Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Lab Report
On

“Sorting”

[Code No.: COMP 202]

Submitted by

Hridayanshu Raj Acharya (1)

Sankalp Acharya (2)

UNG-CE (II/I)

Submitted to

Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

Submission Date: 2024/06/16

Lab Report 4

Sorting

Sorting is a fundamental operation in computer science, where the elements of a list or array are arranged in a specific order, typically in ascending or descending order. Efficient sorting algorithms are crucial for optimizing the performance of various applications, including search algorithms, database query processing, and more.

Two commonly used sorting algorithms are **Insertion Sort** and **Quick Sort**. Each has its strengths and weaknesses, and they are chosen based on the specific requirements of the task at hand.

Insertion Sort

Insertion Sort is a simple and intuitive sorting algorithm that builds the final sorted array one item at a time. It is much like the way you might sort playing cards in your hands.
General idea:-

- **Start** with the second element (element at index 1) as the key.
- **Compare** the key with elements in the sorted part of the array (to its left).
- **Shift** all elements greater than the key to the right.
- **Insert** the key at the correct position.
- **Repeat** steps 2-4 for all elements in the array.

Quick Sort

Quick Sort is a highly efficient sorting algorithm that uses the divide-and-conquer strategy. It works by selecting a 'pivot' element and partitioning the array into two halves such that elements less than the pivot are on the left, and elements greater than the pivot are on the right. The process is then recursively applied to both halves.
General idea:-

- **Choose** a pivot element from the array.
- **Partition** the array such that elements less than the pivot are on the left, and elements greater than the pivot are on the right.
- **Recursively** apply the above steps to the left and right sub-arrays.
- **Combine** the sub-arrays to get the sorted array.

