CSE 2246 Analysis of Algorithms

Assignment 1

Havva Nur Özen 150119771

Ezgi Eren 150119515

Ikranur Akan 150123827

Table of Contents

Designing the Experiment

Deciding on Reasonable Inputs

1. **[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]**
    - For Insertion Sort : We chose this sorted list of 10 elements as this represents the **best-case** time complexity of O(n) where the algorithms can terminate early due to minimal comparisons or swaps.
    - For Quick Sort Algorithm : If the first element is always chosen as the pivot, each partition will only contain one element (the pivot), and all other elements will be on the other side. This results in a recursion tree with a height equal to the number of elements in the array, leading to a **worst-case** time complexity of O(n^2).
    - Quick Select with Median-of-Three Pivot Selection : In Sorted or nearly sorted arrays , the median-of-three pivot selection might always pick the same element (like the middle one) as the pivot, leading to very unbalanced partitions. This imbalance forces the algorithm to perform many unnecessary comparisons on the larger sub-array in each recursive step. The process continues until reaching the desired k-th element, resulting in a slower runtime O(n^2) in the **worst case.**
    - Quick Select with Median-of-Medians Pivot Selection : Since the array is already sorted, choosing the middle element as the pivot consistently partitions the array into very unbalanced sub-arrays.
    - For Max Heap: We chose this sorted list of 10 elements as this represents the **worst-case** scenario for these algorithms where they require maximum operations to reach the desired outcome.
    - For Merge Sort algorithm there is **no worst- or best-case** scenario.
2. **[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]**
    - For Insertion Sort : We used this reverse sorted list of 10 elements to demonstrate the **worst-case** scenario where these algorithms perform the maximum number of comparisons and swaps.
    - For #1 and #2 Quick Select algorithm : when choosing the first element as the pivot during partitioning, the array is already sorted in either ascending or descending order. This leads to the most unbalanced partitioning, where one

partition has all the elements except the pivot, and the other partition has no elements. This results in a recursion tree with a height equal to the number of elements in the array, leading to a **worst-case** time complexity of O(n^2).

- Quick Select with Median-of-Three Pivot Selection : When the data is sorted, the median-of-three selection, which typically picks the middle element from the first, last, and middle positions of the array, will consistently choose a value near the center. In a sorted array, this translates to always picking an element that separates all the smaller elements on one side and all the larger elements on the other. This creates a highly unbalanced partition, with one sub-array containing almost all the elements (n-1) and the other containing only one element. Therefore, this input **worst case** time complexity.
- Quick Select with Median-of-Medians Pivot Selection : When elements are not sorted in a particular order, the Median-of-Median selection is more likely to choose a pivot that leads to a balanced partition but in this input, array is sorted descending order choosing a pivot leads to unbalance therefore, this input **worst case** time complexity.
- For Quick Sort Algorithm : If the first element is always chosen as the pivot, each partition will only contain one element (the pivot), and all other elements will be on the other side. This results in a recursion tree with a height equal to the number of elements in the array, leading to a **worst-case** time complexity of O(n^2).
- For Max Heap: We selected this reverse sorted list of 10 elements to highlight the **best-case** scenario for these algorithms where they still operate efficiently despite the input being reverse sorted.
- For Merge Sort algorithm there is **no worst or best-case** scenario.

3. **[5, 2, 9, 1, 6, 3, 8, 4, 10, 7]**
4. **[81, 19]**
5. **[56, 12, 87, 3, 67, 41, 25, 92, 10, 77]**
6. **[5, 2, 7, 1, 9, 3, 8, 4, 10, 6]**
7. **[7, 4, 3, 9, 1, 6, 8, 10, 2, 5]**
8. **[8, 3, 10, 5, 1, 9, 7, 2, 6, 4]**
9. **[302, 29, 81, 442, 446, 412, 779, 813, 307, 127, 841, 239, 980, 693, 366, 695, 282, 857, 76, 160, 124, 240, 600, 440, 265, 444, 354, 861, 598, 945, 586, 266, 784, 786,**

314, 268, 857, 124, 234, 310, 833, 147, 170, 227, 725, 834, 339, 719, 798, 765, 211, 389, 825, 209, 951, 71, 441, 404, 782, 298, 461, 107, 638, 289, 996, 389, 497, 729, 703, 880, 847, 575, 914, 132, 786, 631, 642, 67, 369, 863, 232, 416, 941]

