National University of Computer & Emerging Sciences Karachi Campus

Comparison between Process and Threads

Project Report
Operating System
Section: BSCS-4K

GROUP MEMBERS

Aisha Jalil 22K-4649

Aniqa Azhar 22K-4228

Objective:

The primary objective of this project is to compare the performance of sorting algorithms when implemented as processes versus threads. Specifically, we aim to achieve the following:

1. Implement five sorting algorithms (Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, and Quick Sort) as processes and threads.
2. Measure and analyze the execution time of each sorting algorithm when implemented as processes and threads.
3. Compare the execution time and efficiency of process-based versus thread-based implementations for each sorting algorithm.
4. Present the results through graphical representations for better visualization and understanding.

Project Description:

In modern computing, parallelism plays a crucial role in enhancing the performance of various algorithms and tasks. Processes and threads are two fundamental units of parallel execution in operating systems. Processes represent independent, isolated units of execution, each with its own memory space, while threads are lighter-weight units that share the same memory space within a process.

In this project, we implemented five well-known sorting algorithms: Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, and Quick Sort. Each algorithm was implemented both as processes and as threads using POSIX threading (pthread library). We used large datasets consisting of random numbers to test the performance of these algorithms. The datasets were provided through files. Datasets included 1000, 10000, 10000, 100000, and 200000 random numbers generated between 1 to 100.

The implementation involved creating functions for each sorting algorithm, as well as additional functions for thread creation (separate functions were created for 2 threaded processes and 4 threaded processes). We carefully designed the code to ensure that the sorting algorithms functioned correctly in both process and thread environments.

After implementing the algorithms, we conducted experiments to measure the execution time of each algorithm when implemented as processes and threads. The execution time was measured using the clock() function from the C++ standard library.

Code:

```
#include <iostream>

#include <pthread.h>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <fstream>


#define THREADS_2 2

#define THREADS_4 4


using namespace std;
```

```
struct ThreadArgs {
        int* arr;
        int low;
        int high;
};

void insertionSortProcess(int* arr, int ARRAY_SIZE)
{
        int i, key, j;
        for (i = 1; i < ARRAY_SIZE; i++) {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key) {
        arr[j + 1] = arr[j];
        j = j - 1;
        }
        arr[j + 1] = key;
        }
}


void* insertionSort_for_Threads(void* args) {
        ThreadArgs* tArgs = (ThreadArgs*)args;
```

```
    int start = tArgs->low;

    int end = tArgs->high;

    int* arr = tArgs->arr;



    for (int i = start + 1; i <= end; ++i) {

    int key = arr[i];

    int j = i - 1;


    while (j >= start && arr[j] > key) {

    arr[j + 1] = arr[j];

    j = j - 1;

    }

    arr[j + 1] = key;

    }


    pthread_exit(NULL);

}


void selectionSortProcess(int* arr, int ARRAY_SIZE)

{

    int i, j, min_idx;


    for (i = 0; i < ARRAY_SIZE - 1; i++) {
    min_idx = i;
    for (j = i + 1; j < ARRAY_SIZE; j++) {
    if (arr[j] < arr[min_idx])
    min_idx = j;
    }

    if (min_idx != i)
    swap(arr[min_idx], arr[i]);
    }
}

void* selectionSort_for_Threads(void* args) {
    ThreadArgs* tArgs = (ThreadArgs*)args;

    int start = tArgs->low;
    int end = tArgs->high;
    int* arr = tArgs->arr;

    int i, j, min_idx;

    for (i = start; i < end - 1; i++) {
    min_idx = i;
    for (j = i + 1; j < end; j++) {
    if (arr[j] < arr[min_idx])
    min_idx = j;
    }
    if (min_idx != i)
    swap(arr[min_idx], arr[i]);
    }

    pthread_exit(NULL);
}


void bubbleSortProcess(int *arr, int ARRAY_SIZE)
{
    int i, j;
    bool swapped;
    for (i = 0; i < ARRAY_SIZE - 1; i++) {
    swapped = false;
    for (j = 0; j < ARRAY_SIZE - i - 1; j++) {
    if (arr[j] > arr[j + 1]) {
    swap(arr[j], arr[j + 1]);
    swapped = true;
    }
    }
    if (swapped == false)
    break;
    }
}
```

```
void* bubbleSort_for_Threads(void* args) {
    ThreadArgs* tArgs = (ThreadArgs*)args;


    int start = tArgs->low;
    int end = tArgs->high;
    int* arr = tArgs->arr;


    bool swapped;


    for (int i = start; i < end - 1; i++) {
    swapped = false;
    for (int j = start; j < end - i -1 + start; j++) { //adding start for correct indexing
    if (arr[j] > arr[j + 1]) {
    swap(arr[j], arr[j + 1]);
    swapped = true;
    }
    }


    if (swapped == false)
    break;
    }


    pthread_exit(NULL);
}
```

```
void merge(int low, int mid, int high, int* arr) {
	int* L = new int[mid - low + 1];
	int* R = new int[high - mid];

	int size_1 = mid - low + 1;
	int size_2 = high - mid;

	for (int i = 0; i < size_1; i++)
	L[i] = arr[i + low];

	for (int i = 0; i < size_2; i++)
	R[i] = arr[i + mid + 1];

	int k = low;
	int i = 0, j = 0;

	while (i < size_1 && j < size_2) {
	if (L[i] <= R[j])
	arr[k++] = L[i++];
	else
	arr[k++] = R[j++];
	}

	while (i < size_1)
	arr[k++] = L[i++];
```

```
    while (j < size_2)
    arr[k++] = R[j++];

    delete [] L;
    delete [] R;
}

void mergeSort(int low, int high, int* arr) {
    int mid = low + (high - low) / 2;
    if (low < high) {
    mergeSort(low, mid, arr);
    mergeSort(mid + 1, high, arr);
    merge(low, mid, high, arr);
    }
}

int partition(int low, int high, int* arr) {
    int pivot = arr[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
    if (arr[j] < pivot) {
    i++;
    swap(arr[i], arr[j]);
    }
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
}

void quickSort(int low, int high, int* arr) {
    if (low < high) {
    int pi = partition(low, high, arr);

    quickSort(low, pi - 1, arr);
    quickSort(pi + 1, high, arr);
    }
}

void* quickSort_for_Threads(void* arg) {
    ThreadArgs* tArgs = (ThreadArgs*)arg;
    int low = tArgs->low;
    int high = tArgs->high;
    int* arr = tArgs->arr;


    if (low < high) {
    int pi = partition(low, high, arr);

    quickSort(low, pi - 1, arr);
    quickSort(pi + 1, high, arr);
    }

    pthread_exit(NULL);

}

void* mergeSort_for_Threads(void* arg) {
    ThreadArgs* tArgs = (ThreadArgs*)arg;
    int low = tArgs->low;
    int high = tArgs->high;
    int* arr = tArgs->arr;


    int mid = low + (high - low) / 2;
    if (low < high) {
    mergeSort(low, mid, arr);
    mergeSort(mid + 1, high, arr);
    merge(low, mid, high, arr);
    }

    pthread_exit(NULL);
}


void reset_array(int* og_arr, int* arr, int ARRAY_SIZE){
    for(int i=0; i<ARRAY_SIZE; i++){
     arr[i] = og_arr[i];

    }

}


// Main function

int main() {


    vector <int> arr;

    vector <int> og_arr;

    int ARRAY_SIZE;


    string filename;


    cout<<"Enter the name of file you want elements from: ";

    cin>>filename;


    ifstream file(filename);


    if (!file)

    {

    cerr << "Failed to open file" << endl;

    return -1;

    }


    int element;
```

```
if (file.is_open()) {
while (file >> element) {
arr.push_back(element);
}
}

file.close();

ARRAY_SIZE = arr.size();

//saving original array read from file in og_arr

for(int i=0; i<arr.size(); i++){
 og_arr.push_back(arr[i]);
}


clock_t startTime, endTime;




//PROCESSES

//bubble sort
```

```
        cout << "Unsorted array: ";
        for (int i = 0; i < ARRAY_SIZE; ++i) {
        cout << arr[i] << " ";
        }

        cout<<endl<<endl;

        startTime = clock();

        bubbleSortProcess(arr.data(), ARRAY_SIZE);

        endTime = clock();

        cout << "Sorted array: ";
        for (int i = 0; i < ARRAY_SIZE; ++i) {
        cout << arr[i] << " ";
        }
        cout << endl<<endl;

        cout << "Time taken for bubble sort process: " << (double)(endTime - startTime) / (double)CLOCKS_PER_SEC << " seconds" << endl<<endl;


        //insertion sort

        reset_array(og_arr.data(), arr.data(), ARRAY_SIZE);
        cout << "Unsorted array: ";
        for (int i = 0; i < ARRAY_SIZE; ++i) {
        cout << arr[i] << " ";
        }


        cout<<endl<<endl;


        startTime = clock();


        insertionSortProcess(arr.data(), ARRAY_SIZE);


        endTime = clock();


        cout << "Sorted array: ";
        for (int i = 0; i < ARRAY_SIZE; ++i) {
        cout << arr[i] << " ";
        }
        cout << endl<<endl;


        cout << "Time taken for insertion sort process: " << (double)(endTime - startTime) /
(double)CLOCKS_PER_SEC << " seconds" << endl<<endl;




        //selection sort
        reset_array(og_arr.data(), arr.data(), ARRAY_SIZE);


        cout << "Unsorted array: ";

        for (int i = 0; i < ARRAY_SIZE; ++i) {

        cout << arr[i] << " ";

        }


        cout<<endl<<endl;


        startTime = clock();


        selectionSortProcess(arr.data(), ARRAY_SIZE);


        endTime = clock();


        cout << "Sorted array: ";

        for (int i = 0; i < ARRAY_SIZE; ++i) {

        cout << arr[i] << " ";

        }

        cout << endl<<endl;


        cout << "Time taken for selection sort process: " << (double)(endTime - startTime) /
(double)CLOCKS_PER_SEC << " seconds" << endl<<endl;


        //merge sort
```

```
	reset_array(og_arr.data(), arr.data(), ARRAY_SIZE);


	cout << "Unsorted array: ";

	for (int i = 0; i < ARRAY_SIZE; ++i) {

	cout << arr[i] << " ";

	}


	cout<<endl<<endl;


	startTime = clock();


	mergeSort(0, ARRAY_SIZE-1, arr.data());


	endTime = clock();


	cout << "Sorted array: ";

	for (int i = 0; i < ARRAY_SIZE; ++i) {

	cout << arr[i] << " ";

	}

	cout << endl<<endl;


	cout << "Time taken for merge sort process: " << (double)(endTime - startTime) /
(double)CLOCKS_PER_SEC << " seconds" << endl<<endl;


	//quick sort
```

```
        reset_array(og_arr.data(), arr.data(), ARRAY_SIZE);


        cout << "Unsorted array: ";

        for (int i = 0; i < ARRAY_SIZE; ++i) {

        cout << arr[i] << " ";

        }


        cout<<endl<<endl;


        startTime = clock();


        quickSort(0, ARRAY_SIZE-1, arr.data());


        endTime = clock();


        cout << "Sorted array: ";

        for (int i = 0; i < ARRAY_SIZE; ++i) {

        cout << arr[i] << " ";

        }

        cout << endl<<endl;


        cout << "Time taken for quick sort process: " << (double)(endTime - startTime) / (double)CLOCKS_PER_SEC << " seconds" << endl<<endl;
```

```
// 2 THREADED PROCESSES

pthread_t threads[THREADS_2];
ThreadArgs threadArgs[THREADS_2];
int segment_size = ARRAY_SIZE / THREADS_2;

//bubble sort with 2 threads

reset_array(og_arr.data(), arr.data(), ARRAY_SIZE);

cout << "Unsorted array: ";
for (int i = 0; i < ARRAY_SIZE; ++i) {
cout << arr[i] << " ";
}

cout<<endl<<endl;

startTime = clock();
```

```
        for (int i = 0; i < THREADS_2; i++) {

        threadArgs[i].arr = arr.data();

        threadArgs[i].low = i * segment_size;

        threadArgs[i].high = (i + 1) * segment_size;

        pthread_create(&threads[i], NULL, bubbleSort_for_Threads, (void*)&threadArgs[i]);

        }


        for (int i = 0; i < THREADS_2; ++i) {

        pthread_join(threads[i], NULL);

        }


        merge(0, (ARRAY_SIZE - 1) / 2, ARRAY_SIZE - 1, arr.data());


        endTime = clock();


        cout << "Sorted array: ";

        for (int i = 0; i < ARRAY_SIZE; ++i) {

        cout << arr[i] << " ";

        }

        cout << endl << endl;


        cout << "Time taken for bubble sort with 2 threads: " << (double)(endTime - startTime) /
(double)CLOCKS_PER_SEC << " seconds" << endl<<endl;
```

```
//insertion sort with 2 threads

reset_array(og_arr.data(), arr.data(), ARRAY_SIZE);

cout << "Unsorted array: ";
for (int i = 0; i < ARRAY_SIZE; ++i) {
cout << arr[i] << " ";
}
cout<<endl<<endl;

startTime = clock();

for (int i = 0; i < THREADS_2; i++) {
threadArgs[i].arr = arr.data();
threadArgs[i].low = i * segment_size;
threadArgs[i].high = (i + 1) * segment_size - 1;
pthread_create(&threads[i], NULL, insertionSort_for_Threads, (void*)&threadArgs[i]);
}

for (int i = 0; i < THREADS_2; ++i) {
pthread_join(threads[i], NULL);
}
```

```
	}



	merge(0, (ARRAY_SIZE - 1) / 2, ARRAY_SIZE - 1, arr.data());


	endTime = clock();


	cout << "Sorted array: ";

	for (int i = 0; i < ARRAY_SIZE; ++i) {

	cout << arr[i] << " ";

	}

	cout << endl <<endl;


	cout << "Time taken for insertion sort with 2 threads: " << (double)(endTime - startTime) /
(double)CLOCKS_PER_SEC << " seconds" << endl << endl;






	//selection sort with 2 threads


	reset_array(og_arr.data(), arr.data(), ARRAY_SIZE);


	cout << "Unsorted array: ";

	for (int i = 0; i < ARRAY_SIZE; ++i) {

	cout << arr[i] << " ";

	}
```

```
cout<<endl<<endl;


startTime = clock();

for (int i = 0; i < THREADS_2; i++) {
threadArgs[i].arr = arr.data();
threadArgs[i].low = i * segment_size;
threadArgs[i].high = (i + 1) * segment_size;
pthread_create(&threads[i], NULL, selectionSort_for_Threads, (void*)&threadArgs[i]);
}

for (int i = 0; i < THREADS_2; i++) {
pthread_join(threads[i], NULL);
}


merge(0, (ARRAY_SIZE - 1) / 2, ARRAY_SIZE - 1, arr.data());

endTime = clock();

cout << "Sorted array: ";
for (int i = 0; i < ARRAY_SIZE; i++)
cout << arr[i] << " ";
cout << endl;
```

```
	cout << "Time taken for selection sort with 2 threads: " << (double)(endTime - startTime) /
(double)CLOCKS_PER_SEC << " seconds" << endl << endl;



	//merge sort with 2 threads


	reset_array(og_arr.data(), arr.data(), ARRAY_SIZE);


	cout << "Unsorted array: ";

	for (int i = 0; i < ARRAY_SIZE; ++i) {

	cout << arr[i] << " ";

	}


	cout << endl << endl;


	startTime = clock();


	for (int i = 0; i < THREADS_2; i++) {

	threadArgs[i].arr = arr.data();

	threadArgs[i].low = i * segment_size;

	threadArgs[i].high = (i + 1) * segment_size - 1;

	pthread_create(&threads[i], NULL, mergeSort_for_Threads, (void*)&threadArgs[i]);
```

```
        }

        for (int i = 0; i < THREADS_2; i++)
        pthread_join(threads[i], NULL);

        merge(0, (ARRAY_SIZE - 1) / 2, ARRAY_SIZE - 1, arr.data());

        endTime = clock();

        cout << "Sorted array: ";
        for (int i = 0; i < ARRAY_SIZE; i++)
        cout << arr[i] << " ";
        cout << endl;

        cout << "Time taken for merge sort with 2 threads: " << (double)(endTime - startTime) /
(double)CLOCKS_PER_SEC << " seconds" << endl << endl;




        //quick sort with 2 threads

        reset_array(og_arr.data(), arr.data(), ARRAY_SIZE);

        cout << "Unsorted array: ";
```

```
for (int i = 0; i < ARRAY_SIZE; ++i) {
cout << arr[i] << " ";
}

cout<<endl<<endl;

startTime = clock();

for (int i = 0; i < THREADS_2; i++) {
threadArgs[i].arr = arr.data();
threadArgs[i].low = i * segment_size;
threadArgs[i].high = (i + 1) * segment_size - 1;
pthread_create(&threads[i], NULL, quickSort_for_Threads, (void*)&threadArgs[i]);
}

for (int i = 0; i < THREADS_2; i++)
pthread_join(threads[i], NULL);


merge(0, (ARRAY_SIZE - 1) / 2, ARRAY_SIZE - 1, arr.data());

endTime = clock();

cout << "Sorted array: ";
for (int i = 0; i < ARRAY_SIZE; i++)
cout << arr[i] << " ";
```

```
        cout << endl;

        cout << "Time taken for quick sort with 2 threads: " << (double)(endTime - startTime) /
CLOCKS_PER_SEC << " seconds" << endl<<endl;

        //PROCESSES WITH 4 THREADS

        pthread_t threads_4[THREADS_4];

        ThreadArgs args_4[THREADS_4];

        segment_size = ARRAY_SIZE / THREADS_4;

        //bubble sort with 4 threads

        reset_array(og_arr.data(), arr.data(), ARRAY_SIZE);

        cout << "Unsorted array: ";
```

```
for (int i = 0; i < ARRAY_SIZE; ++i) {
 cout << arr[i] << " ";
}

cout<<endl<<endl;

startTime = clock();

for (int i = 0; i < THREADS_4; i++) {
 args_4[i].arr = arr.data();
 args_4[i].low = i * segment_size;
 args_4[i].high = (i + 1) * segment_size;
 pthread_create(&threads_4[i], NULL, bubbleSort_for_Threads, (void*)&args_4[i]);
}

for (int i = 0; i < THREADS_4; ++i) {
 pthread_join(threads_4[i], NULL);
}

//mids calculated with start+(end-start)/2
merge(0, (ARRAY_SIZE / 2 - 1) / 2, ARRAY_SIZE / 2 - 1, arr.data());
merge(ARRAY_SIZE / 2, ARRAY_SIZE / 2 + (ARRAY_SIZE - 1 - ARRAY_SIZE / 2) / 2, ARRAY_SIZE - 1, arr.data());
merge(0, (ARRAY_SIZE - 1) / 2, ARRAY_SIZE - 1, arr.data());

endTime = clock();
```

```
    cout << "Sorted array: ";

    for (int i = 0; i < ARRAY_SIZE; ++i) {
     cout << arr[i] << " ";
    }
    cout << endl<<endl;

    cout << "Time taken for bubble sort with 4 threads: " << (double)(endTime - startTime) / (double)CLOCKS_PER_SEC << " seconds" << endl<<endl;



    //insertion sort with 4 threads

    reset_array(og_arr.data(), arr.data(), ARRAY_SIZE);

    cout << "Unsorted array: ";
    for (int i = 0; i < ARRAY_SIZE; ++i) {
     cout << arr[i] << " ";
    }
    cout<<endl<<endl;

    startTime = clock();
```

```
for (int i = 0; i < THREADS_4; i++) {
 args_4[i].arr = arr.data();
 args_4[i].low = i * segment_size;
 args_4[i].high = (i + 1) * segment_size-1;
 pthread_create(&threads_4[i], NULL, insertionSort_for_Threads, (void*)&args_4[i]);
}


for (int i = 0; i < THREADS_4; ++i) {
 pthread_join(threads_4[i], NULL);
}


merge(0, (ARRAY_SIZE / 2 - 1) / 2, ARRAY_SIZE / 2 - 1, arr.data());
merge(ARRAY_SIZE / 2, ARRAY_SIZE / 2 + (ARRAY_SIZE - 1 - ARRAY_SIZE / 2) / 2, ARRAY_SIZE - 1, arr.data());
merge(0, (ARRAY_SIZE - 1) / 2, ARRAY_SIZE - 1, arr.data());




endTime = clock();


cout << "Sorted array: ";
for (int i = 0; i < ARRAY_SIZE; ++i) {
 cout << arr[i] << " ";
}
cout << endl <<endl;
```

```
	cout << "Time taken for insertion sort with 4 threads: " << (double)(endTime - startTime) / (double)CLOCKS_PER_SEC << " seconds" << endl<<endl;




	//selection sort with 4 threads

	reset_array(og_arr.data(), arr.data(), ARRAY_SIZE);

	cout << "Unsorted array: ";
	for (int i = 0; i < ARRAY_SIZE; ++i) {
	 cout << arr[i] << " ";
	}

	cout<<endl<<endl;


	startTime = clock();

	for (int i = 0; i < THREADS_4; i++) {
	 args_4[i].arr = arr.data();
	 args_4[i].low = i * segment_size;
	 args_4[i].high = (i + 1) * segment_size;
	 pthread_create(&threads_4[i], NULL, selectionSort_for_Threads, (void*)&args_4[i]);
```