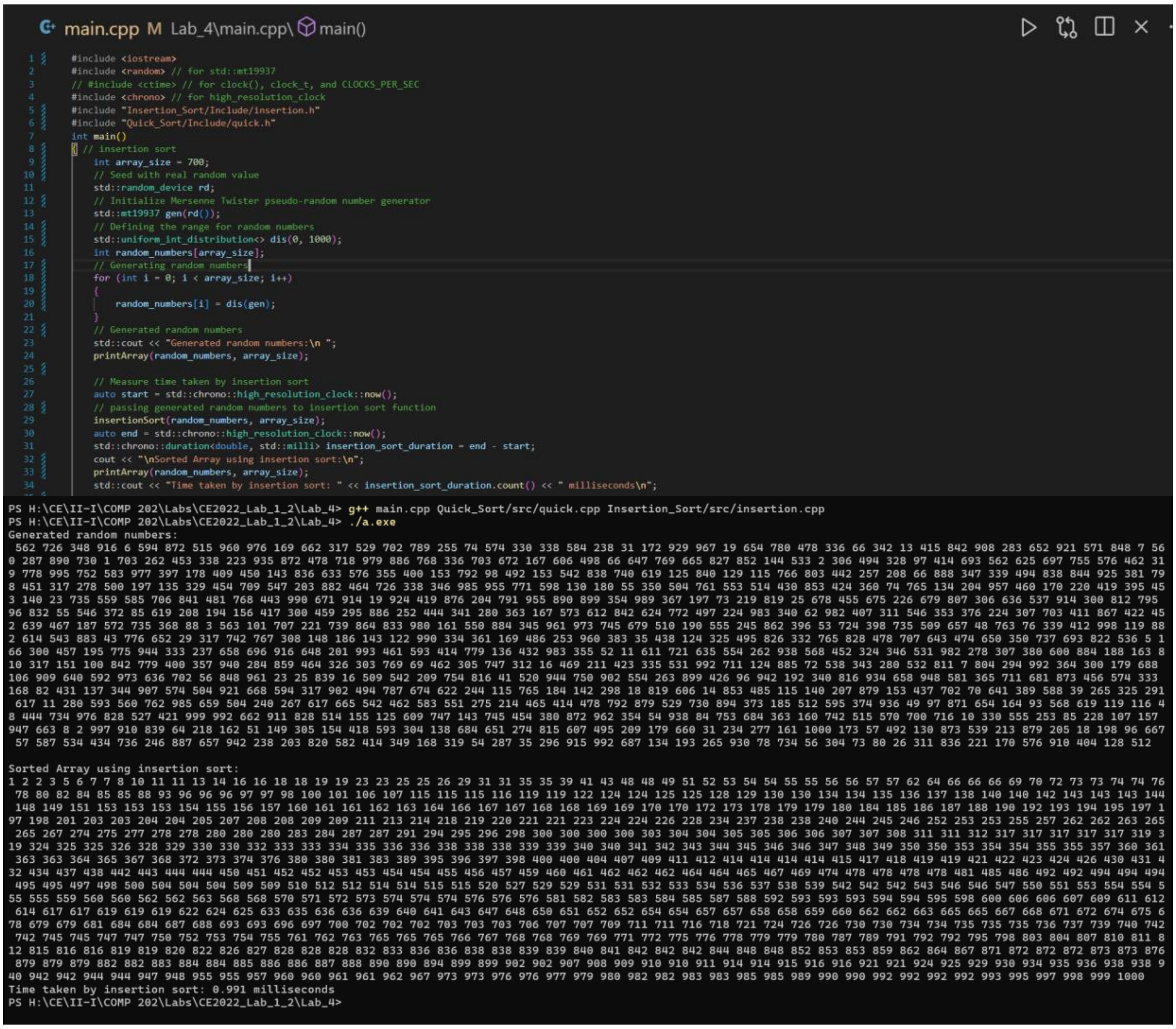
Code

Github Link: https://github.com/hridayanshu236/CE2022_Lab_1_2

Outputs:

For randomly generated datas:

-Insertion Sort:



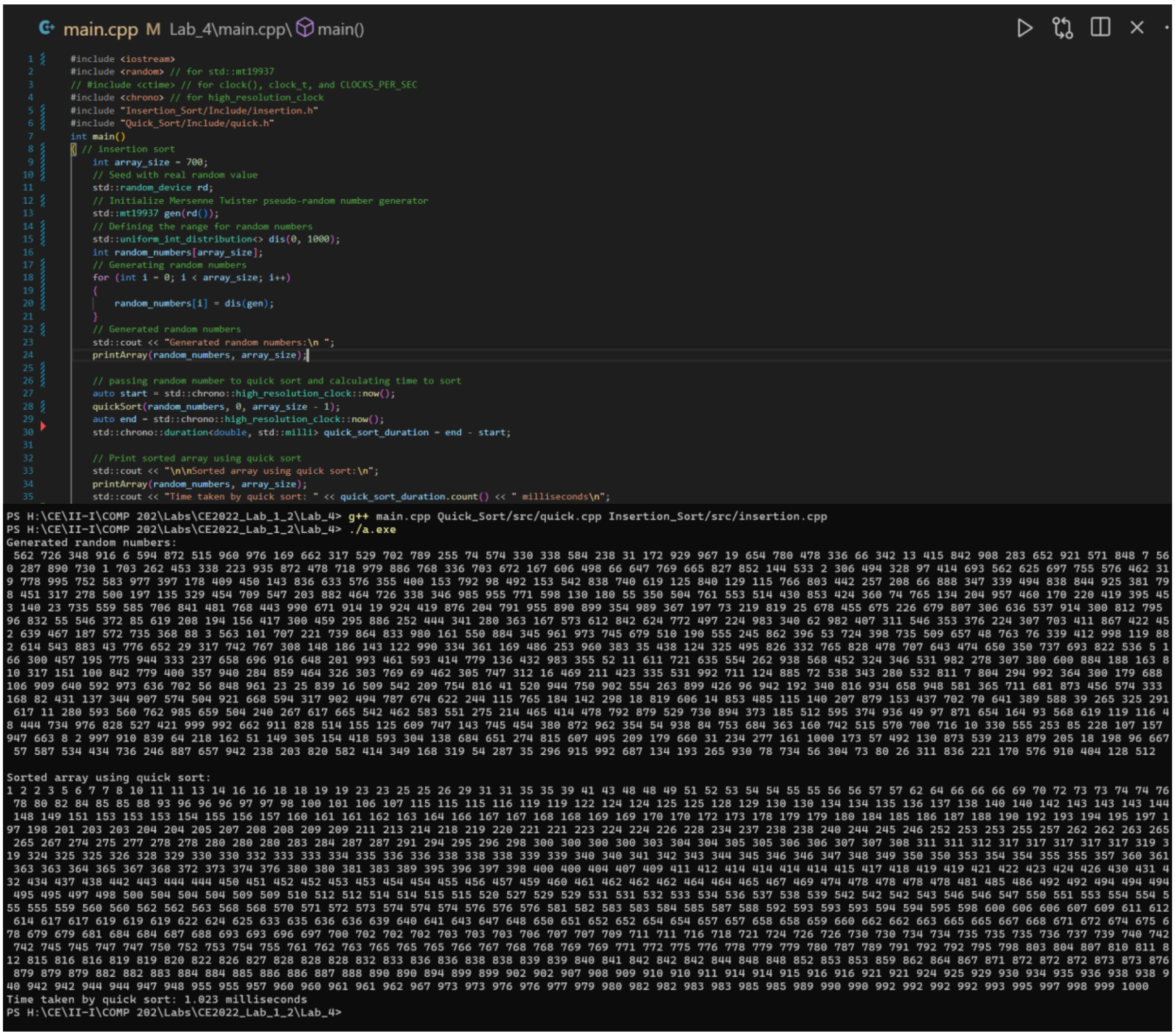
```cpp
#include <iostream>
#include <random> // for std::mt19937
// #include <ctime> // for clock(), clock_t, and CLOCKS_PER_SEC
#include <chrono> // for high_resolution_clock
#include "Insertion_Sort/Include/insertion.h"
#include "Quick_Sort/Include/quick.h"
int main()
{ // insertion sort
    int array_size = 700;
    // Seed with real random value
    std::random_device rd;
    // Initialize Mersenne Twister pseudo-random number generator
    std::mt19937 gen(rd());
    // Defining the range for random numbers
    std::uniform_int_distribution<> dis(0, 1000);
    int random_numbers[array_size];
    // Generating random numbers
    for (int i = 0; i < array_size; i++)
    {
        random_numbers[i] = dis(gen);
    }
    // Generated random numbers
    std::cout << "Generated random numbers:\n ";
    printArray(random_numbers, array_size);

    // Measure time taken by insertion sort
    auto start = std::chrono::high_resolution_clock::now();
    // passing generated random numbers to insertion sort function
    insertionSort(random_numbers, array_size);
    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double, std::milli> insertion_sort_duration = end - start;
    cout << "\nSorted Array using insertion sort:\n";
    printArray(random_numbers, array_size);
    std::cout << "Time taken by insertion sort: " << insertion_sort_duration.count() << " milliseconds\n";
```

```
PS H:\CE\II-I\COMP 202\Labs\CE2022_Lab_1_2\Lab_4> g++ main.cpp Quick_Sort/src/quick.cpp Insertion_Sort/src/insertion.cpp
PS H:\CE\II-I\COMP 202\Labs\CE2022_Lab_1_2\Lab_4> ./a.exe
Generated random numbers:
 562 726 348 916 6 594 872 515 960 976 169 662 317 529 702 789 255 74 574 330 338 584 238 31 172 929 967 19 654 780 478 336 66 342 13 415 842 908 283 652 921 571 848 7 56
0 287 890 730 1 703 262 453 338 223 935 872 478 718 979 886 768 336 703 672 167 606 498 66 647 769 665 827 852 144 533 2 306 494 328 97 414 693 562 625 697 755 576 462 31
9 778 995 752 583 977 397 178 409 450 143 836 633 576 355 400 153 792 98 492 153 542 838 740 619 125 840 129 115 766 803 442 257 208 66 888 347 339 494 838 844 925 381 79
8 451 317 278 500 197 135 329 454 709 547 203 882 464 726 338 346 985 955 771 598 130 180 55 350 504 761 553 514 430 853 424 360 74 765 134 204 957 460 170 220 419 395 45