10. **[260, 35, 542, 900, 686, 520, 894, 609, 683, 90, 44, 150, 164, 117, 919, 354, 4, 715, 907, 884, 42, 480, 560, 522, 434, 541, 944, 288, 197, 294, 257, 96, 576, 170, 170, 992, 569, 190, 24, 857, 131, 438, 175, 933, 401, 900, 990, 31, 375, 7, 204, 930, 182, 363, 846, 231, 848, 550, 173, 450, 946, 61, 745, 531, 287, 751, 219, 359, 615, 303, 490, 638, 948, 223, 216, 265]**
11. **[38, 94, 33, 86, 57, 25, 7, 35, 34, 53, 61, 67, 74, 10, 68, 65, 33, 2, 60, 39, 62, 81, 87, 5, 24, 4, 28, 74, 2, 39, 23, 49, 5, 70, 61, 6, 81, 52, 7, 50, 66, 92, 25, 32, 45, 11, 2, 37, 49, 47, 7, 45, 24]**
12. **[39, 12, 16, 83, 92, 8, 44, 62, 100, 52, 76, 71, 13, 24, 55, 41, 60, 10, 26, 63, 22, 21, 97, 84, 50, 78, 68, 73, 96, 81, 37, 2, 69]**
13. **[340, 695, 140, 868, 216, 543, 199, 323, 602, 327, 263, 929, 198, 328, 616, 302, 807, 539, 791, 122, 493, 117, 850, 411, 879, 675, 328, 873, 761, 711, 285, 392, 116, 898, 416, 506, 344, 463, 126, 808, 746, 256, 822, 374, 288, 860, 345, 452, 944, 191, 169, 408, 768, 402, 639, 197, 816, 192, 950, 623, 515, 409, 935, 799, 497, 763, 662, 451, 654, 463, 438, 632, 448, 682, 884, 148, 539]**
14. **[82, 86, 11, 72, 25, 36, -81, 70, 25, 87, -59, 7, 70, -1, 32, -60, 67, -46, 9, -98, -16, -40, 25, -11, 55, 79, -98, -95, -30, 64, 42, 64, 75, -86, -80, 21, -82, 75, 58, -6, 3, -5, -91, 84, 93, 4]**
15. **[83, -78, -99, -24, -67, -90, -71, 11, -94, -85, 59, 100, -47, -20, 52, -59, -43, 36, 94, -62, 47, -12, -87, 10, 60, 12, 63, -78, 97, 30, -93, 65, 83, 51, 6, -94, 85, -18, -8]**
16. **[-146, 98, -457, 76, -478, -169, -290, -782, 77, -624, 85, -602, -249, -151, -690, -156, -54, -9, -237, -464, 61, -456, -437, -281, -670]**
17. **[38, 36, 100, -58, -70, 65, 10, 2, -48]**
18. **[-2, 40, 9, 15, -1, 52, 15, 92, -6, 12, 46, 16]**
19. **[84, 30, 28, 94, -10, 33, 63, 86, 31, -2, 52, 15, 30, 14, 6, 15, 24, 52, 71, 12, 97, 31, -7, 64, 88, 5, 67, 49, 23, 87, -1, 99, -5, 44, -4, 12, 52, 62, 96, 31, 1, 49, 39, 73, -2, 74, 6, 10, 65, 86, 42, 23, 7, 93, 72, 9, 84, 57, 43, 1, 16, 39, 57, 95, 4, 22, 65, -4, 33, 87, 73, 95, 64, 52, 80, 9, 61, 51, 14, 37, 42, 88, 39, 0, 16, -9, 57, 49, 96, 11, 26, 17, 80, 41, 41, 43, 94, 64, 48, 16, 48, 15, 50, 0, 48, 93, 2, 97, 94, 16, -10, 71, 68, 94, 31, 84, -5, -4, 96, 69, 69, 55, 0, -3, 63, 21, 10, 90, 5, 93, 57, 76, 86, 98, 71, 57, 95, 89, 100, 96, 36, 35, 44, 31, 80, 10, -4]**

