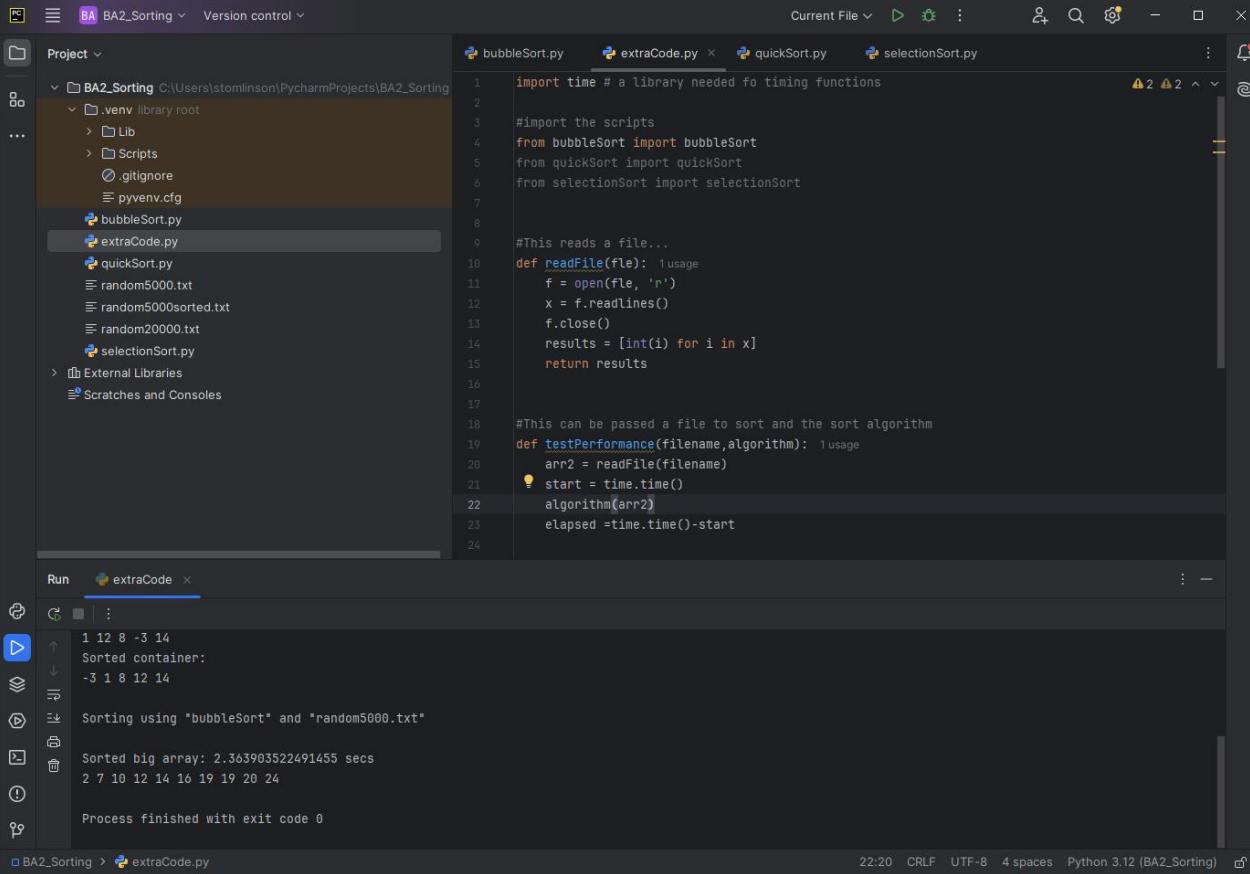


Implementing Algorithms

Bioinformatics Algorithms Week2

Simon Tomlinson

Use PyCharm



The screenshot shows the PyCharm IDE interface with the following details:

- Project View:** Shows the `BA2_Sorting` project structure. It includes a `.venv` folder containing `Lib`, `Scripts`, `.gitignore`, and `pyenv.cfg`. There are also files `bubbleSort.py`, `extraCode.py`, `quickSort.py`, and `selectionSort.py`, along with several `random*.txt` files.
- Code Editor:** The `extraCode.py` file is open, displaying Python code for sorting algorithms and file I/O. The code includes functions for reading files, performing bubble sort, quick sort, and selection sort, and testing performance.
- Run Tab:** The `extraCode` tab is selected in the run configuration dropdown. The run history shows:
 - 1 12 8 -3 14
Sorted container:
-3 1 8 12 14
 - Sorting using "bubbleSort" and "random5000.txt"
 - Sorted big array: 2.363903522491455 secs
2 7 10 12 14 16 19 19 20 24
 - Process finished with exit code 0
- Status Bar:** Shows the current time (22:20), encoding (CRLF), character set (UTF-8), and code style settings (4 spaces). It also indicates the Python version (Python 3.12) and the current file (BA2_Sorting > extraCode.py).

- Use Pycharm available on the teaching machines

Use PyCharm

The screenshot shows the PyCharm IDE interface with the following details:

- Project View:** Shows a project named "BA2_Sorting" containing files: bubbleSort.py, extraCode.py (selected), quickSort.py, and selectionSort.py.
- Code Editor:** Displays the content of extraCode.py. The code includes imports for time and various sorting functions, and defines two functions: readFile and testPerformance.
- Run Tab:** Shows the output of running extraCode.py. It displays sorted arrays and timing information for sorting a large array using bubbleSort.
- Status Bar:** Shows the file path as "BA2_Sorting > extraCode.py", encoding as "UTF-8", and Python version as "Python 3.12 (BA2_Sorting)".

- 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

The screenshot shows the PyCharm IDE interface with the following details:

- Project View:** Shows the `BA2_Sorting` project structure. It includes a `.venv` folder containing `Lib`, `Scripts`, `.gitignore`, and `pyenv.cfg`. The `extraCode.py` file is selected in the list.
- Edit Window:** Displays the contents of `extraCode.py`. The code reads a file named `random5000.txt` and sorts its contents using the `bubbleSort` function from the `bubbleSort.py` module. It also includes a `testPerformance` function to measure the execution time of different sorting algorithms on large files.
- Run Tab:** Shows the output of running the `extraCode.py` script. The output window displays:
 - The sorted array: `1 12 8 -3 14`.
 - The message: `Sorted container:`
 - The sorted array again: `-3 1 8 12 14`.
 - The message: `Sorting using "bubbleSort" and "random5000.txt"`.
 - The message: `Sorted big array: 2.363903522491455 secs`.
 - The sorted array: `2 7 10 12 14 16 19 19 20 24`.
 - The message: `Process finished with exit code 0`.
- Status Bar:** Shows the current file is `extraCode.py`, the encoding is `UTF-8`, and the Python version is `Python 3.12 (BA2_Sorting)`.

- 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 l 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