3 140 23 735 559 585 706 841 481 768 443 990 671 914 19 924 419 876 204 791 955 890 899 354 989 367 197 73 219 819 25 678 455 675 226 679 807 306 636 537 914 300 812 795
96 832 55 546 372 85 619 208 194 156 417 300 459 295 886 252 444 341 280 363 167 573 612 842 624 772 497 224 983 340 62 982 407 311 546 353 376 224 307 703 411 867 422 45
2 639 467 187 572 735 368 88 3 563 101 707 221 739 864 833 980 161 550 884 345 961 973 745 679 510 190 555 245 862 396 53 724 398 735 509 657 48 763 76 339 412 998 119 88
2 614 543 883 43 776 652 29 317 742 767 308 148 186 143 122 990 334 361 169 486 253 960 383 35 438 124 325 495 826 332 765 828 478 707 643 474 650 350 737 693 822 536 5 1
66 300 457 195 775 944 333 237 658 696 916 648 201 993 461 593 414 779 136 432 983 355 52 11 611 721 635 554 262 938 568 452 324 346 531 982 278 307 380 600 884 188 163 8
10 317 151 100 842 779 400 357 940 284 859 464 326 303 769 69 462 305 747 312 16 469 211 423 335 531 992 711 124 885 72 538 343 280 532 811 7 804 294 992 364 300 179 688
106 909 640 592 973 636 702 56 848 961 23 25 839 16 509 542 209 754 816 41 520 944 750 902 554 263 899 426 96 942 192 340 816 934 658 948 581 365 711 681 873 456 574 333
168 82 431 137 344 907 574 504 921 668 594 317 902 494 787 674 622 244 115 765 184 142 298 18 819 606 14 853 485 115 140 207 879 153 437 702 70 641 389 588 39 265 325 291
 617 11 280 593 560 762 985 659 504 240 267 617 665 542 462 583 551 275 214 465 414 478 792 879 529 730 894 373 185 512 595 374 936 49 97 871 654 164 93 568 619 119 116 4
8 444 734 976 828 527 421 999 992 662 911 828 514 155 125 609 747 143 745 454 380 872 962 354 54 938 84 753 684 363 160 742 515 570 700 716 10 330 555 253 85 228 107 157
947 663 8 2 997 910 839 64 218 162 51 149 305 154 418 593 304 138 684 651 274 815 607 495 209 179 660 31 234 277 161 1000 173 57 492 130 873 539 213 879 205 18 198 96 667
 57 587 534 434 736 246 887 657 942 238 203 820 582 414 349 168 319 54 287 35 296 915 992 687 134 193 265 930 78 734 56 304 73 80 26 311 836 221 170 576 910 404 128 512

Sorted Array using insertion sort:
1 2 2 3 5 6 7 7 8 10 11 11 13 14 16 16 18 18 19 19 23 23 25 25 26 29 31 31 35 35 39 41 43 48 48 49 51 52 53 54 54 55 55 56 56 57 57 62 64 66 66 66 69 70 72 73 73 74 74 76
 78 80 82 84 85 85 88 93 96 96 96 97 97 98 100 101 106 107 115 115 115 116 119 119 122 124 124 125 125 128 129 130 130 134 134 135 136 137 138 140 140 142 143 143 143 144