20. **[60, 89, 77, 47, -3, 93, 17, 88, 34, 79, 95, 96, -10, 35, -7, 47, 12, 38, 52, 38, 14, 0, 49, 99, 96, 9, 70, 83, -5, 59, 53, 10, 89, 34, 8, 83, 97, 91, 57, 68, 21, -10, 99, 19, 75, 83, 72, 14, 65, 36, 88, -9, 1, 76, -4, 94, 14, 8, 49, 89, 60, 20, 70, 46, 32, 20, 71, 89, 23, 27, -8, 60, 21, 79, 8, 88, 41, 44, 9, -8, -9, 14, 96, 37, 7, 67, 72, -1, 30, 36, 22, 83, 6, -7, 60, -8, -7, 71, 15, 62, 73, 98, 43, 25, 50, 21, 23, 43, 10, 32, 22, 50, 74, 81, 43, 60, 42, 80, 87, 65, -4, -10, 23, -10, 98, 33, 98, 28, -3, 59, 85, -7, 2, 16, 14, 30, -1, 82, 75, 29, 38, 71, 6, 25, 28, 88, 71, 4, 51, -7, 51, 10, 57, -5, 88, 73, 24, 41, 15, 28, 5, 12, 90, 25, 4, 96, 52, 81, 79, 90, 44, 55, -9, 33, 48, 88, 71, 70, 13, 59, 53, 57, 88, 99, 2, 31, 89, 5, 96, 15, 58, 79, -6, -1, 8, -10, -10, 45, 37, 72, 42, 39, 32, 53, 11, 30, 12, 53, 43, 50, 47, 32, 93, 8, 26, 40, 7, 29, 6, -2, 57, 21, 24, 89, 72, 17, 17, 55, 29, 62, 22, 65, 88, 80, 23, 62, 26, -7, -8, 37, -3, 93, 94, 44, 60, 4, 21, 83, 33, 73, 71, 19, 29, 62, 40, 70, 16, 94, 58, -4, 0, 1, 76, 1, 26, 28, 46, 61, 6, 11, 97, 66, 38, 33, 51, 84, 68, 49, 76, 46, 27, 70, 6, 11, 49, 51, 85, 71, 38, 55, 22, 35, 69, 27, 18, 59, 27, 2, -8, 60, 45, 42, 33, -2, 45, -5, 97, 84, 69, 26, 71, 65, 48, 15, 72, 20, 59, 89, 4, -9, 68, 92, 75, 97, 62, 32, 47, 8, 18, 36, 95, -4, -6, 83, 49, 93, 98, 17, 76, 11, 48, 14, 24, 10, 25, 38, 33, 20, 39, 47, 100, 81, 31, 70, 19, 32, 50, 10, 1, 7, 11, 10, 33, 78, 82, 64, 68, 24, 22, 78, 82, 16, 29, 46, 26, 16, 36, 65, 45, -1, 29, 0, -5, 86, -1, 98, 40, 28, 36, 58, 14, 72, 45, 96, 69, 52, 5, -4, 76, 50, 5, 73, 98, 48, 19, 99, -1, 0, 19, 92, -7, 50, 30, 69, 7, -2, -1, 27, 84, -5, 84, 31, 83, 80, 19, 66, 57, -4, 86, 77, 66, 44, 74, 21, 12, 32, 22, 86, 30, 97, 75, 86, 53, 64, 80, 84, 62, 27, 32, 51, 81, 53, 87, 78, 67, 25, 34, 92, 4, 98, 72, 61, -6, 11, 70, 100, 53, 85, 35, 75, 8, 4, 57, 88, 88, 0, 75, 17, 95, 83, 12, 82, 67, 63, 0, 83, 84, 81, 34, 39, 40, 25, -1, -8, 100, 47, 76, 47, 10, 6, 36, 87, 93, 92, 9, 62, 93, 75, 32, 54, 53, 84, -7, 4, 32, 62, 25, 58, 74, 10, 49, 8, 5, 29, 9, 74, 53, 16, 54, 38, 63, 5, 85, 78, 93, 34, 61, 14, 15, 67, -7, 12, 68, -4, 28, -10, 28, 33, -4, 100, 31, 67, 52, -3, 42, 97, 54, 77, -5, 90, 89, 39, 73, -4, 16, 8, 80, 44, 10, 89, 0, 100, -9, 98, 52, 1, 45, 57, 81, 99, 62, 85, 86, 26, 64, 44, 98, 48, 72, 14, 67, 27, 46, 2, 13, -6, 96, 91, 10, 5, 69, 70, 82, -1, 51, 55, 23, 21, 64, 77, 46, -6, 91, 51, 20]**
21. **[5, 9, 13, 2, 1, 5, 11, 5, 1, 2, 13, 9, 5]**
22. **[-57, 80, 29, -55, 12, 49, -50, 18, 56, 61, 15, 90, -6, 34, 59, -37, -70, 76, -99, -46, -94, 20, -96, 88, 5, 19, 21, -60, -79, 28, 46, -26, -60, -51, 41, -80, -28, 39, 78, 37, -77, 56, 45, -14, 32, -49, 66, 38, 79, 45, -90, 10, 15, 89, 36, -66, 18, 86, -16, -40, 18, -54, 37, 18, 2, -79, 18, 7, -83, 82, 90, 16, -51, 84, 8, 96, 38, -47, -62, 48, -74, -6, -56, 3, -69, -80, 61, -99, -84, 86, -10, -99, -42, -55, 4, 28, 66, -49, 98, 31, 31, -23, -74, -3, 21, 22, 83, -4, 77, 7, -12, 5, -75, 3, 90, 21, 29, 71, -90, -6, 46, 70, 37, -94, 6, -61, -91, -5, -90,**