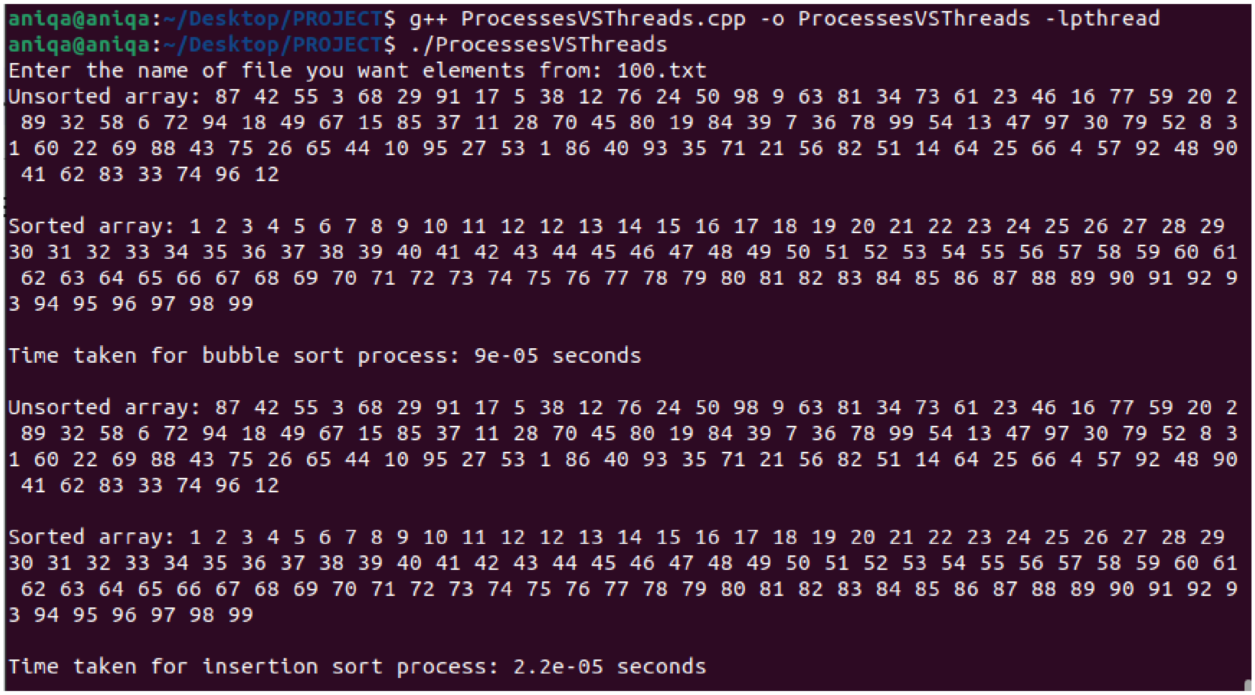
```
        }

        for (int i = 0; i < THREADS_4; i++) {
         pthread_join(threads_4[i], NULL);
        }

        merge(0, (ARRAY_SIZE / 2 - 1) / 2, ARRAY_SIZE / 2 - 1, arr.data());
        merge(ARRAY_SIZE / 2, ARRAY_SIZE / 2 + (ARRAY_SIZE - 1 - ARRAY_SIZE / 2) / 2,
ARRAY_SIZE - 1, arr.data());
        merge(0, (ARRAY_SIZE - 1) / 2, ARRAY_SIZE - 1, arr.data());

        endTime = clock();

        cout << "Sorted array: ";
        for (int i = 0; i < ARRAY_SIZE; i++)
        cout << arr[i] << " ";
        cout << endl << endl;

        cout << "Time taken for selection sort with 4 threads: " << (double)(endTime - startTime) /
(double)CLOCKS_PER_SEC << " seconds" << endl<<endl;
```

```
//merge sort with 4 threads

reset_array(og_arr.data(), arr.data(), ARRAY_SIZE);

cout << "Unsorted array: ";
for (int i = 0; i < ARRAY_SIZE; ++i) {
 cout << arr[i] << " ";
}

cout<<endl<<endl;

startTime = clock();

for (int i = 0; i < THREADS_4; i++) {
 args_4[i].arr = arr.data();
 args_4[i].low = i * segment_size;
 args_4[i].high = (i + 1) * segment_size - 1;
 pthread_create(&threads_4[i], NULL, mergeSort_for_Threads, (void*)&args_4[i]);
}

for (int i = 0; i < THREADS_4; i++)
 pthread_join(threads_4[i], NULL);



merge(0, (ARRAY_SIZE / 2 - 1) / 2, ARRAY_SIZE / 2 - 1, arr.data());
```