 148 149 151 153 153 153 154 155 156 157 160 161 161 162 163 164 166 167 167 168 168 169 169 170 170 172 173 178 179 179 180 184 185 186 187 188 190 192 193 194 195 197 1
97 198 201 203 203 204 204 205 207 208 208 209 209 211 213 214 218 219 220 221 221 223 224 224 226 228 234 237 238 238 240 244 245 246 252 253 253 255 257 262 262 263 265
 265 267 274 275 277 278 278 280 280 280 283 284 287 287 291 294 295 296 298 300 300 300 300 303 304 304 305 305 306 306 307 307 308 311 311 312 317 317 317 317 317 319 3
19 324 325 325 326 328 329 330 330 332 333 333 334 335 336 336 338 338 338 339 339 340 340 341 342 343 344 345 346 346 347 348 349 350 350 353 354 354 355 355 357 360 361
 363 363 364 365 367 368 372 373 374 376 380 380 381 383 389 395 396 397 398 400 400 404 407 409 411 412 414 414 414 414 415 417 418 419 419 421 422 423 424 426 430 431 4
32 434 437 438 442 443 444 444 450 451 452 452 453 453 454 454 455 456 457 459 460 461 462 462 462 464 464 465 467 469 474 478 478 478 478 481 485 486 492 492 494 494 494
 495 495 497 498 500 504 504 504 509 509 510 512 512 514 514 515 515 520 527 529 529 531 531 532 533 534 536 537 538 539 542 542 542 543 546 546 547 550 551 553 554 554 5
55 555 559 560 560 562 562 563 568 568 570 571 572 573 574 574 574 576 576 576 581 582 583 583 584 585 587 588 592 593 593 593 594 594 595 598 600 606 606 607 609 611 612
 614 617 617 619 619 619 622 624 625 633 635 636 636 639 640 641 643 647 648 650 651 652 652 654 654 657 657 658 658 659 660 662 662 663 665 665 667 668 671 672 674 675 6
78 679 679 681 684 684 687 688 693 693 696 697 700 702 702 702 703 703 703 706 707 707 709 711 711 716 718 721 724 726 726 730 730 734 734 735 735 735 736 737 739 740 742
 742 745 745 747 747 750 752 753 754 755 761 762 763 765 765 765 766 767 768 768 769 769 771 772 775 776 778 779 779 780 787 789 791 792 792 795 798 803 804 807 810 811 8
12 815 816 816 819 819 820 822 826 827 828 828 828 832 833 836 836 838 838 839 839 840 841 842 842 842 844 848 848 852 853 853 859 862 864 867 871 872 872 872 873 873 876
 879 879 879 882 882 883 884 884 885 886 886 887 888 890 890 894 899 899 902 902 907 908 909 910 910 911 914 914 915 916 916 921 921 924 925 929 930 934 935 936 938 938 9
40 942 942 944 944 947 948 955 955 957 960 960 961 961 962 967 973 973 976 976 977 979 980 982 982 983 983 985 985 989 990 990 992 992 992 992 993 995 997 998 999 1000
Time taken by insertion sort: 0.991 milliseconds
PS H:\CE\II-I\COMP 202\Labs\CE2022_Lab_1_2\Lab_4>
```

