Nikita Semeniuk **ID:** 2722726
Wenjing Qiu

Task 4.

Part (a)

k	SelectionSort Time (s)	Comparisons	MergeSort Time (s)	Comparisons
500	0.000	125,249	0.000	7259
1,000	0.001	500,499	0.000	16019
2,000	0.003	2,000,999	0.000	35039
4,000	0.013	8,001,999	0.000	76079
8,000	0.053	32,003,999	0.001	164159
16,000	0.221	128,007,999	0.001	352319
32,000	0.908	512,015,999	0.003	752639

NOTE:

- **MergeSort Comparisons**:

  Grows as ~k $\log_2 k$ (matches theoretical O(n log n) complexity):

  - k=500: 7,259 ≈ 500 × 14.5
  - k=32,000: 752,639 ≈ 32,000 × 23.5

  Ratio: Comparisons/k ≈ $\log_2 k$ (confirms linearithmic growth)

- **SelectionSort vs. MergeSort**:

  At k=32,000:

  - SelectionSort: 512M comparisons
  - MergeSort: 753k comparisons (680× fewer)

Part (b)
Here is an example output of the last run:

```
        ===== ALL SORTS BENCHMARK =====
        ===== k = 100  =====

+---------------+---------------+---------------+---------------+
| Algorithm     | Time (s)      | Comparisons   | Exchanges     |
+---------------+---------------+---------------+---------------+
| Insertion     | 0.000004      | 2360          | 2459          |
| Selection     | 0.000012      | 5049          | 92            |
| Shell         | 0.000005      | 398           | 901           |
| Merge TD      | 0.000007      | 1127          | 1344          |
| Merge BU      | 0.000006      | 496           | 1093          |
| Quick         | 0.000005      | 352           | 418           |

        ===== ALL SORTS BENCHMARK =====
        ===== k = 200  =====

+---------------+---------------+---------------+---------------+
| Algorithm     | Time (s)      | Comparisons   | Exchanges     |
+---------------+---------------+---------------+---------------+
| Insertion     | 0.000016      | 8724          | 8923          |
| Selection     | 0.000036      | 20099         | 196           |
| Shell         | 0.000011      | 986           | 2189          |
| Merge TD      | 0.000013      | 2555          | 3088          |
| Merge BU      | 0.000010      | 1171          | 2447          |
| Quick         | 0.000009      | 966           | 1099          |

        ===== ALL SORTS BENCHMARK =====
        ===== k = 500  =====

+---------------+---------------+---------------+---------------+
| Algorithm     | Time (s)      | Comparisons   | Exchanges     |
+---------------+---------------+---------------+---------------+
| Insertion     | 0.000090      | 52603         | 53102         |
| Selection     | 0.000213      | 125249        | 493           |
| Shell         | 0.000034      | 2729          | 6235          |
| Merge TD      | 0.000035      | 7259          | 8976          |
| Merge BU      | 0.000028      | 3418          | 6936          |
| Quick         | 0.000027      | 2379          | 2710          |
        ===== ALL SORTS BENCHMARK =====
        ===== k = 1000 =====

+----------------+----------------+----------------+----------------+
| Algorithm      | Time (s)       | Comparisons| Exchanges   |
+----------------+----------------+----------------+----------------+
| Insertion      | 0.000457       | 224735         | 225734         |
| Selection      | 0.000854       | 500499         | 993            |
| Shell          | 0.000082       | 7321           | 15327          |
| Merge TD       | 0.000075       | 16019          | 19952          |
| Merge BU       | 0.000062       | 7654           | 15364          |
| Quick          | 0.000054       | 5977           | 6645           |
+----------------+----------------+----------------+----------------+
```

Algorithm running times presented below (average of 5 runs in seconds):

Algorithm	**k=100**	**k=200**	**k=500**	**k=1000**
Insertion	0.000004	0.000012	0.000115	0.000370
Selection	0.000011	0.000039	0.000212	0.000833
Shell	0.000005	0.000013	0.000036	0.000080
Merge (TD)	0.000006	0.000012	0.000034	0.000068
Merge (BU)	0.000006	0.000012	0.000029	0.000062
Quick	0.000004	0.000009	0.000024	0.000058

Key observations:

1. **Quick Sort is consistently the fastest across all array sizes**
2. **Insertion and Selection sorts show quadratic growth (time ~4x when array size doubles)**
3. **Shell/Merge/Quick sorts show near-linear growth (time ~2x when array size doubles)**
4. **Merge Bottom-Up is slightly faster than Merge Top-Down for larger arrays**