-11, 33, -28, -7, -71, -71, 8, -65, -3, -59, 61, 34, -69, 5, 87, -27, 27, -51, -32, 36, 78, -60, 81, -12, 30, 90, -82, 48, -99, 47, -26, 0, 10, 28, 56, -34, 76, 39, 72, 41, 74, -56, 100, 76, 99, 70, -57, -40, -14, -29, -94, -29, -42, 6, -40, 38, -22, 90, 64, -96, 0, -36, -66, 25, 80, -28, 57, 78, -14, 12, 58, -49, 34, -8, 51, -53, 9, -21, -14, 61, -70, -100, 31, -27, -56, -6, 40, 83, -41, -10, 34, -43, 33, 47, -24, -76, 19, 30, 61, -99, 21, -6, -75, -8, -71, -46, 100, -76, 44, 35, 5, 78, -52, -52, 69, 37, 49, 95, -19, -6, 42, -14, 29, -65, -63, 72, 5, 56, 9, -61, -5, -75, -47, 95, 20, -65, 50, 43, 61, -79, -74, 87, -23, -39, 76, 14, -97, -59, -41, 100, 88, -74, 95, 5, -29, -84, -73, -8, -77, 39, 78, -22, 13, 70, 37, 85, 26, -14, -99, -87, -86, -4, 78, 32, -13, -20, -26, 27, 30, -52, -13, -81, 60, 69, -84, -24, 44, 57, 24, -65, 42, 0, 20, 2, 20, -83, -35, 4, -60, -46, -46, -71, -2, -2, -69, -24, 41, 51, -12, 25, 10, 24, -51, -76, 55, -35, 41, 35, 82, -46, -24, -5, 4, 27, 18, -100, -24, 69, 94, -74, 99, -74, -58, 79, -3, 75, 85, 58, -3, 31, -5, 8, -88, 89, 17, 32, 51, 2, 0, 91, -6, -26, 16, -16, -66, -42, -87, -51, -29, -37, -5, 49, 39, -74, -39, -12, 28, -36, 50, 27, 33, -16, 87, -87, 92, 86, -34, 54, -5, 65, 79, 36, 73, 74, 46, 17, -67, 8, -96, -74, -4, -75, -76, 91, 92, 40, -24, -19, -37, 33, 85, 70, 67, 99, -65, -79, -29, 27, 52, 51, -17, -89, 77, -90, 44, -8, -61, 84, -22, 78, 5, -51, 92, 33, -61, 63, 75, 54, 93, -75, 31, -14, -50, 66, 90, -100, 90, 34, 31, -29, 63, 37, -29, -69, -8, -5, 70, 96, -3, -99, 61, 5, -86, 18, -86, 10, -3, -84, -98, -78, -21, -31, 74, -46, 54, -41, -98, 57, 16, -8, 60, -61, -64, -57, 87, 47, 79, -22, -39, -90, -56, -85, -46, 82, 24, -12, 56, -7, -54, 28, 22, -49, 26, 14, -91, -39, -41, -23, 6, 18, -12, -7, -21, -57, -50, -29, -17, 58, 37, -98, -70, 63, -62, 98, 99, 37, 42, 94, 71, 90, 44, -69, 73, 2, -92, 58, 76, -43, -38, -7, -68, 90, -29, 97, -79, 98, 83, -68, -47, -34, -30, 74, 69, -76, 90, 6, 11, -72, 42, -88, -49, 73, 58, 4, 75, -5, 22, 37, 10, 70, 25, -53, 83, -4, -21, -61, 79, -24, 11, 11, -77, -69, -2, 27, 45, 54, 56, -67, 55, -67, -52, -93, -36, 9, -18, 2, 12, -71, -21, -18, -13, 10, -45, 73, -43, -42, -47, 56, 62, -88, -5, -12, 45, -51, -64, 1, -98, -44, -38, 96, 84, 52, 36, -43, 21, -16, -30, -100, -11, -73, 81, 95, 38, 41, -88, -30, -83, -33, 64, 10, -24, -36, 17, -28, 42, -54, -16, 17, -35, -57, 60, -35, -92, -74, 82, 88, -71, 69, -96, 40, -53, -32, -70, -40, -32, 4, -53, 99, 88, -41, 90, -41, -70, -67, 17, 72, -51, -61, -24, -89, 55, 34, -20, 59, -13, 50, 6, 81, 96, 46, 90, 89, 86, 54, -93, 33, 56, -68, -7, 73, -75, 28, 40, 99, -32, -9, -35, -96, -51, 97, 95, -92, -53, -8, 90]