Time Complexities:-

- Worst-case: O(n^2).
- Best-case: O(n) (when the array is already sorted).
- Average-case: O(n^2).

-Quick Sort



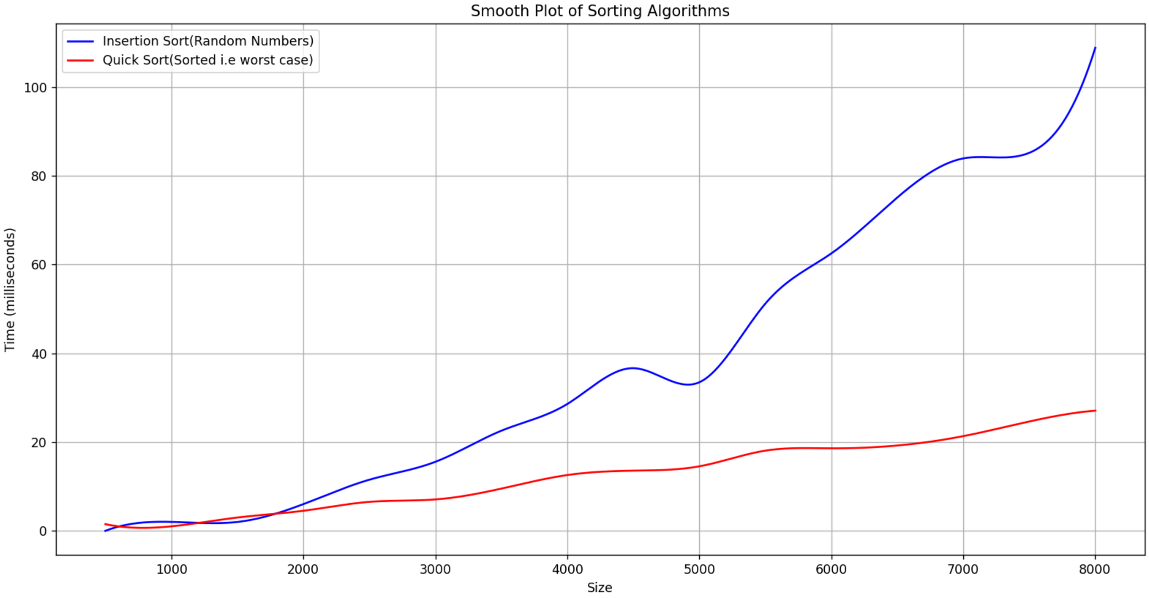
```cpp
#include <iostream>
#include <random> // for std::mt19937
// #include <ctime> // for clock(), clock_t, and CLOCKS_PER_SEC
#include <chrono> // for high_resolution_clock
#include "Insertion_Sort/Include/insertion.h"
#include "Quick_Sort/Include/quick.h"
int main()
{ // insertion sort
    int array_size = 700;
    // Seed with real random value
    std::random_device rd;
    // Initialize Mersenne Twister pseudo-random number generator
    std::mt19937 gen(rd());
    // Defining the range for random numbers
    std::uniform_int_distribution<> dis(0, 1000);
    int random_numbers[array_size];
    // Generating random numbers
    for (int i = 0; i < array_size; i++)
    {
        random_numbers[i] = dis(gen);
    }
    // Generated random numbers
    std::cout << "Generated random numbers:\n ";
    printArray(random_numbers, array_size);

    // passing random number to quick sort and calculating time to sort
    auto start = std::chrono::high_resolution_clock::now();
    quickSort(random_numbers, 0, array_size - 1);
    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double, std::milli> quick_sort_duration = end - start;

    // Print sorted array using quick sort
    std::cout << "\n\nSorted array using quick sort:\n";
    printArray(random_numbers, array_size);
    std::cout << "Time taken by quick sort: " << quick_sort_duration.count() << " milliseconds\n";
```

