timer before reading the array file to give an indication of the overall runtime performance

Extra Code Questions

```
def selectionSort(a):
    for i in range(len(a)):
        min = i
        for j in range(i + 1,
len(a)):
            if a[j] < a[min]:
                min = j

        # swap the elements
        a[i], a[min] = a[min], a[i]
```

- One problem with this algorithm is that it does not “know” to stop when the array is already sorted. Is it possible to make a small design change to make sensing the sorted state of the array possible??

Java version (just for information)

- If you would like to look at the Java version…
  - upload `java_sorting.zip`
  - unzip the archive
  - compile the file `javac SortClass.java`
  - run the Java program `java SortClass`
  - The code can be edited using Vi to compare the different sorting algorithms