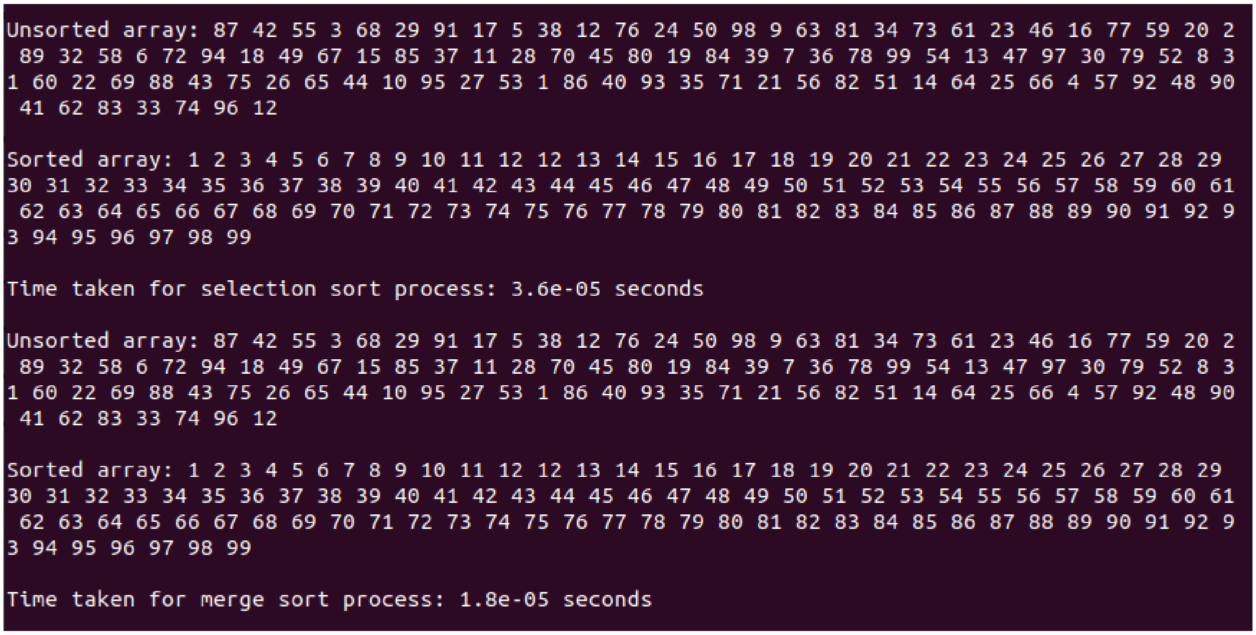
```
        merge(ARRAY_SIZE / 2, ARRAY_SIZE / 2 + (ARRAY_SIZE - 1 - ARRAY_SIZE / 2) / 2,
ARRAY_SIZE - 1, arr.data());

        merge(0, (ARRAY_SIZE - 1) / 2, ARRAY_SIZE - 1, arr.data());



        endTime = clock();


        cout << "Sorted array: ";

        for (int i = 0; i < ARRAY_SIZE; i++)

         cout << arr[i] << " ";

        cout << endl << endl;


        cout << "Time taken for merge sort with 4 threads: " << (double)(endTime - startTime) /
CLOCKS_PER_SEC << " seconds" << endl<<endl;



        //quick sort with 4 threads


        reset_array(og_arr.data(), arr.data(), ARRAY_SIZE);


        cout << "Unsorted array: ";


        for (int i = 0; i < ARRAY_SIZE; ++i) {

         cout << arr[i] << " ";
        }

        cout<<endl<<endl;

        startTime = clock();


        for (int i = 0; i < THREADS_4; i++) {
         args_4[i].arr = arr.data();
         args_4[i].low = i * segment_size;
         args_4[i].high = (i + 1) * segment_size - 1;

        pthread_create(&threads_4[i], NULL, quickSort_for_Threads, (void*)&args_4[i]);
        }

        for (int i = 0; i < THREADS_4; i++)
         pthread_join(threads_4[i], NULL);


        merge(0, (ARRAY_SIZE / 2 - 1) / 2, ARRAY_SIZE / 2 - 1, arr.data());
        merge(ARRAY_SIZE / 2, ARRAY_SIZE / 2 + (ARRAY_SIZE - 1 - ARRAY_SIZE / 2) / 2,
ARRAY_SIZE - 1, arr.data());
        merge(0, (ARRAY_SIZE - 1) / 2, ARRAY_SIZE - 1, arr.data());


        endTime = clock();
        cout << "Sorted array: ";

        for (int i = 0; i < ARRAY_SIZE; i++)

         cout << arr[i] << " ";

        cout << endl << endl;



   cout << "Time taken for quick sort with 4 threads: " << (double)(endTime - startTime) /
(double)CLOCKS_PER_SEC << " seconds" << endl<<endl;






        return 0;

}
```

Results (comparison via graphs):

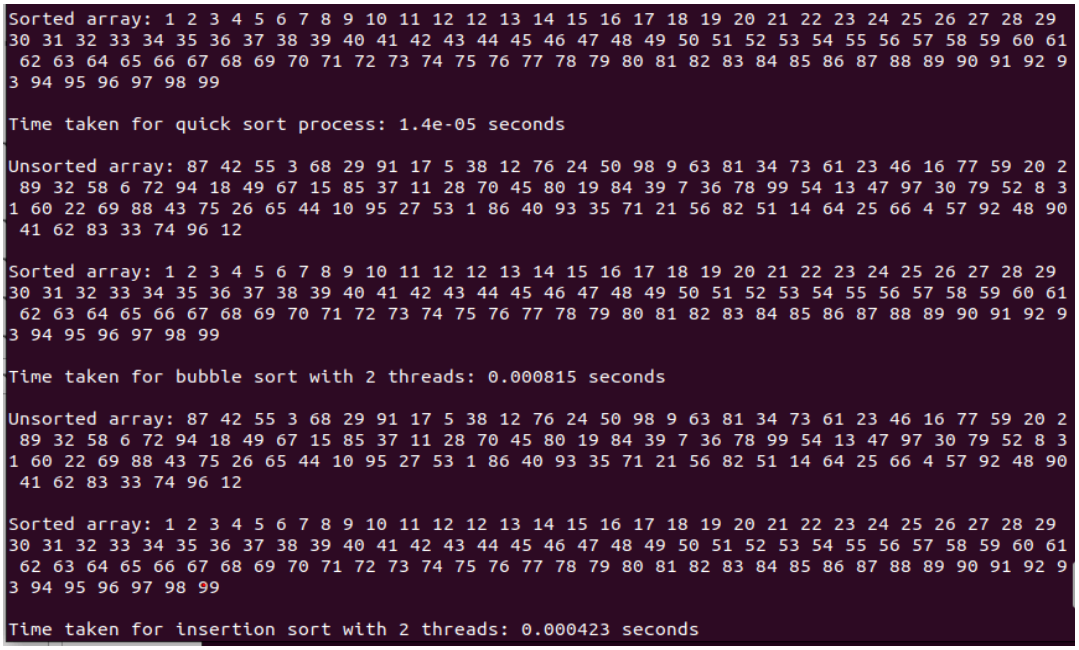
Results with 100 data points:



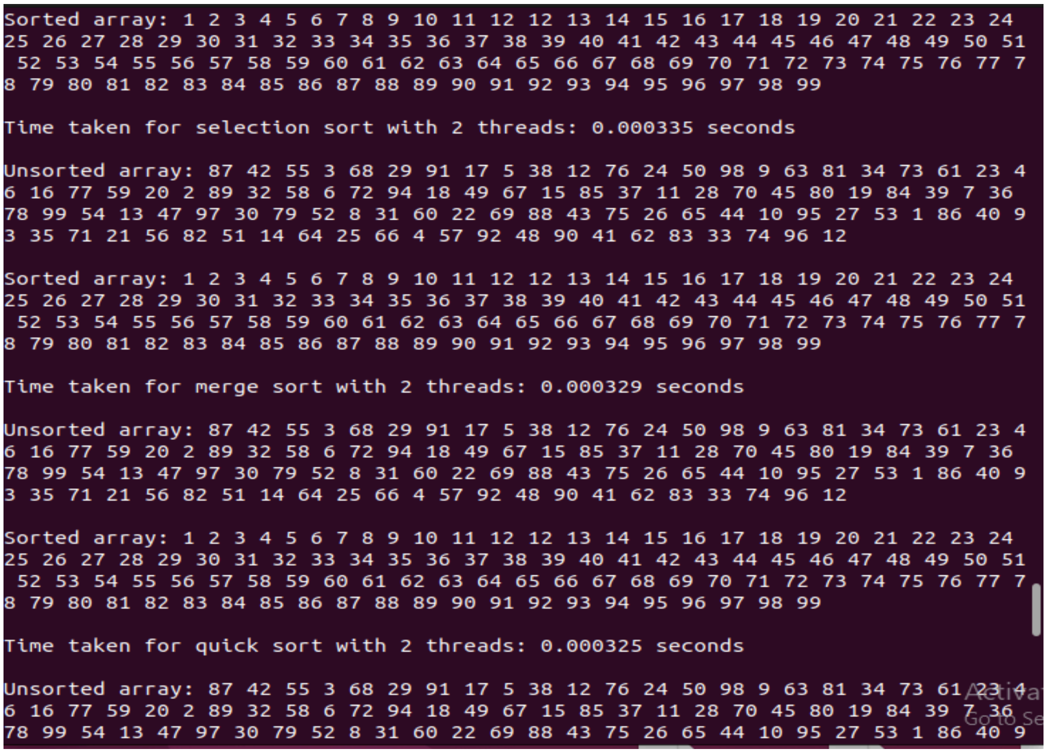
```
aniqa@aniqa:~/Desktop/PROJECT$ g++ ProcessesVSThreads.cpp -o ProcessesVSThreads -lpthread
aniqa@aniqa:~/Desktop/PROJECT$ ./ProcessesVSThreads
Enter the name of file you want elements from: 100.txt
Unsorted array: 87 42 55 3 68 29 91 17 5 38 12 76 24 50 98 9 63 81 34 73 61 23 46 16 77 59 20 2
 89 32 58 6 72 94 18 49 67 15 85 37 11 28 70 45 80 19 84 39 7 36 78 99 54 13 47 97 30 79 52 8 3
1 60 22 69 88 43 75 26 65 44 10 95 27 53 1 86 40 93 35 71 21 56 82 51 14 64 25 66 4 57 92 48 90
 41 62 83 33 74 96 12

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

Time taken for bubble sort process: 9e-05 seconds

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

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

Time taken for insertion sort process: 2.2e-05 seconds
```



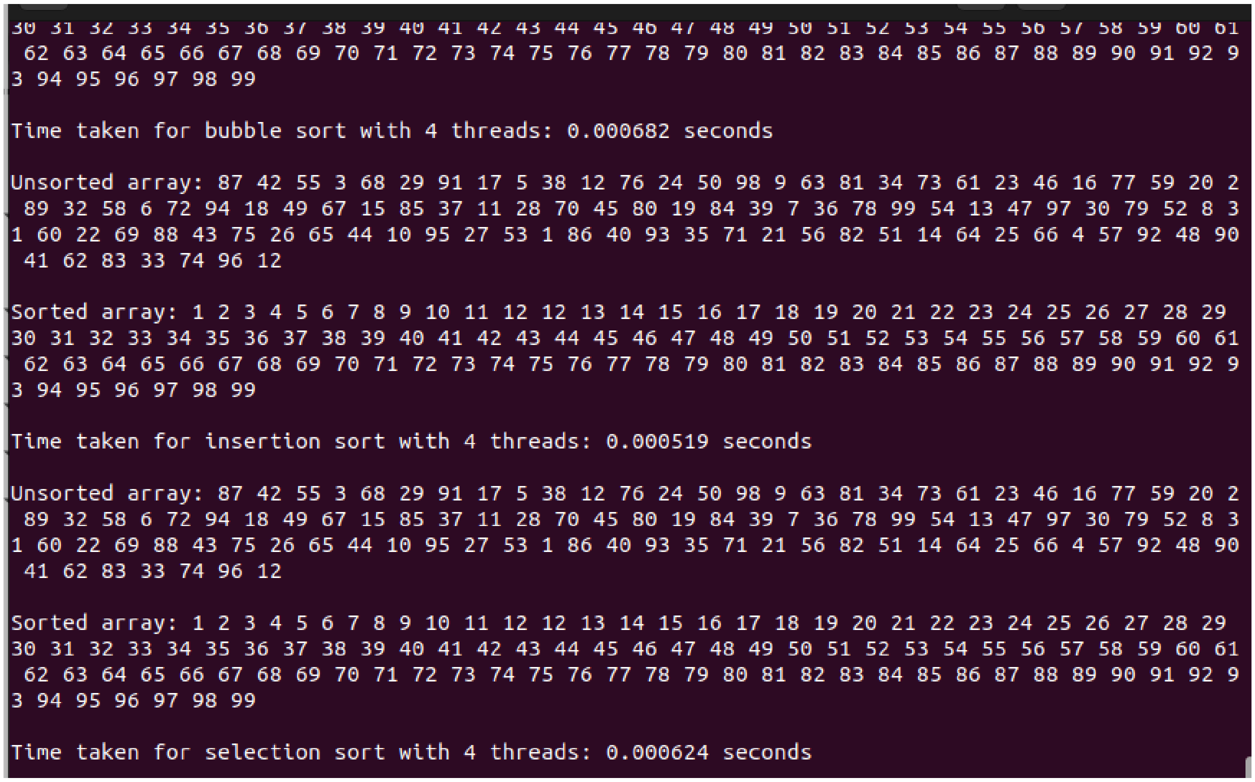
```
Unsorted array: 87 42 55 3 68 29 91 17 5 38 12 76 24 50 98 9 63 81 34 73 61 23 46 16 77 59 20 2
 89 32 58 6 72 94 18 49 67 15 85 37 11 28 70 45 80 19 84 39 7 36 78 99 54 13 47 97 30 79 52 8 3
1 60 22 69 88 43 75 26 65 44 10 95 27 53 1 86 40 93 35 71 21 56 82 51 14 64 25 66 4 57 92 48 90
 41 62 83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

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

Time taken for selection sort process: 3.6e-05 seconds

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

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

Time taken for merge sort process: 1.8e-05 seconds
```