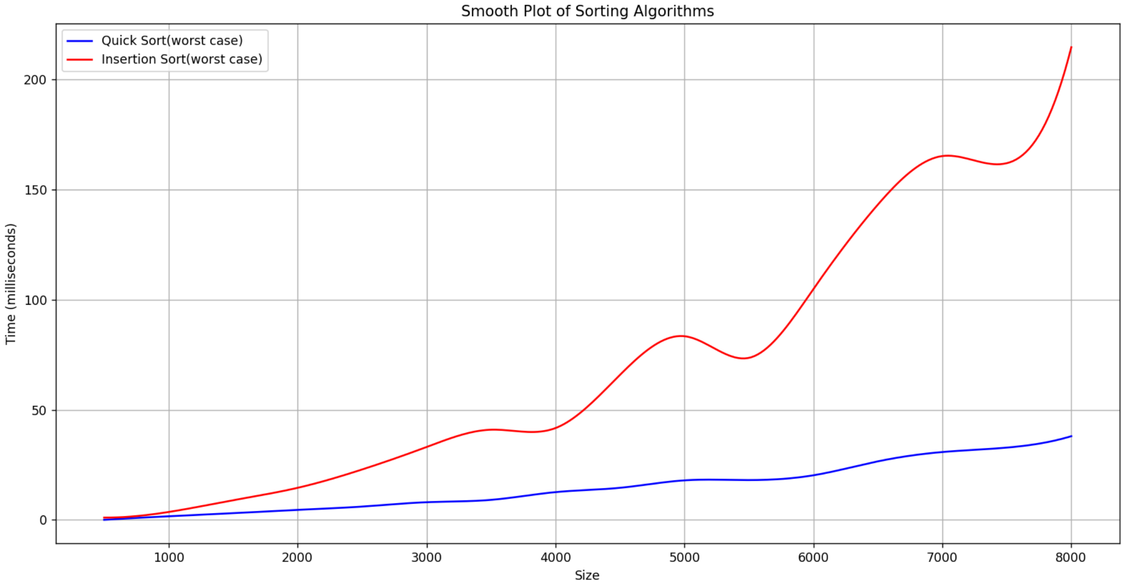
```
PS H:\CE\II-I\COMP 202\Labs\CE2022_Lab_1_2\Lab_4> g++ main.cpp Quick_Sort/src/quick.cpp Insertion_Sort/src/insertion.cpp
PS H:\CE\II-I\COMP 202\Labs\CE2022_Lab_1_2\Lab_4> ./a.exe
Generated random numbers:
 562 726 348 916 6 594 872 515 960 976 169 662 317 529 702 789 255 74 574 330 338 584 238 31 172 929 967 19 654 780 478 336 66 342 13 415 842 908 283 652 921 571 848 7 56
0 287 890 730 1 703 262 453 338 223 935 872 478 718 979 886 768 336 703 672 167 606 498 66 647 769 665 827 852 144 533 2 306 494 328 97 414 693 562 625 697 755 576 462 31
9 778 995 752 583 977 397 178 409 450 143 836 633 576 355 400 153 792 98 492 153 542 838 740 619 125 840 129 115 766 803 442 257 208 66 888 347 339 494 838 844 925 381 79
8 451 317 278 500 197 135 329 454 709 547 203 882 464 726 338 346 985 955 771 598 130 180 55 350 504 761 553 514 430 853 424 360 74 765 134 204 957 460 170 220 419 395 45
3 140 23 735 559 585 706 841 481 768 443 990 671 914 19 924 419 876 204 791 955 890 899 354 989 367 197 73 219 819 25 678 455 675 226 679 807 306 636 537 914 300 812 795
96 832 55 546 372 85 619 208 194 156 417 300 459 295 886 252 444 341 280 363 167 573 612 842 624 772 497 224 983 340 62 982 407 311 546 353 376 224 307 703 411 867 422 45
2 639 467 187 572 735 368 88 3 563 101 707 221 739 864 833 980 161 550 884 345 961 973 745 679 510 190 555 245 862 396 53 724 398 735 509 657 48 763 76 339 412 998 119 88
2 614 543 883 43 776 652 29 317 742 767 308 148 186 143 122 990 334 361 169 486 253 960 383 35 438 124 325 495 826 332 765 828 478 707 643 474 650 350 737 693 822 536 5 1
66 300 457 195 775 944 333 237 658 696 916 648 201 993 461 593 414 779 136 432 983 355 52 11 611 721 635 554 262 938 568 452 324 346 531 982 278 307 380 600 884 188 163 8
10 317 151 100 842 779 400 357 940 284 859 464 326 303 769 69 462 305 747 312 16 469 211 423 335 531 992 711 124 885 72 538 343 280 532 811 7 804 294 992 364 300 179 688
106 909 640 592 973 636 702 56 848 961 23 25 839 16 509 542 209 754 816 41 520 944 750 902 554 263 899 426 96 942 192 340 816 934 658 948 581 365 711 681 873 456 574 333
168 82 431 137 344 907 574 504 921 668 594 317 902 494 787 674 622 244 115 765 184 142 298 18 819 606 14 853 485 115 140 207 879 153 437 702 70 641 389 588 39 265 325 291
 617 11 280 593 560 762 985 659 504 240 267 617 665 542 462 583 551 275 214 465 414 478 792 879 529 730 894 373 185 512 595 374 936 49 97 871 654 164 93 568 619 119 116 4
8 444 734 976 828 527 421 999 992 662 911 828 514 155 125 609 747 143 745 454 380 872 962 354 54 938 84 753 684 363 160 742 515 570 700 716 10 330 555 253 85 228 107 157
947 663 8 2 997 910 839 64 218 162 51 149 305 154 418 593 304 138 684 651 274 815 607 495 209 179 660 31 234 277 161 1000 173 57 492 130 873 539 213 879 205 18 198 96 667
 57 587 534 434 736 246 887 657 942 238 203 820 582 414 349 168 319 54 287 35 296 915 992 687 134 193 265 930 78 734 56 304 73 80 26 311 836 221 170 576 910 404 128 512

Sorted array using quick sort:
1 2 2 3 5 6 7 7 8 10 11 11 13 14 16 16 18 18 19 19 23 23 25 25 26 29 31 31 35 35 39 41 43 48 48 49 51 52 53 54 54 55 55 56 56 57 57 62 64 66 66 66 69 70 72 73 73 74 74 76
 78 80 82 84 85 85 88 93 96 96 96 97 97 98 100 101 106 107 115 115 115 116 119 119 122 124 124 125 125 128 129 130 130 134 134 135 136 137 138 140 140 142 143 143 143 144
