Project 02 Sorting

We will share few of the essential functions required by students to complete in their project.

Selection sort

```
def selection_sort(data: List[T], *, comparator: Callable[[T, T], bool] = lambda x, y: x < y,
                   descending: bool = False) -> None:
    """
    Sorts a list in place using selection sort algorithm.
    :param data: List of type T to be sorted
    :param comparator: A function which takes two arguments of type T and returns True when the first argument should be
     treated as less than or equal to the second argument.
    :param descending: Perform the sort in descending order when this is True
    """
    # Traverse the list with two indices, i & j.
    for i in range(len(data) - 1):

        # Find the index of the smallest element
        index_smallest = i
        for j in range(i + 1, len(data)):
            if do_comparison(data[j], data[index_smallest], comparator, descending):
                index_smallest = j

        # Swap data[i] and smallest element
        data[i], data[index_smallest] = data[index_smallest], data[i]
```

Bubble sort

```python
def bubble_sort(data: List[T], *, comparator: Callable[[T, T], bool] = lambda x, y: x < y,
                descending: bool = False) -> None:
    """
    Sorts a list in place using bubble sort algorithm.
    :param data: List of type T to be sorted
    :param comparator: A function which takes two arguments of type T and returns True when the first argument should be
     treated as less than or equal to the second argument.
    :param descending: Perform the sort in descending order when this is True
    """
    while True:
        # We can stop passing over the list when we get a full pass without having to swap
        performed_swap = False
        for i in range(len(data) - 1):
            # Swap elements if they're in the wrong order
            if do_comparison(data[i + 1], data[i], comparator, descending):
                data[i], data[i + 1] = data[i + 1], data[i]
                performed_swap = True
        # Went through whole list and nothing was out of order
        # That means we are done sorting
        if not performed_swap:
            break
```

Insertion sort is just the modified version of any of the basic sorts insertion, bubble, selection were shared in lecture notes….

Hybrid sort

```python
def hybrid_merge_sort(data: List[T], *, threshold: int = 12,
                      comparator: Callable[[T, T], bool] = lambda x, y: x < y, descending: bool = False) -> None:
    """..."""
    # If length is not more than one, data is already sorted
    if len(data) <= 1:
        return

    # Defer to insertion sort if data length is less than threshold
    if len(data) <= threshold:
        insertion_sort(data, comparator=comparator, descending=descending)
    else:
        # split the array in half
        middle = len(data) // 2
        left = data[:middle]
        right = data[middle:]

        hybrid_merge_sort(left, threshold=threshold, comparator=comparator, descending=descending)
        hybrid_merge_sort(right, threshold=threshold, comparator=comparator, descending=descending)

        # Set up indices to perform merge
        left_index = right_index = index = 0

        # Merge the left and right sub-arrays together
        while left_index < len(left) and right_index < len(right):
            # if left[left_index] < right[right_index]:
            if do_comparison(left[left_index], right[right_index], comparator, descending):
                data[index] = left[left_index]
                left_index += 1
            else:
                data[index] = right[right_index]
                right_index += 1
            index += 1

        if left_index < len(left):
            data[index:] = left[left_index:]
        else:
            data[index:] = right[right_index:]
```

Quick sort is already given, check solution.py