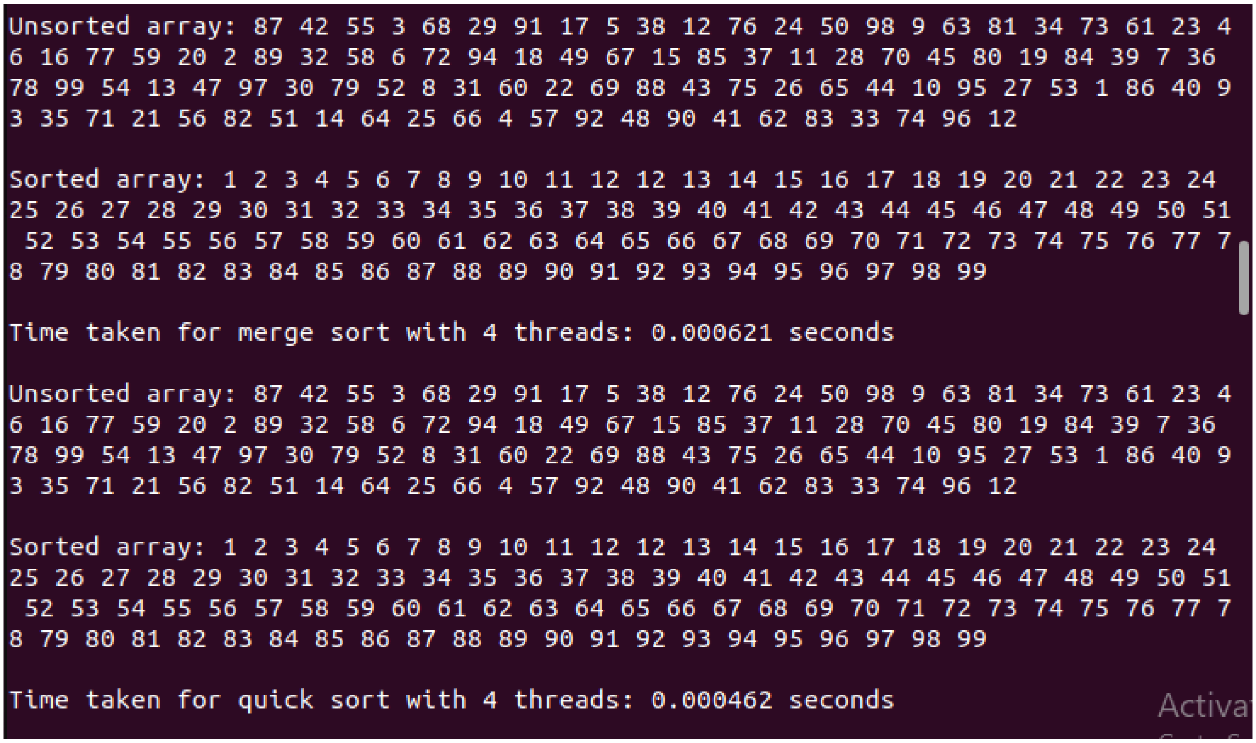
```
Sorted array: 1 2 3 4 5 6 7 8 9 10 11 12 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 9
3 94 95 96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

Time taken for quick sort process: 1.4e-05 seconds

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

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

Time taken for bubble sort with 2 threads: 0.000815 seconds

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

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

Time taken for insertion sort with 2 threads: 0.000423 seconds
```



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

Time taken for selection sort with 2 threads: 0.000335 seconds

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

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

Time taken for merge sort with 2 threads: 0.000329 seconds

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

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

Time taken for quick sort with 2 threads: 0.000325 seconds

Unsorted array: 87 42 55 3 68 29 91 17 5 38 12 76 24 50 98 9 63 81 34 73 61 23 4
6 16 77 59 20 2 89 32 58 6 72 94 18 49 67 15 85 37 11 28 70 45 80 19 84 39 7 36
78 99 54 13 47 97 30 79 52 8 31 60 22 69 88 43 75 26 65 44 10 95 27 53 1 86 40 9
```

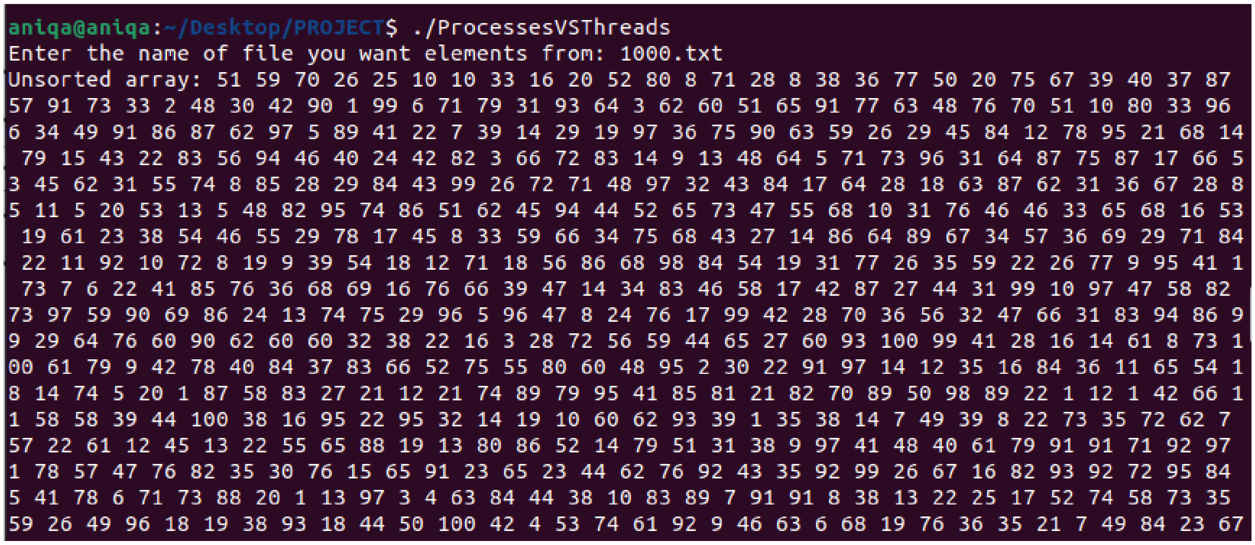


```
30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 9
3 94 95 96 97 98 99

Time taken for bubble sort with 4 threads: 0.000682 seconds

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

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

Time taken for insertion sort with 4 threads: 0.000519 seconds

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

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

Time taken for selection sort with 4 threads: 0.000624 seconds
```

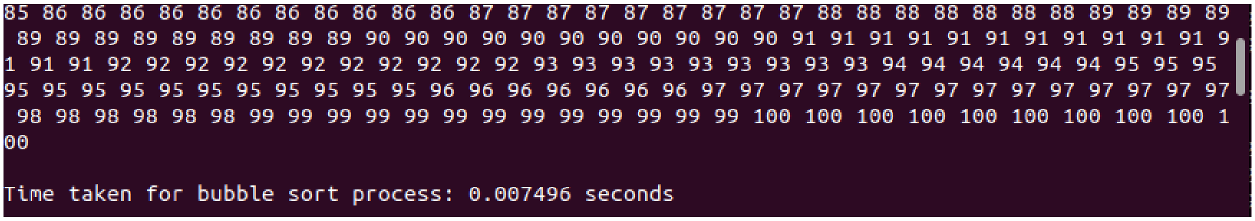


```
Unsorted array: 87 42 55 3 68 29 91 17 5 38 12 76 24 50 98 9 63 81 34 73 61 23 4
6 16 77 59 20 2 89 32 58 6 72 94 18 49 67 15 85 37 11 28 70 45 80 19 84 39 7 36
78 99 54 13 47 97 30 79 52 8 31 60 22 69 88 43 75 26 65 44 10 95 27 53 1 86 40 9
3 35 71 21 56 82 51 14 64 25 66 4 57 92 48 90 41 62 83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

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

Time taken for merge sort with 4 threads: 0.000621 seconds

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

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

Time taken for quick sort with 4 threads: 0.000462 seconds
```

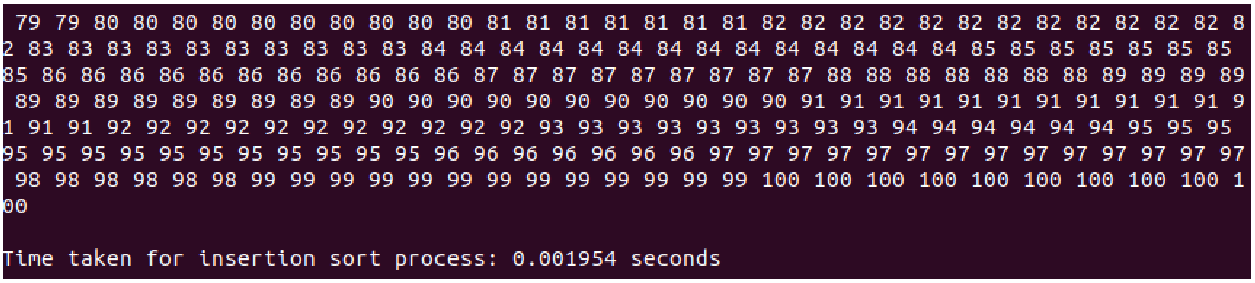
Results with 1000 data points:



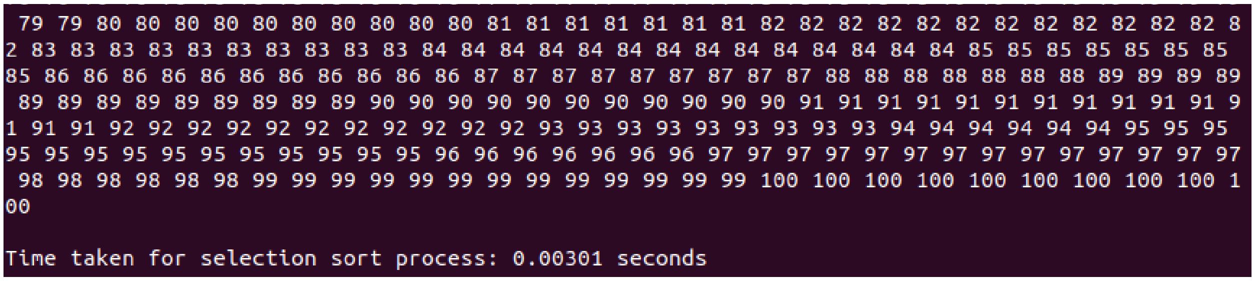
```
aniqa@aniqa:~/Desktop/PROJECT$ ./ProcessesVSThreads
Enter the name of file you want elements from: 1000.txt
Unsorted array: 51 59 70 26 25 10 10 33 16 20 52 80 8 71 28 8 38 36 77 50 20 75 67 39 40 37 87
57 91 73 33 2 48 30 42 90 1 99 6 71 79 31 93 64 3 62 60 51 65 91 77 63 48 76 70 51 10 80 33 96
6 34 49 91 86 87 62 97 5 89 41 22 7 39 14 29 19 97 36 75 90 63 59 26 29 45 84 12 78 95 21 68 14
 79 15 43 22 83 56 94 46 40 24 42 82 3 66 72 83 14 9 13 48 64 5 71 73 96 31 64 87 75 87 17 66 5
3 45 62 31 55 74 8 85 28 29 84 43 99 26 72 71 48 97 32 43 84 17 64 28 18 63 87 62 31 36 67 28 8
5 11 5 20 53 13 5 48 82 95 74 86 51 62 45 94 44 52 65 73 47 55 68 10 31 76 46 46 33 65 68 16 53
 19 61 23 38 54 46 55 29 78 17 45 8 33 59 66 34 75 68 43 27 14 86 64 89 67 34 57 36 69 29 71 84
 22 11 92 10 72 8 19 9 39 54 18 12 71 18 56 86 68 98 84 54 19 31 77 26 35 59 22 26 77 9 95 41 1
 73 7 6 22 41 85 76 36 68 69 16 76 66 39 47 14 34 83 46 58 17 42 87 27 44 31 99 10 97 47 58 82
73 97 59 90 69 86 24 13 74 75 29 96 5 96 47 8 24 76 17 99 42 28 70 36 56 32 47 66 31 83 94 86 9
9 29 64 76 60 90 62 60 60 32 38 22 16 3 28 72 56 59 44 65 27 60 93 100 99 41 28 16 14 61 8 73 1
00 61 79 9 42 78 40 84 37 83 66 52 75 55 80 60 48 95 2 30 22 91 97 14 12 35 16 84 36 11 65 54 1
8 14 74 5 20 1 87 58 83 27 21 12 21 74 89 79 95 41 85 81 21 82 70 89 50 98 89 22 1 12 1 42 66 1
1 58 58 39 44 100 38 16 95 22 95 32 14 19 10 60 62 93 39 1 35 38 14 7 49 39 8 22 73 35 72 62 7
57 22 61 12 45 13 22 55 65 88 19 13 80 86 52 14 79 51 31 38 9 97 41 48 40 61 79 91 91 71 92 97
1 78 57 47 76 82 35 30 76 15 65 91 23 65 23 44 62 76 92 43 35 92 99 26 67 16 82 93 92 72 95 84
5 41 78 6 71 73 88 20 1 13 97 3 4 63 84 44 38 10 83 89 7 91 91 8 38 13 22 25 17 52 74 58 73 35
59 26 49 96 18 19 38 93 18 44 50 100 42 4 53 74 61 92 9 46 63 6 68 19 76 36 35 21 7 49 84 23 67
```



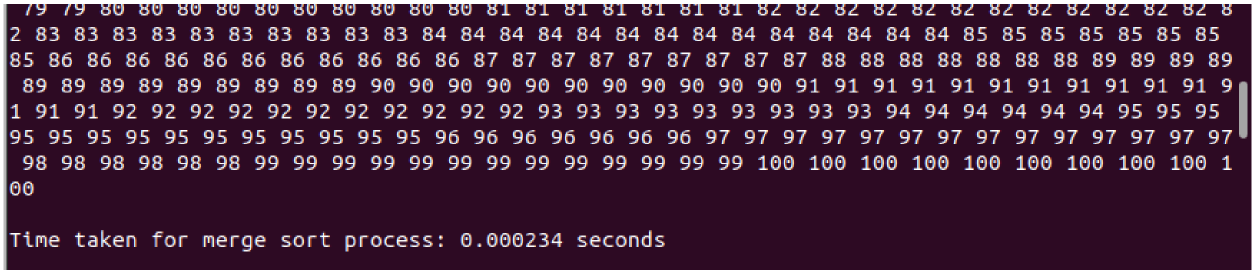
```
85 86 86 86 86 86 86 86 86 86 86 86 87 87 87 87 87 87 87 87 87 88 88 88 88 88 88 88 89 89 89 89
 89 89 89 89 89 89 89 89 89 90 90 90 90 90 90 90 90 90 90 90 91 91 91 91 91 91 91 91 91 91 91 9
1 91 91 92 92 92 92 92 92 92 92 92 92 92 93 93 93 93 93 93 93 93 93 94 94 94 94 94 94 95 95 95
95 95 95 95 95 95 95 95 95 95 95 96 96 96 96 96 96 96 97 97 97 97 97 97 97 97 97 97 97 97 97 97
 98 98 98 98 98 98 99 99 99 99 99 99 99 99 99 99 99 99 99 100 100 100 100 100 100 100 100 100 1
00