23. **[56, -23, 14, 45, 72, 89, 51, 0, 64, 17, 25, -22, -14, 52, -5, 30, 36, -72, 14, 85, -60, 95, 10, -12, -58, -17, 46, 16, -26, 71, -63, 81, 54, -19, 45, 100, -2, -8, 34, -26, -28, 54, 81, 35, -57, -38, 94, -99, 29, 87, -86, 14, -87, 53, -8, 24, -52, 86, 22, -75, 98, 35, -21, -96, -34, 94, -61, -27, 28, 10, -69, -40, 71, 32, 76, -78, -66, 83, 90, 37, 61, 9, -30, -42, -61, 14, -94, 79, 19, -78, 54, -49, -14, -96, -10, -24, 43, -89, -92, -34, -83, 45, 3, 24, 62, -**

62, -20, -54, -42, 85, 67, -25, -65, 16, 71, 60, 100, 64, -77, -15, -72, 50, -9, -3, -94, -9, -96, -72, 10, -7, -4, -50, -99, 74, 81, 100, 92, -64, -76, 34, -18, -47, 49, -100, 39, 73, 81, -13, -100, -19, -90, 79, 19, 11, -8, -29, -80, 44, 21, -35, -52, 36, -86, 35, 59, -17, 28, -56, 10, 40, -76, 24, 99, 62, -4, -6, -26, 23, 22, 30, 81, 5, 24, 34]

24. **[-30, 25, 13, -6, -45, -6, 23, -33, 26, -32, -2, -30, -23, 21, -23, 20, 6, -4, 35, 36, -46, -23, -10, -15, -43, -23, -15, -21, -26, 34, 21, -6, -36, 1, 10, -3, 13, -39, -45, 3, -29, -5, 38, -11, 36, -2, 18, 12, -9, 9, -19, 24, -41, 23, 3, -27, 23, -44, 0, 29, -7, 19, -30, 40, 37, 23, -47, 39, 1, 10, 36, -7]**
25. **[-44, -26, -44, 35, -5, 24, -35, 19, 31, 1, -41, -17, -45, -25, -34, -44, 2, -5, -24, -41, -42, 0, -33, 9, 28, -40, -8]**
26. **[21, 33, 35, -11, -6, 9, -32, -9, 34, 13, -20, -31, -25, -47, 33, -35]**
27. **[-16, -18, -45, 2, -22, -40, 37, -7, 15, -36, 33, -14, 35, 29, 40, -38, 23, -37, -23, 35, -38, -26, 3, 33, -11, 24, -48, 12, -7, 16, 35, -16, -26, -19, -23, 0, -8, 36, 30, 16, -4, 19, -22, -29, -28, -32, -22, 16, 28, -22, -25, -36, 32, -42, -16, 32, -19, -42, 16, -1, -44, 32, -48, 10, -11, 31, 19, 24, -41, -8, 5, -26, -31, -41, -48, -27, -40, 28, 14, -37, -25, -26, 7, -44, -32, -33, -30, 1, -24, 5, 9, 17, 27, -43, -36, 40, 26]**
28. **[42, 19, 44, 46, 4, 42, 15, 4, 49, 27, 12, 10, 2, 19, 29, 11, 49, 50, 29, 30, 13, 45, 31, 21, 28, 29, 16, 17, 24, 49, 1, 20, 12, 43, 18, 2, 37, 14, 13, 44, 17, 18, 18, 32, 31, 33, 25, 25, 10, 11, 8, 8]**
29. **[42, 36, 39, 43, 17, 50, 49, 14, 31, 19, 35, 31, 18, 33, 7, 47, 47, 49, 13, 43, 50, 27, 43, 47, 25, 17, 3, 38, 32, 36, 8, 25, 41, 6, 20, 37, 26, 44, 36, 28]**
30. **[5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5]**

- For Insertion Sort, Merge Sort, Quick Sort, Max Heap, Quick Select, Quick Select with Median-of-Three Pivot Selection, and Quick Select with Median-of-Medians Pivot Selection: We used the inputs 3-30, which are random lists, to simulate the average-case scenario where the input is not sorted in any order.

Deciding on Reasonable Metrics

In the experiment, we utilized nanoseconds as the unit of measurement for code execution time, while milliseconds were used for graphical representation. This approach ensured that we could accurately capture and analyse fine-grained timing details at the code level, while also presenting the results in a visually comprehensible format. For the input

arrays we chose integers. Some of the inputs we generated ourselves but for some inputs we wrote a python code that randomly generates an input with various sizes and various integers.