 148 149 151 153 153 153 154 155 156 157 160 161 161 162 163 164 166 167 167 168 168 169 169 170 170 172 173 178 179 179 180 184 185 186 187 188 190 192 193 194 195 197 1
97 198 201 203 203 204 204 205 207 208 208 209 209 211 213 214 218 219 220 221 221 223 224 224 226 228 234 237 238 238 240 244 245 246 252 253 253 255 257 262 262 263 265
 265 267 274 275 277 278 278 280 280 280 283 284 287 287 291 294 295 296 298 300 300 300 303 304 304 305 305 306 306 307 307 308 311 311 312 317 317 317 317 317 319 3
19 324 325 325 326 328 329 330 330 332 333 333 334 335 336 336 338 338 338 339 339 340 340 341 342 343 344 345 346 346 347 348 349 350 350 353 354 354 355 355 357 360 361
 363 363 364 365 367 368 372 373 374 376 380 380 381 383 389 395 396 397 398 400 400 404 407 409 411 412 414 414 414 414 415 417 418 419 419 421 422 423 424 426 430 431 4
32 434 437 438 442 443 444 444 450 451 452 452 453 453 454 454 455 456 457 459 460 461 462 462 462 464 464 465 467 469 474 478 478 478 478 481 485 486 492 492 494 494 494
 495 495 497 498 500 504 504 504 509 509 510 512 512 514 514 515 515 520 527 529 529 531 531 532 533 534 536 537 538 539 542 542 542 543 546 546 547 550 551 553 554 554 5
55 555 559 560 560 562 562 563 568 568 570 571 572 573 574 574 574 576 576 576 581 582 583 583 584 585 587 588 592 593 593 593 594 594 595 598 600 606 606 607 609 611 612
 614 617 617 619 619 619 622 624 625 633 635 636 636 639 640 641 643 647 648 650 651 652 652 654 654 657 657 658 658 659 660 662 662 663 665 665 667 668 671 672 674 675 6
78 679 679 681 684 684 687 688 693 693 696 697 700 702 702 702 703 703 703 706 707 707 709 711 711 716 718 721 724 726 726 730 730 734 734 735 735 735 736 737 739 740 742
 742 745 745 747 747 750 752 753 754 755 761 762 763 765 765 765 766 767 768 768 769 769 771 772 775 776 778 779 779 780 787 789 791 792 792 795 798 803 804 807 810 811 8
12 815 816 816 819 819 820 822 826 827 828 828 828 832 833 836 836 838 838 839 839 840 841 842 842 842 844 848 848 852 853 853 859 862 864 867 871 872 872 872 873 873 876
 879 879 879 882 882 883 884 884 885 886 886 887 888 890 890 894 899 899 902 902 907 908 909 910 910 911 914 914 915 916 916 921 921 924 925 929 930 934 935 936 938 938 9
40 942 942 944 944 947 948 955 955 957 960 960 961 961 962 967 973 973 976 976 977 979 980 982 982 983 983 985 985 989 990 990 992 992 992 992 993 995 997 998 999 1000
Time taken by quick sort: 1.023 milliseconds
PS H:\CE\II-I\COMP 202\Labs\CE2022_Lab_1_2\Lab_4>
```

Time Complexities

- Worst-case: O(n^2) (when the smallest or largest element is always chosen as the pivot)
- Best-case: O(n log n)
- Average-case: O(n log n)

Graphs of Size of Data vs Time:-





Conclusion:

Both Insertion Sort and Quick Sort have their unique advantages and are suited for different scenarios. Insertion Sort is ideal for small datasets and arrays that are already nearly sorted. Quick Sort, with its average-case O(n log n) complexity, is one of the fastest sorting algorithms for larger datasets but requires careful pivot selection to avoid worst-case performance. Understanding these algorithms and their characteristics is essential for choosing the appropriate sorting technique based on the specific requirements of the task.