Time taken for bubble sort process: 0.007496 seconds
```



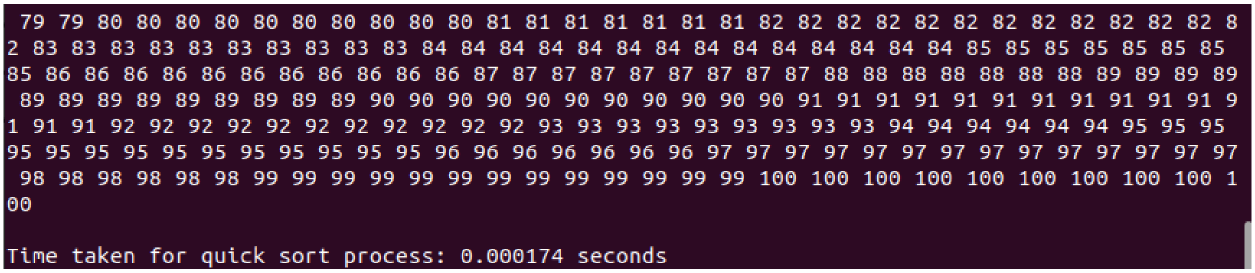
```
 79 79 80 80 80 80 80 80 80 80 80 80 81 81 81 81 81 81 81 82 82 82 82 82 82 82 82 82 82 82 82 8
2 83 83 83 83 83 83 83 83 83 83 84 84 84 84 84 84 84 84 84 84 84 84 84 84 85 85 85 85 85 85 85
85 86 86 86 86 86 86 86 86 86 86 86 87 87 87 87 87 87 87 87 87 88 88 88 88 88 88 88 89 89 89 89
 89 89 89 89 89 89 89 89 89 90 90 90 90 90 90 90 90 90 90 90 91 91 91 91 91 91 91 91 91 91 91 9
1 91 91 92 92 92 92 92 92 92 92 92 92 92 93 93 93 93 93 93 93 93 93 94 94 94 94 94 94 95 95 95
95 95 95 95 95 95 95 95 95 95 95 96 96 96 96 96 96 96 97 97 97 97 97 97 97 97 97 97 97 97 97 97
 98 98 98 98 98 98 99 99 99 99 99 99 99 99 99 99 99 99 99 100 100 100 100 100 100 100 100 100 1
00

Time taken for insertion sort process: 0.001954 seconds
```



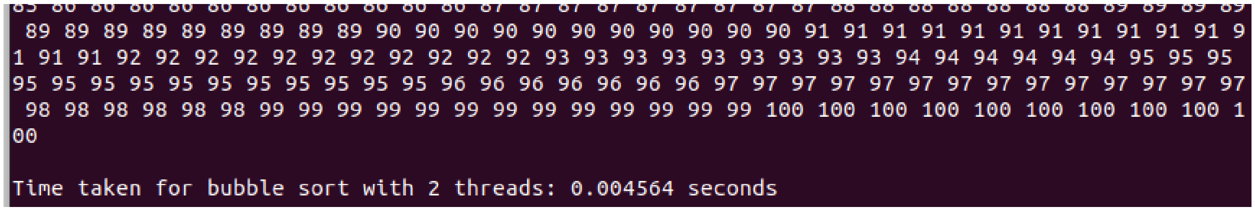
```
 79 79 80 80 80 80 80 80 80 80 80 80 81 81 81 81 81 81 81 82 82 82 82 82 82 82 82 82 82 82 82 8
2 83 83 83 83 83 83 83 83 83 83 84 84 84 84 84 84 84 84 84 84 84 84 84 84 85 85 85 85 85 85 85
85 86 86 86 86 86 86 86 86 86 86 86 87 87 87 87 87 87 87 87 87 88 88 88 88 88 88 88 89 89 89 89
 89 89 89 89 89 89 89 89 89 90 90 90 90 90 90 90 90 90 90 90 91 91 91 91 91 91 91 91 91 91 91 9
1 91 91 92 92 92 92 92 92 92 92 92 92 92 93 93 93 93 93 93 93 93 93 94 94 94 94 94 94 95 95 95
95 95 95 95 95 95 95 95 95 95 95 96 96 96 96 96 96 96 97 97 97 97 97 97 97 97 97 97 97 97 97 97
 98 98 98 98 98 98 99 99 99 99 99 99 99 99 99 99 99 99 99 100 100 100 100 100 100 100 100 100 1
00

Time taken for selection sort process: 0.00301 seconds
```



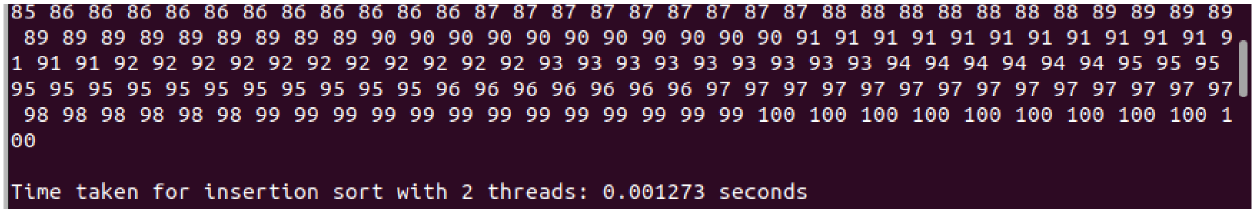
```
 79 79 80 80 80 80 80 80 80 80 80 80 81 81 81 81 81 81 81 82 82 82 82 82 82 82 82 82 82 82 82 8
2 83 83 83 83 83 83 83 83 83 83 84 84 84 84 84 84 84 84 84 84 84 84 84 84 85 85 85 85 85 85 85
85 86 86 86 86 86 86 86 86 86 86 86 87 87 87 87 87 87 87 87 87 88 88 88 88 88 88 88 89 89 89 89
 89 89 89 89 89 89 89 89 89 90 90 90 90 90 90 90 90 90 90 90 91 91 91 91 91 91 91 91 91 91 91 9
1 91 91 92 92 92 92 92 92 92 92 92 92 92 93 93 93 93 93 93 93 93 93 94 94 94 94 94 94 95 95 95
95 95 95 95 95 95 95 95 95 95 95 96 96 96 96 96 96 96 97 97 97 97 97 97 97 97 97 97 97 97 97 97
 98 98 98 98 98 98 99 99 99 99 99 99 99 99 99 99 99 99 99 100 100 100 100 100 100 100 100 100 1
00

Time taken for merge sort process: 0.000234 seconds
```



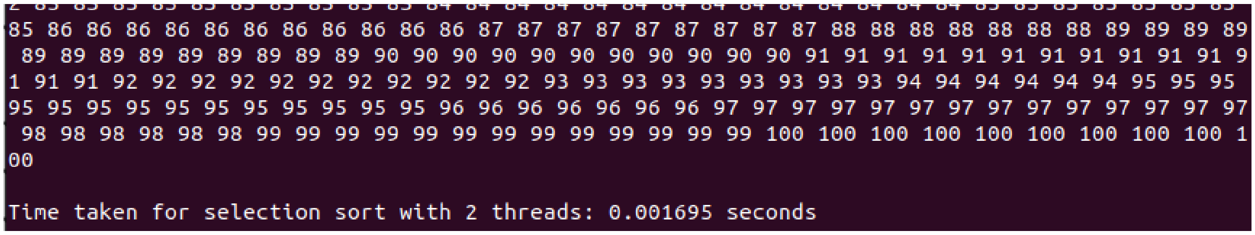
```
 79 79 80 80 80 80 80 80 80 80 80 80 81 81 81 81 81 81 81 82 82 82 82 82 82 82 82 82 82 82 82 8
2 83 83 83 83 83 83 83 83 83 83 84 84 84 84 84 84 84 84 84 84 84 84 84 84 85 85 85 85 85 85 85
85 86 86 86 86 86 86 86 86 86 86 86 87 87 87 87 87 87 87 87 87 88 88 88 88 88 88 88 89 89 89 89
 89 89 89 89 89 89 89 89 89 90 90 90 90 90 90 90 90 90 90 90 91 91 91 91 91 91 91 91 91 91 91 9
1 91 91 92 92 92 92 92 92 92 92 92 92 92 93 93 93 93 93 93 93 93 93 94 94 94 94 94 94 95 95 95
95 95 95 95 95 95 95 95 95 95 95 96 96 96 96 96 96 96 97 97 97 97 97 97 97 97 97 97 97 97 97 97
 98 98 98 98 98 98 99 99 99 99 99 99 99 99 99 99 99 99 99 100 100 100 100 100 100 100 100 100 1
00

Time taken for quick sort process: 0.000174 seconds
```



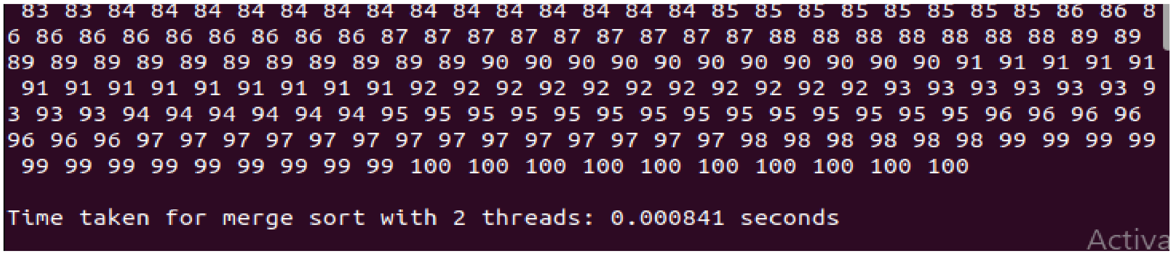
```
85 86 86 86 86 86 86 86 86 86 86 86 87 87 87 87 87 87 87 87 87 88 88 88 88 88 88 88 89 89 89 89
 89 89 89 89 89 89 89 89 89 90 90 90 90 90 90 90 90 90 90 90 91 91 91 91 91 91 91 91 91 91 91 9
1 91 91 92 92 92 92 92 92 92 92 92 92 92 93 93 93 93 93 93 93 93 93 94 94 94 94 94 94 95 95 95
95 95 95 95 95 95 95 95 95 95 95 96 96 96 96 96 96 96 97 97 97 97 97 97 97 97 97 97 97 97 97 97
 98 98 98 98 98 98 99 99 99 99 99 99 99 99 99 99 99 99 99 100 100 100 100 100 100 100 100 100 1
00

Time taken for bubble sort with 2 threads: 0.004564 seconds
```



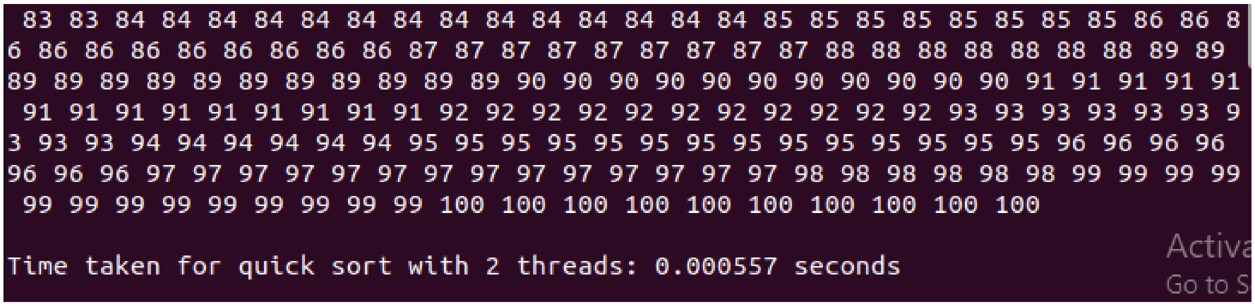
```
85 86 86 86 86 86 86 86 86 86 86 86 87 87 87 87 87 87 87 87 87 88 88 88 88 88 88 88 89 89 89 89
 89 89 89 89 89 89 89 89 89 90 90 90 90 90 90 90 90 90 90 90 91 91 91 91 91 91 91 91 91 91 91 9
1 91 91 92 92 92 92 92 92 92 92 92 92 92 93 93 93 93 93 93 93 93 93 94 94 94 94 94 94 95 95 95
95 95 95 95 95 95 95 95 95 95 95 96 96 96 96 96 96 96 97 97 97 97 97 97 97 97 97 97 97 97 97 97
 98 98 98 98 98 98 99 99 99 99 99 99 99 99 99 99 99 99 99 100 100 100 100 100 100 100 100 100 1
00