Coding and Running

All sorting algorithms implemented in Java language. They are located inside the Zip file.

```
Input Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Algorithm which is used: Insertion Sort
Result: 5
EXECUTION TIME(nanosecond):21145300
```

```
Input Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Algorithm which is used: Merge Sort
Result: 5
EXECUTION TIME(nanosecond):17278400
```

```
Input Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Algorithm which is used: Quick Sort
Result: 5
EXECUTION TIME(nanosecond):16750900
```

```
Input Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Algorithm which is used: Max Heap
Result: 5
EXECUTION TIME(nanosecond):65922500
```

```
Input Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Algorithm which is used: Quick Select (Median-of-Three Pivot)
Result: 5
EXECUTION TIME(nanosecond):14303100
```

```
Input Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Algorithm which is used: Quick Select (First Pivot)
Result: 5
EXECUTION TIME(nanosecond):16440600
```

```
Input Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Algorithm which is used: Quick Select (Median-of-Medians Pivot)
Result: 5
EXECUTION TIME(nanosecond):21631000
```

```
Input Array: [8, 3, 10, 5, 1, 9, 7, 2, 6, 4]
Algorithm which is used: Insertion Sort
Result: 5
EXECUTION TIME(nanosecond):22597500
```

```
Input Array: [8, 3, 10, 5, 1, 9, 7, 2, 6, 4]
Algorithm which is used: Merge Sort
Result: 5
EXECUTION TIME(nanosecond):16189700
```

```
Input Array: [8, 3, 10, 5, 1, 9, 7, 2, 6, 4]
Algorithm which is used: Quick Sort
Result: 5
EXECUTION TIME(nanosecond):19712600
```

```
Input Array: [8, 3, 10, 5, 1, 9, 7, 2, 6, 4]
Algorithm which is used: Max Heap
Result: 5
EXECUTION TIME(nanosecond):17582700
```

```
Input Array: [8, 3, 10, 5, 1, 9, 7, 2, 6, 4]
Algorithm which is used: Quick Select (First Pivot)
Result: 5
EXECUTION TIME(nanosecond):17598600
```

```
Input Array: [8, 3, 10, 5, 1, 9, 7, 2, 6, 4]
Algorithm which is used: Quick Select (Median-of-Three Pivot)
Result: 5
EXECUTION TIME(nanosecond):15312100
```

```
Input Array: [8, 3, 10, 5, 1, 9, 7, 2, 6, 4]
Algorithm which is used: Quick Select (Median-of-Medians Pivot)
Result: 5
EXECUTION TIME(nanosecond):19795700
```

```
Input Array: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Algorithm which is used: Merge Sort
Result: 5
EXECUTION TIME(nanosecond):14407300
```

```
Input Array: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Algorithm which is used: Insertion Sort
Result: 5
EXECUTION TIME(nanosecond):11791900
```

```
Input Array: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Algorithm which is used: Quick Sort
Result: 5
EXECUTION TIME(nanosecond):15727500
```

```
Input Array: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Algorithm which is used: Max Heap
Result: 5
EXECUTION TIME(nanosecond):18201400
```

```
Input Array: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Algorithm which is used: Quick Select (First Pivot)
Result: 5
EXECUTION TIME(nanosecond):16703900
```

```
Input Array: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Algorithm which is used: Quick Select (Median-of-Medians Pivot)
Result: 5
EXECUTION TIME(nanosecond):18292900
```

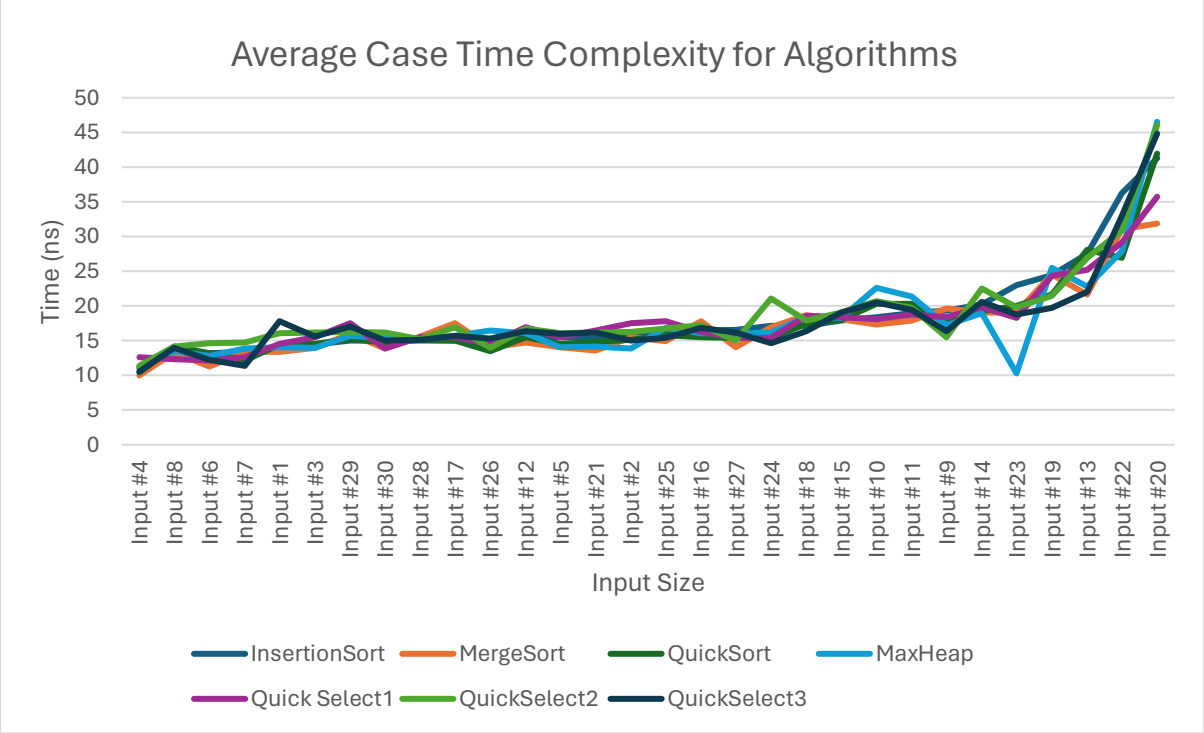
```
Input Array: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Algorithm which is used: Quick Select (Median-of-Three Pivot)
Result: 5
EXECUTION TIME(nanosecond):20024700
```

```python
import random


def generate_random_array():
    # Generate a random size for the array (between 0 and 100)
    array_size = random.randint(1, 100)
    #array_size = random.randint(1, 1000)

    # Generate an array of random integers
    random_array = [random.randint(-100,100) for _ in range(array_size)]

    return random_array


# Example usage:
random_array = generate_random_array()
print("Random array:", random_array)