Time taken for insertion sort with 2 threads: 0.001273 seconds
```



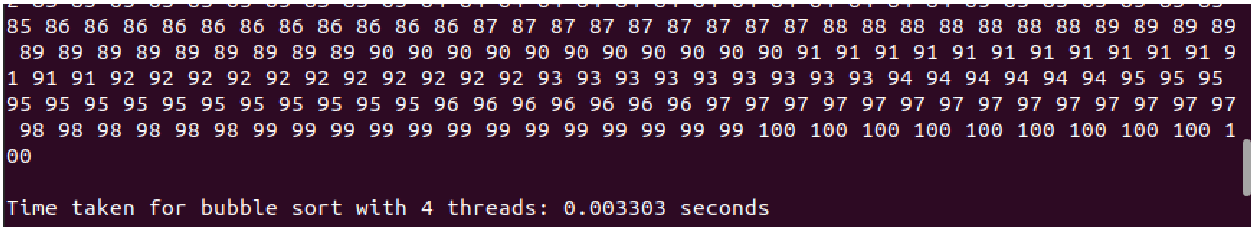
```
2 83 83 83 83 83 83 83 83 83 83 84 84 84 84 84 84 84 84 84 84 84 84 84 84 85 85 85 85 85 85 85
85 86 86 86 86 86 86 86 86 86 86 86 87 87 87 87 87 87 87 87 87 88 88 88 88 88 88 88 89 89 89 89
 89 89 89 89 89 89 89 89 89 90 90 90 90 90 90 90 90 90 90 90 91 91 91 91 91 91 91 91 91 91 91 9
1 91 91 92 92 92 92 92 92 92 92 92 92 92 93 93 93 93 93 93 93 93 93 94 94 94 94 94 94 95 95 95
95 95 95 95 95 95 95 95 95 95 95 96 96 96 96 96 96 96 97 97 97 97 97 97 97 97 97 97 97 97 97 97
 98 98 98 98 98 98 99 99 99 99 99 99 99 99 99 99 99 99 99 100 100 100 100 100 100 100 100 100 1
00

Time taken for selection sort with 2 threads: 0.001695 seconds
```



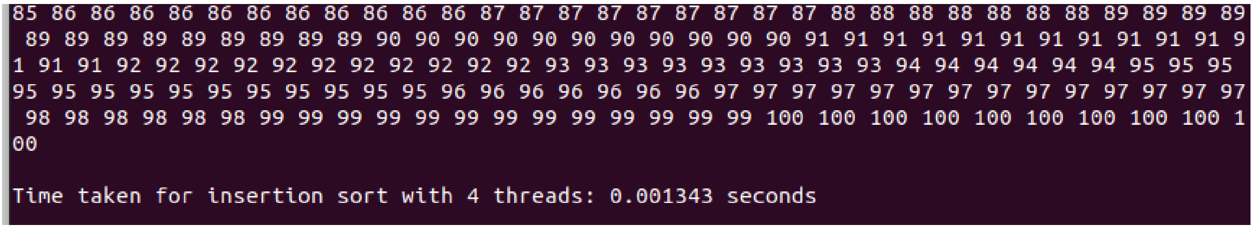
```
 83 83 84 84 84 84 84 84 84 84 84 84 84 84 84 84 85 85 85 85 85 85 85 85 86 86 8
6 86 86 86 86 86 86 86 86 87 87 87 87 87 87 87 87 87 88 88 88 88 88 88 88 89 89
89 89 89 89 89 89 89 89 89 89 89 90 90 90 90 90 90 90 90 90 90 90 91 91 91 91 91
 91 91 91 91 91 91 91 91 91 92 92 92 92 92 92 92 92 92 92 92 93 93 93 93 93 93 9
3 93 93 94 94 94 94 94 94 95 95 95 95 95 95 95 95 95 95 95 95 95 95 96 96 96 96
96 96 96 97 97 97 97 97 97 97 97 97 97 97 97 97 97 98 98 98 98 98 98 99 99 99 99
 99 99 99 99 99 99 99 99 99 100 100 100 100 100 100 100 100 100 100

Time taken for merge sort with 2 threads: 0.000841 seconds
```



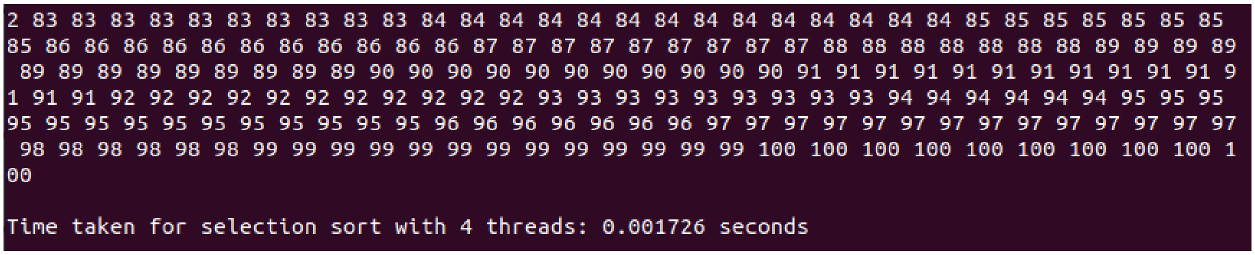
```
 83 83 84 84 84 84 84 84 84 84 84 84 84 84 84 84 85 85 85 85 85 85 85 85 86 86 8
6 86 86 86 86 86 86 86 86 87 87 87 87 87 87 87 87 87 88 88 88 88 88 88 88 89 89
89 89 89 89 89 89 89 89 89 89 89 90 90 90 90 90 90 90 90 90 90 90 91 91 91 91 91
 91 91 91 91 91 91 91 91 91 92 92 92 92 92 92 92 92 92 92 92 93 93 93 93 93 93 9
3 93 93 94 94 94 94 94 94 95 95 95 95 95 95 95 95 95 95 95 95 95 95 96 96 96 96
96 96 96 97 97 97 97 97 97 97 97 97 97 97 97 97 97 98 98 98 98 98 98 99 99 99 99
 99 99 99 99 99 99 99 99 99 100 100 100 100 100 100 100 100 100 100

Time taken for quick sort with 2 threads: 0.000557 seconds
```



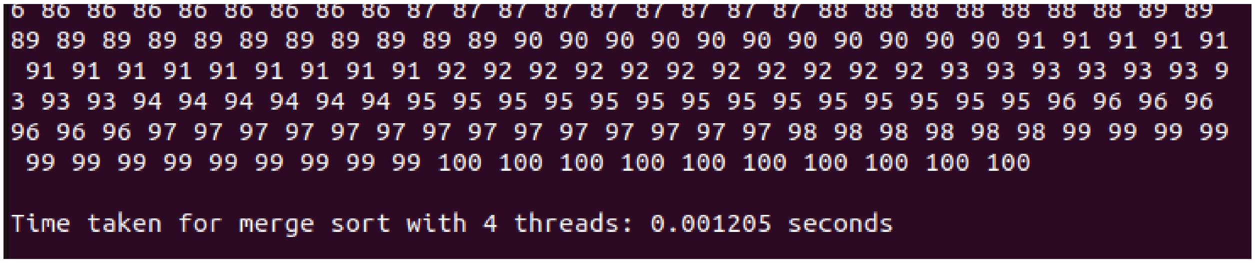
```
85 86 86 86 86 86 86 86 86 86 86 86 87 87 87 87 87 87 87 87 87 88 88 88 88 88 88 88 89 89 89 89
 89 89 89 89 89 89 89 89 89 90 90 90 90 90 90 90 90 90 90 90 91 91 91 91 91 91 91 91 91 91 91 9
1 91 91 92 92 92 92 92 92 92 92 92 92 92 93 93 93 93 93 93 93 93 93 94 94 94 94 94 94 95 95 95
95 95 95 95 95 95 95 95 95 95 95 96 96 96 96 96 96 96 97 97 97 97 97 97 97 97 97 97 97 97 97 97
 98 98 98 98 98 98 99 99 99 99 99 99 99 99 99 99 99 99 99 100 100 100 100 100 100 100 100 100 1
00

Time taken for bubble sort with 4 threads: 0.003303 seconds
```



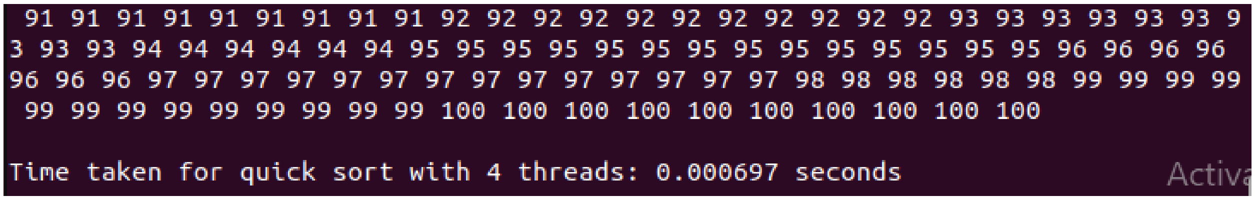
```
85 86 86 86 86 86 86 86 86 86 86 86 87 87 87 87 87 87 87 87 87 88 88 88 88 88 88 88 89 89 89 89
 89 89 89 89 89 89 89 89 89 90 90 90 90 90 90 90 90 90 90 90 91 91 91 91 91 91 91 91 91 91 91 9
1 91 91 92 92 92 92 92 92 92 92 92 92 92 93 93 93 93 93 93 93 93 93 94 94 94 94 94 94 95 95 95
95 95 95 95 95 95 95 95 95 95 95 96 96 96 96 96 96 96 97 97 97 97 97 97 97 97 97 97 97 97 97 97
 98 98 98 98 98 98 99 99 99 99 99 99 99 99 99 99 99 99 99 100 100 100 100 100 100 100 100 100 1
00

Time taken for insertion sort with 4 threads: 0.001343 seconds
```



```
2 83 83 83 83 83 83 83 83 83 83 84 84 84 84 84 84 84 84 84 84 84 84 84 84 85 85 85 85 85 85 85
85 86 86 86 86 86 86 86 86 86 86 86 87 87 87 87 87 87 87 87 87 88 88 88 88 88 88 88 89 89 89 89
 89 89 89 89 89 89 89 89 89 90 90 90 90 90 90 90 90 90 90 90 91 91 91 91 91 91 91 91 91 91 91 9
1 91 91 92 92 92 92 92 92 92 92 92 92 92 93 93 93 93 93 93 93 93 93 94 94 94 94 94 94 95 95 95
95 95 95 95 95 95 95 95 95 95 95 96 96 96 96 96 96 96 97 97 97 97 97 97 97 97 97 97 97 97 97 97
 98 98 98 98 98 98 99 99 99 99 99 99 99 99 99 99 99 99 99 100 100 100 100 100 100 100 100 100 1
00

Time taken for selection sort with 4 threads: 0.001726 seconds
```

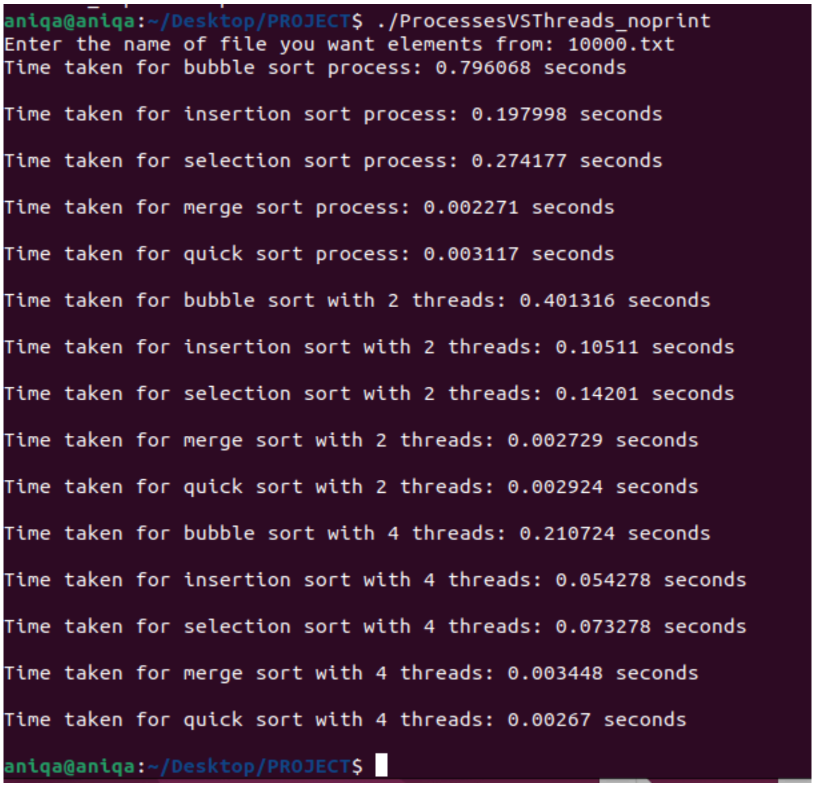


```
6 86 86 86 86 86 86 86 86 87 87 87 87 87 87 87 87 87 88 88 88 88 88 88 88 89 89
89 89 89 89 89 89 89 89 89 89 89 90 90 90 90 90 90 90 90 90 90 90 91 91 91 91 91
 91 91 91 91 91 91 91 91 91 92 92 92 92 92 92 92 92 92 92 92 93 93 93 93 93 93 9
3 93 93 94 94 94 94 94 94 95 95 95 95 95 95 95 95 95 95 95 95 95 95 96 96 96 96
96 96 96 97 97 97 97 97 97 97 97 97 97 97 97 97 97 98 98 98 98 98 98 99 99 99 99
 99 99 99 99 99 99 99 99 99 100 100 100 100 100 100 100 100 100 100

Time taken for merge sort with 4 threads: 0.001205 seconds
```



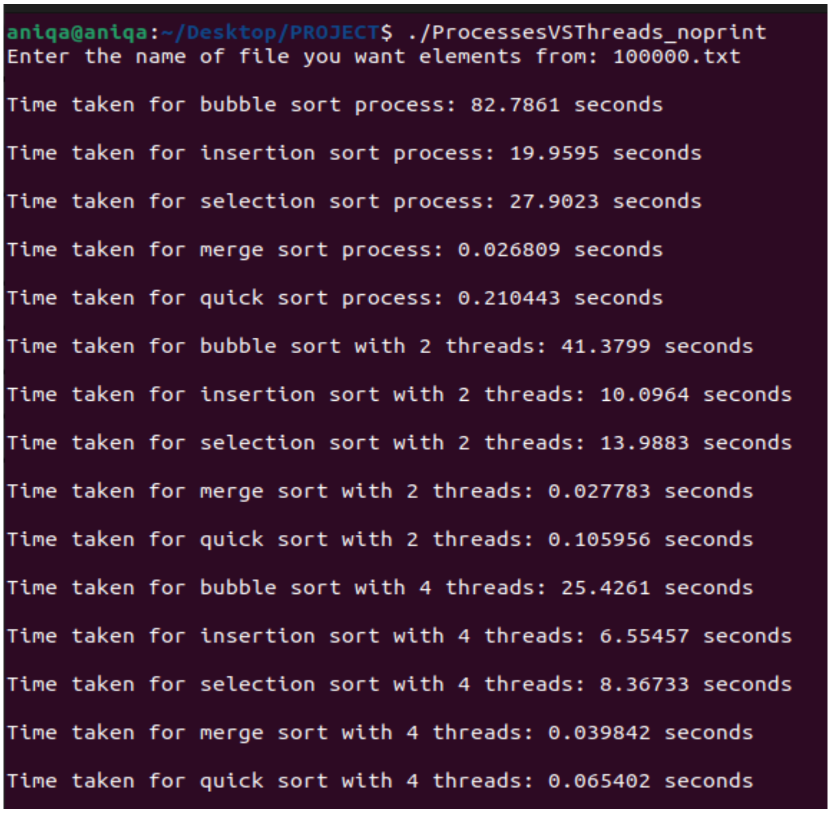
```
 91 91 91 91 91 91 91 91 91 92 92 92 92 92 92 92 92 92 92 92 93 93 93 93 93 93 9
3 93 93 94 94 94 94 94 94 95 95 95 95 95 95 95 95 95 95 95 95 95 95 96 96 96 96
96 96 96 97 97 97 97 97 97 97 97 97 97 97 97 97 97 98 98 98 98 98 98 99 99 99 99
 99 99 99 99 99 99 99 99 99 100 100 100 100 100 100 100 100 100 100

Time taken for quick sort with 4 threads: 0.000697 seconds
```

Results with 10,000 data points:



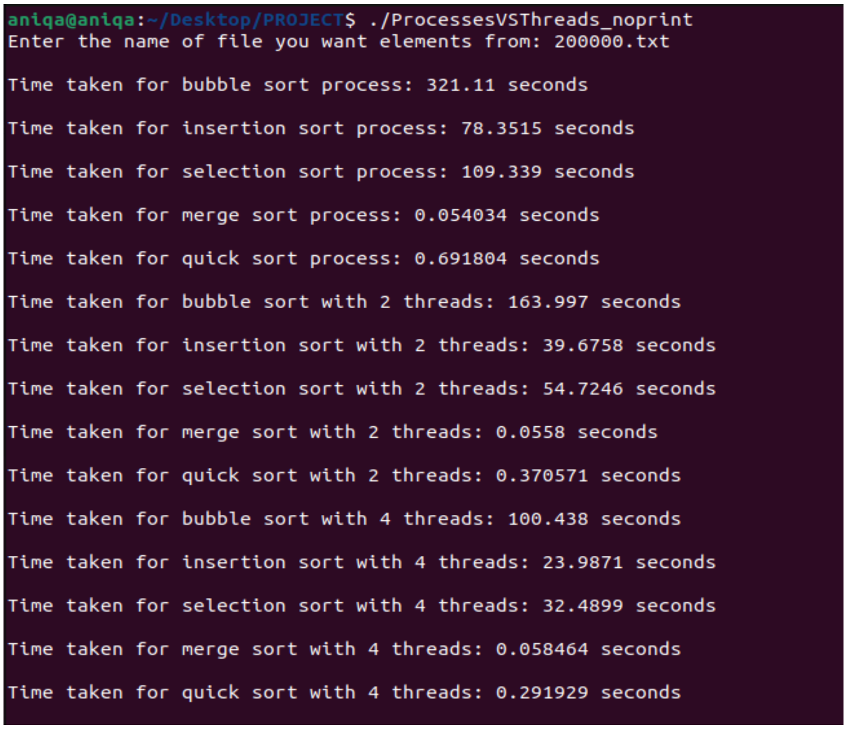
```
aniqa@aniqa:~/Desktop/PROJECT$ ./ProcessesVSThreads_noprint
Enter the name of file you want elements from: 10000.txt
Time taken for bubble sort process: 0.796068 seconds

Time taken for insertion sort process: 0.197998 seconds

Time taken for selection sort process: 0.274177 seconds

Time taken for merge sort process: 0.002271 seconds

Time taken for quick sort process: 0.003117 seconds

Time taken for bubble sort with 2 threads: 0.401316 seconds

Time taken for insertion sort with 2 threads: 0.10511 seconds

Time taken for selection sort with 2 threads: 0.14201 seconds

Time taken for merge sort with 2 threads: 0.002729 seconds

Time taken for quick sort with 2 threads: 0.002924 seconds

Time taken for bubble sort with 4 threads: 0.210724 seconds

Time taken for insertion sort with 4 threads: 0.054278 seconds

Time taken for selection sort with 4 threads: 0.073278 seconds

Time taken for merge sort with 4 threads: 0.003448 seconds

Time taken for quick sort with 4 threads: 0.00267 seconds

aniqa@aniqa:~/Desktop/PROJECT$ 
```

Results with 100,000 data points:



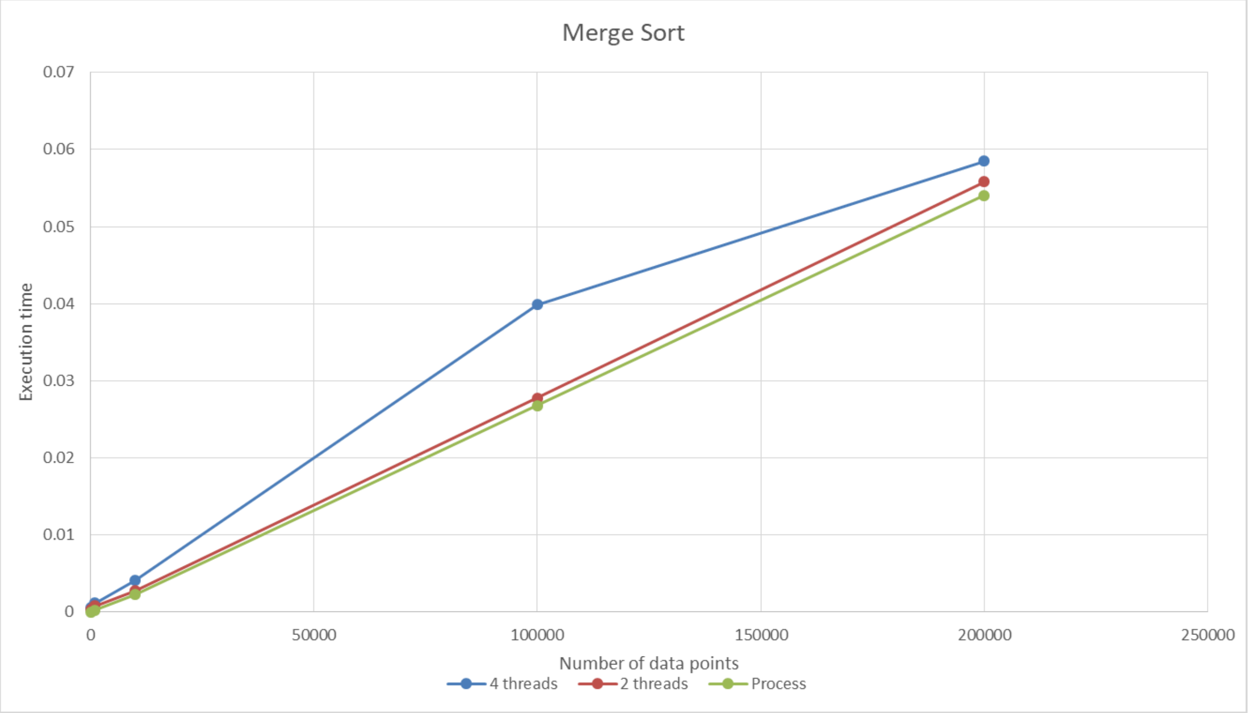
```
aniqa@aniqa:~/Desktop/PROJECT$ ./ProcessesVSThreads_noprint
Enter the name of file you want elements from: 100000.txt

Time taken for bubble sort process: 82.7861 seconds

Time taken for insertion sort process: 19.9595 seconds

Time taken for selection sort process: 27.9023 seconds

Time taken for merge sort process: 0.026809 seconds

Time taken for quick sort process: 0.210443 seconds

Time taken for bubble sort with 2 threads: 41.3799 seconds

Time taken for insertion sort with 2 threads: 10.0964 seconds

Time taken for selection sort with 2 threads: 13.9883 seconds

Time taken for merge sort with 2 threads: 0.027783 seconds

Time taken for quick sort with 2 threads: 0.105956 seconds

Time taken for bubble sort with 4 threads: 25.4261 seconds

Time taken for insertion sort with 4 threads: 6.55457 seconds

Time taken for selection sort with 4 threads: 8.36733 seconds

Time taken for merge sort with 4 threads: 0.039842 seconds

Time taken for quick sort with 4 threads: 0.065402 seconds
```

Results with 200,000 data points:

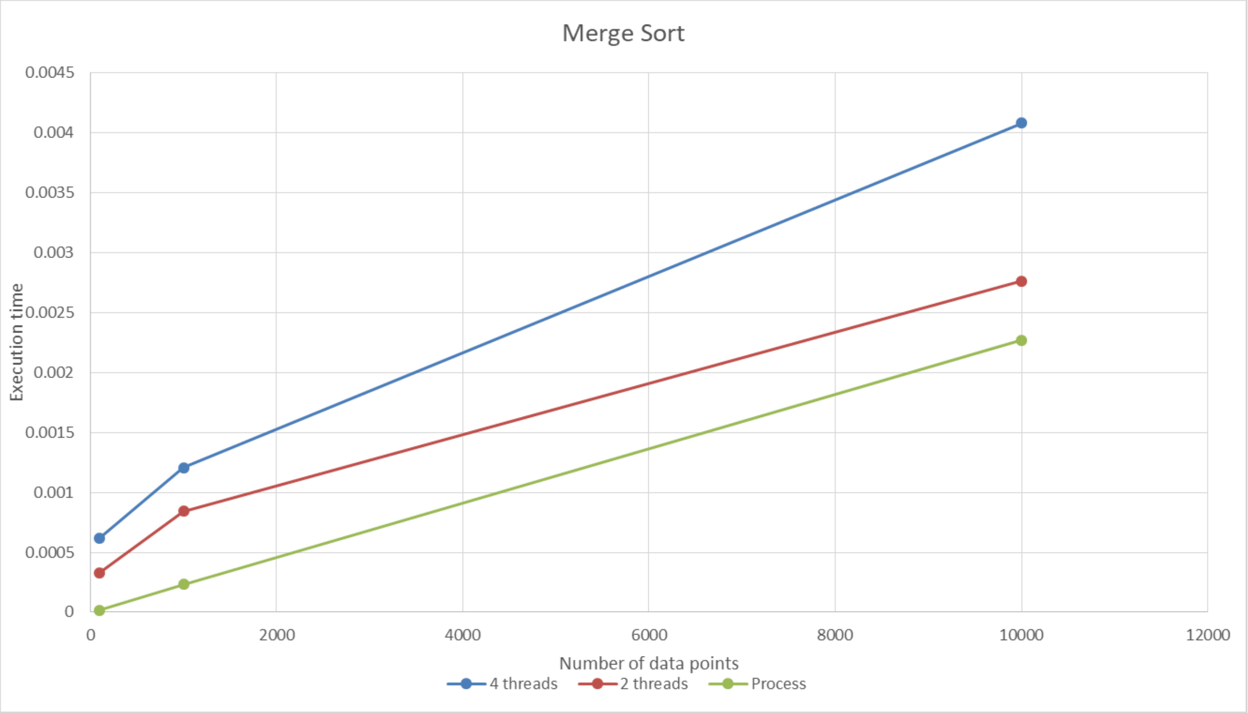


```
aniqa@aniqa:~/Desktop/PROJECT$ ./ProcessesVSThreads_noprint
Enter the name of file you want elements from: 200000.txt