```

Random Array Generating code was written in Python to creating input arrays.

Illustrating and Analysing Results

Table and Graph



Average Case Time Complexity for Algorithms Graph

Inputs/ Algorithms	Insertion Sort	Merge Sort	Quick Sort	Max Heap	Quick Select1	Quick Select2	Quick Select3
Input #4	11,0551	10,0014	10,426	11,2851	12,5798	11,3909	10,5626
Input #8	12,9064	13,2681	14,1523	13,5318	12,3752	14,1586	13,9371
Input #6	13,1703	11,2744	13,2395	12,7802	12,1596	14,6442	12,2051
Input #7	13,3688	13,3908	12,1396	13,915	12,6257	14,7003	11,3498
Input #1	14,2038	13,4099	14,4792	14,0883	14,5678	16,0371	17,7857
Input #3	14,4478	14,0124	14,5059	13,9227	15,4766	16,2109	15,6171
Input #29	14,9808	15,9442	15,2596	15,659	17,529	16,2278	17,0893
Input #30	14,9825	13,8584	14,7803	15,4114	13,869	16,1864	15,054
Input #28	14,9886	15,6084	15,1643	15,1967	15,5189	15,1823	15,1571
Input #17	14,9973	17,4788	15,0665	15,7049	15,5001	16,9064	15,6711
Input #26	15,0345	14,0411	13,5289	16,4328	14,5392	14,0091	15,3199
Input #12	15,3823	14,7663	15,4585	16,0774	16,9396	16,7072	16,4144
Input #5	15,5447	14,0851	14,569	14,0169	15,4637	16,0407	15,9501
Input #21	15,5521	13,5877	14,8635	14,1196	16,5051	16,1503	16,1655
Input #2	15,9373	15,4706	15,1511	13,8546	17,5695	16,2342	15,0063
Input #25	16,2337	14,9129	15,8378	16,8924	17,816	16,7117	15,3887
Input #16	16,4718	17,7664	15,517	16,0457	16,2586	17,1927	16,8147
Input #27	16,5347	14,0399	15,3876	15,9939	15,4111	14,9886	16,1437
Input #24	17,1142	16,9325	15,44604	16,1478	15,4302	21,0623	14,6387
Input #18	17,183	18,718	17,4533	18,4879	18,6205	17,923	16,3444
Input #15	17,9169	18,1345	18,05	18,291	18,3665	19,0489	19,0538
Input #10	18,4295	17,3175	20,2972	22,5827	18,11	20,6475	20,5424
Input #11	18,9916	17,9182	20,2657	21,4104	18,7768	19,7629	19,4421
Input #9	19,3473	19,6363	17,5341	17,2483	18,3629	15,5293	16,4112
Input #14	20,232	19,0674	19,2525	18,9729	19,8008	22,499	20,5959
Input #23	22,9745	19,1029	20,0201	10,29	18,3321	19,8047	18,7533
Input #19	24,406	24,5586	21,684	25,4959	24,3632	21,4174	19,7246
Input #13	27,4879	21,6716	28,0978	22,7972	25,1809	26,9859	22,0106
Input #22	36,2258	31,0949	26,9597	27,9294	29,187	30,9053	32,9555
Input #20	41,2883	31,8787	41,9553	46,5375	35,7377	46,0829	44,8281

Elapsed Time for each Algorithm and each Input Table

Algorithms	Insertion Sort	Merge Sort	Quick Sort	Max Heap	Quick Select 1	Quick Select 2	Quick Select 3
Average (ns)	18246316.67	17092916.13	17483496.13	17703980	17765770	18711616.67	17897760

Resulted Average Time for Each Algorithm Table

Comparisons

When comparing the performance of Merge Sort and Max Heap algorithms on an input of 25 elements ranging from -44 to 35, the graph illustrates that Merge Sort completes in 14,9129 milliseconds, outperforming the Max Heap algorithm, which takes 16,8924 milliseconds to execute.

Input #20 contains 615 elements ranging from -10 to 100. Looking at the table, it's clear that this array takes the longest time to execute among all the inputs. Because of its size relative to the others, it gives us a good idea of how algorithms behave with larger inputs. In this case, the Max Heap algorithm shows the poorest time complexity, while the Merge Sort algorithm exhibits the best time complexity.

Input #30 contains 14 elements, all of which are 5. Like the previous inputs, the Max Heap algorithm exhibits the poorest time complexity in this scenario too. Conversely, the Merge Sort algorithm demonstrates the best-case performance.

Input #1 consists of 10 elements in ascending order, ranging from 1 to 10. This input represented the best-case scenario for algorithms like Insertion Sort, Merge Sort, Quick Sort, Quick Select, and Quick Select with Median-of-Three Pivot Selection, with Merge Sort exhibiting the fastest execution time. However, it was a worst-case scenario for Max Heap and Quick Select using Median-of-Medians Pivot algorithms, with Quick Select using Median-of-Medians Pivot showing the slowest execution time among these algorithms.

Input #2 consists of 10 elements arranged in descending order from 1 to 10. For Max Heap and Quick Select using Median-of-Medians Pivot algorithms, this input represented the best-case scenario, with the Max Heap algorithm showing the best execution time. However, for algorithms like Insertion Sort, Merge Sort, Quick Sort, Quick Select, and Quick Select with Median-of-Three Pivot Selection, it was the worst-case scenario, with the Quick Select algorithm exhibiting the slowest execution time among them.

Input #4 contains only 2 elements: 81 and 19. Looking at the table, it's noticeable that this array has the shortest execution times across all the algorithms. Because of its small size compared to the others, it provides insight into how algorithms perform with smaller inputs. In this instance, the Merge Sort algorithm shows the most efficient time complexity, while the Quick Select algorithm displays the least efficient time complexity.

Theoretical Expectations vs Found Results

Our expectations for Input #1, which has an ascending sort, were that the best-case scenario would be in Insertion Sort, Merge Sort, Quick Sort, Quick Select, and Quick Select with Median-of-Three Pivot Selection. Our best-case scenario has a size of 10 integers. As we can see in the table the other 10 integer sized inputs take more time to sort and find the median then the best-case, we chose. For this case we can see that our expectations were met.

We expected the Max Heap and Quick Select algorithms using Median-of-Medians Pivot Selection to perform poorly in the worst-case scenario involving a list of ten integers based on our investigation of Input #1, which was arranged in ascending order. However, our analysis showed that other inputs with ten integers or more sorted and found the median more quickly than we had predicted for the worst-case scenario. This observation confirms what we had anticipated.

We first predicted that algorithms such as Insertion Sort, Merge Sort, Quick Sort, Quick Select, and Quick Select with Median-of-Three Pivot Selection would reach their worst-case situation when evaluating Input #2, which is ordered descending. This expectation was specifically targeted at a list containing ten integers. Examining the data in the table, we found that, in comparison to our worst-case scenario, other inputs of the same size sorted and found the median more quickly. Thus, the results that were seen supported our expectations.

We anticipated that for Input #2, which is sorted in descending order, the best-case scenario would occur with Max Heap and Quick Select using Median-of-Medians Pivot Selection. After, reviewing the table data, it's clear that other inputs of the same size took longer to sort and find the median compared to the worst-case scenario we selected. Therefore, our expectations were indeed fulfilled.

For Inputs #3-#30, we employed random lists to simulate common situations in which the data isn't ordered. To see how the algorithms work on average, we also changed the array size for a few inputs. This helped us determine which approach is most effective overall and which is more appropriate for certain scenarios.

The results provide a firm understanding of algorithm behaviour across a range of contexts, closely matching theoretical assumptions. Based on input characteristics and projected performance, this systematic analysis provides insightful information on the

strengths and weaknesses of the algorithm, enabling well-informed decision-making for real-world applications.

Work Division

Havva Nur Ozen: Quick sort and quick select algorithm with pivot as the first element codes.

Ezgi Eren: Insertion sort and quick select algorithm with median of median pivot selection codes.

Ikranur Akan: Merge sort and quick select algorithm with median of three pivot selection.

Remaining Work: Max heap and the report were made together.