Time taken for bubble sort process: 321.11 seconds

Time taken for insertion sort process: 78.3515 seconds

Time taken for selection sort process: 109.339 seconds

Time taken for merge sort process: 0.054034 seconds

Time taken for quick sort process: 0.691804 seconds

Time taken for bubble sort with 2 threads: 163.997 seconds

Time taken for insertion sort with 2 threads: 39.6758 seconds

Time taken for selection sort with 2 threads: 54.7246 seconds

Time taken for merge sort with 2 threads: 0.0558 seconds

Time taken for quick sort with 2 threads: 0.370571 seconds

Time taken for bubble sort with 4 threads: 100.438 seconds

Time taken for insertion sort with 4 threads: 23.9871 seconds

Time taken for selection sort with 4 threads: 32.4899 seconds

Time taken for merge sort with 4 threads: 0.058464 seconds

Time taken for quick sort with 4 threads: 0.291929 seconds
```

Merge Sort:



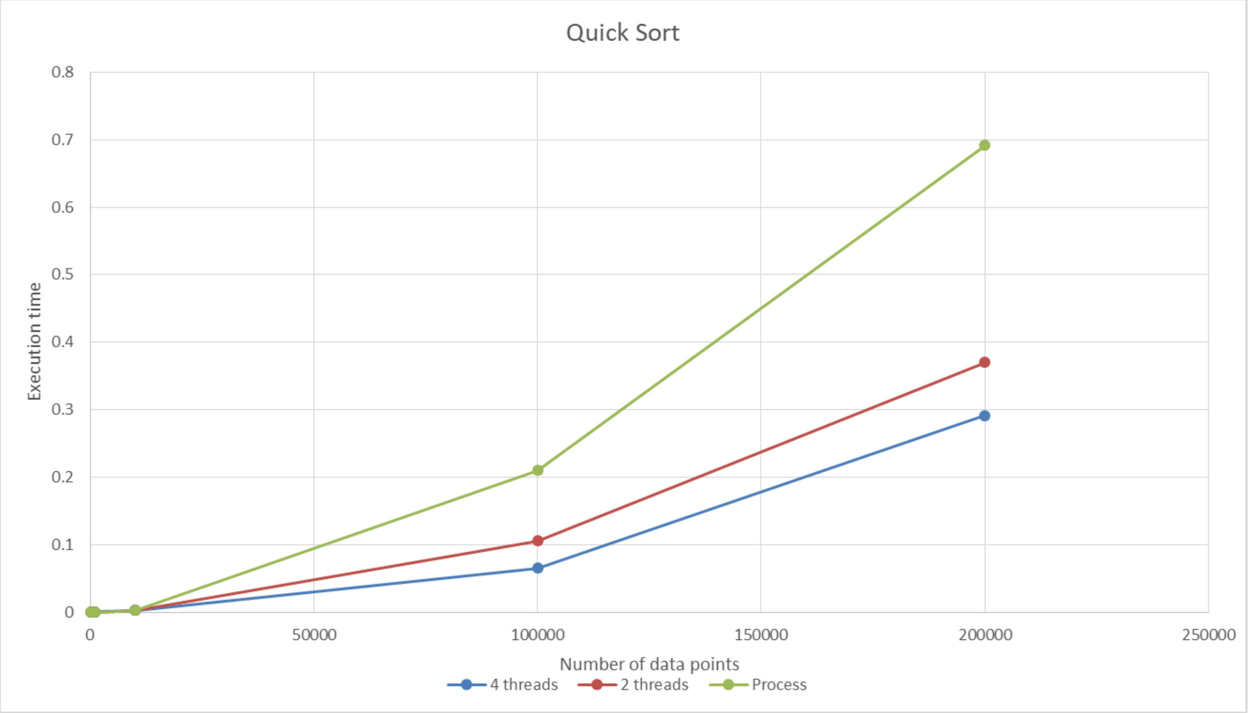
Zoomed in version from 100 to 10,000 data points:



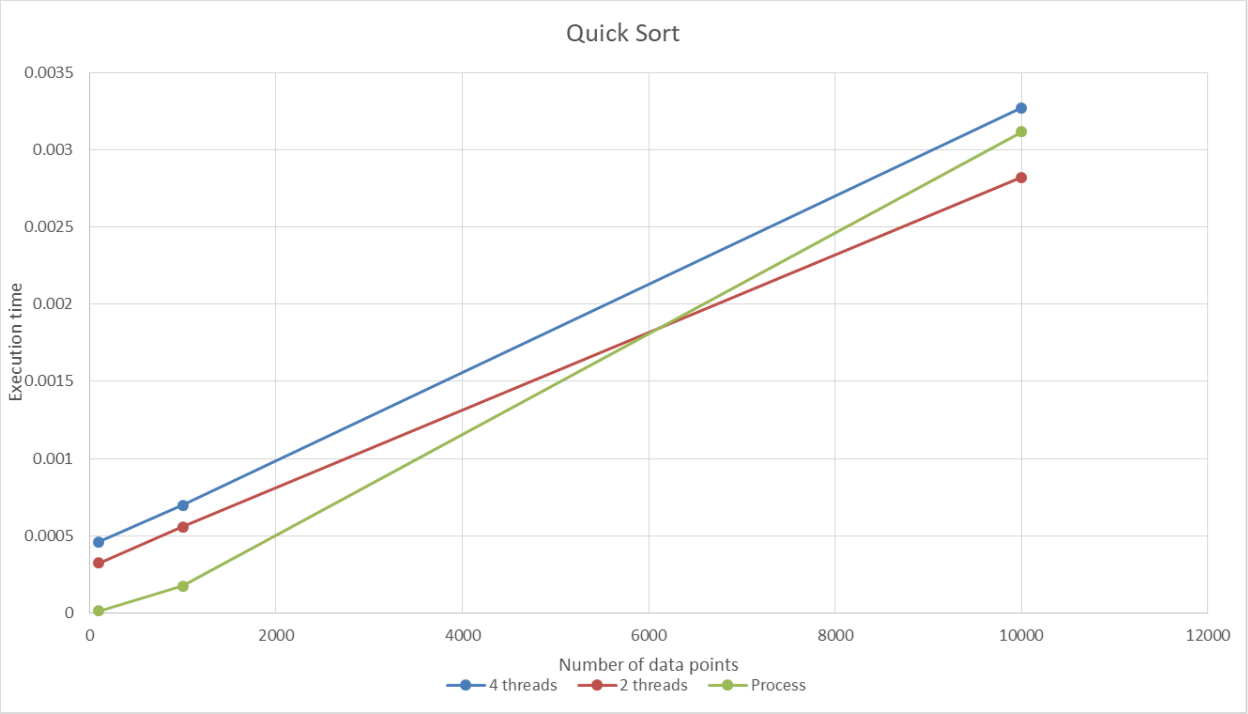
Data points	4 threads	2 threads	Process
100	0.000621	0.000329	0.000018
1000	0.001205	0.000841	0.000234
10000	0.00408	0.002762	0.002271
100000	0.039842	0.027783	0.026809
200000	0.058464	0.0558	0.054034

- The statistics and graphs above show that all processes' execution times steadily increase with dataset size.
- Two-threaded and four-threaded parallel implementations require more time than a serial operation. While two-threaded processes execute faster than four-threaded processes, the overhead of managing four threads results in longer execution times. Serial process executes the fastest out of all threaded processes.

Quick Sort:



Zoomed in version from 100 to 10,000 data points:



Data points	4 threads	2 threads	Process
100	0.000462	0.000325	0.000014
1000	0.000697	0.000557	0.000174
10000	0.003271	0.00282	0.003117
100000	0.065402	0.105956	0.210443
200000	0.291929	0.370571	0.691804

- Quick Sort performs consistently with varying thread setups and dataset sizes.
- The graphs' behavior for 100 and 1000 data points demonstrates that the CPU takes the following order of time to execute a command.

4 threaded processes > 2 threaded processes > serial process

- Additionally, there is a variance in the graph for 10,000 data points:

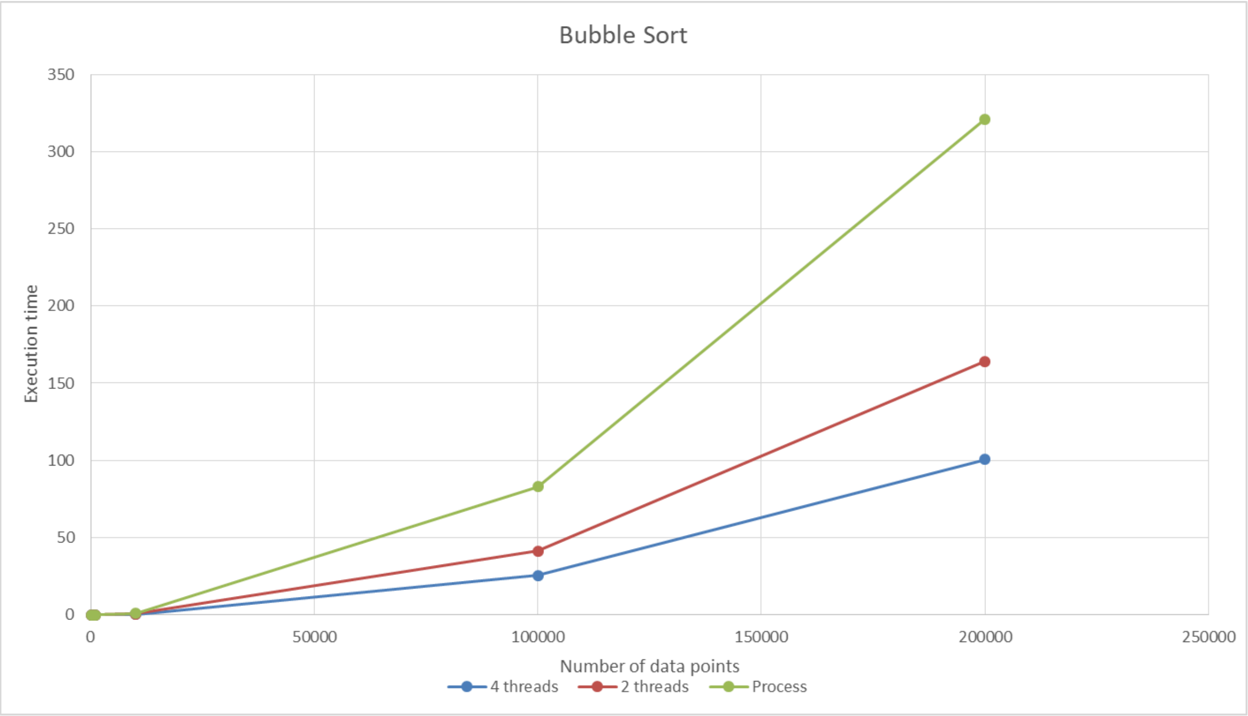
4 threaded processes > serial process > 2 threaded processes

- Behavior of graph for the data points beyond 100,000 to 200,000 have the following order of execution time:

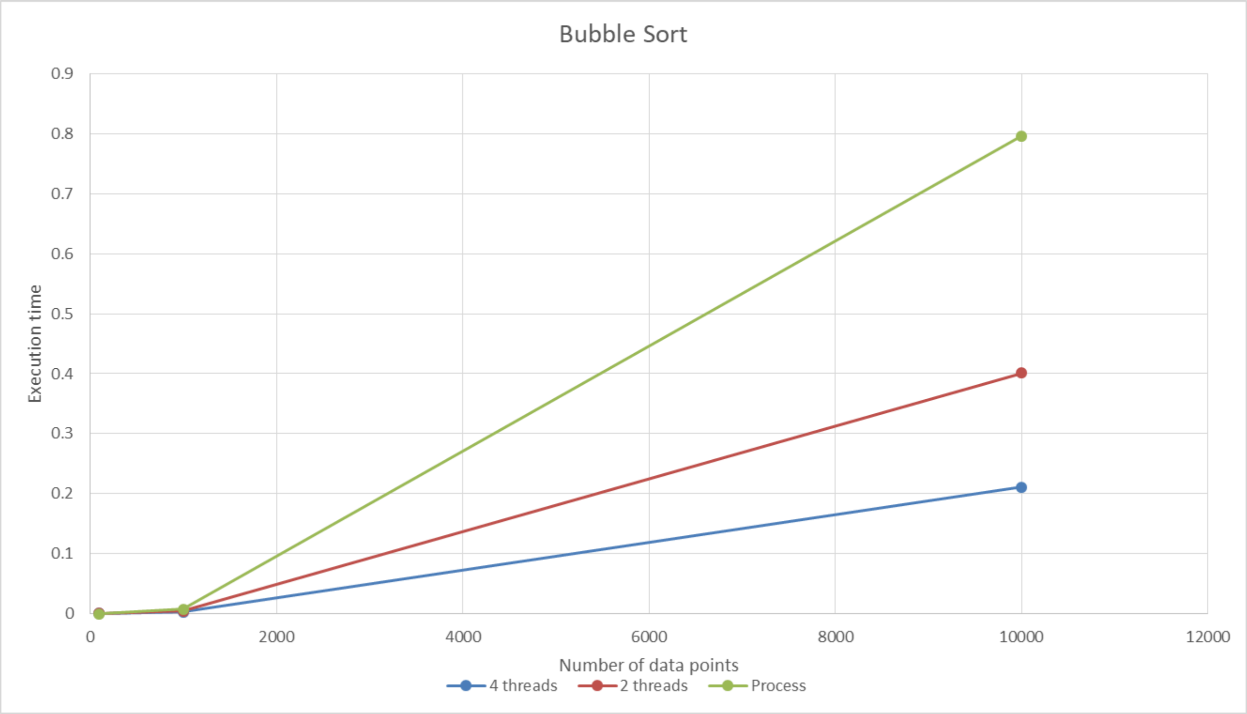
serial process > 2 threaded processes > 4 threaded processes

This shows that the benefit of threading is effectively utilized by larger data sets.

Bubble Sort:



Zoomed in version from 100 to 10,000 data points:



Data points	4 threads	2 threads	Process
100	0.000682	0.000815	0.00009
1000	0.003303	0.004564	0.007496
10000	0.210724	0.401316	0.796068
100000	25.4261	41.3799	82.7861

200000	100.438	163.997	321.11

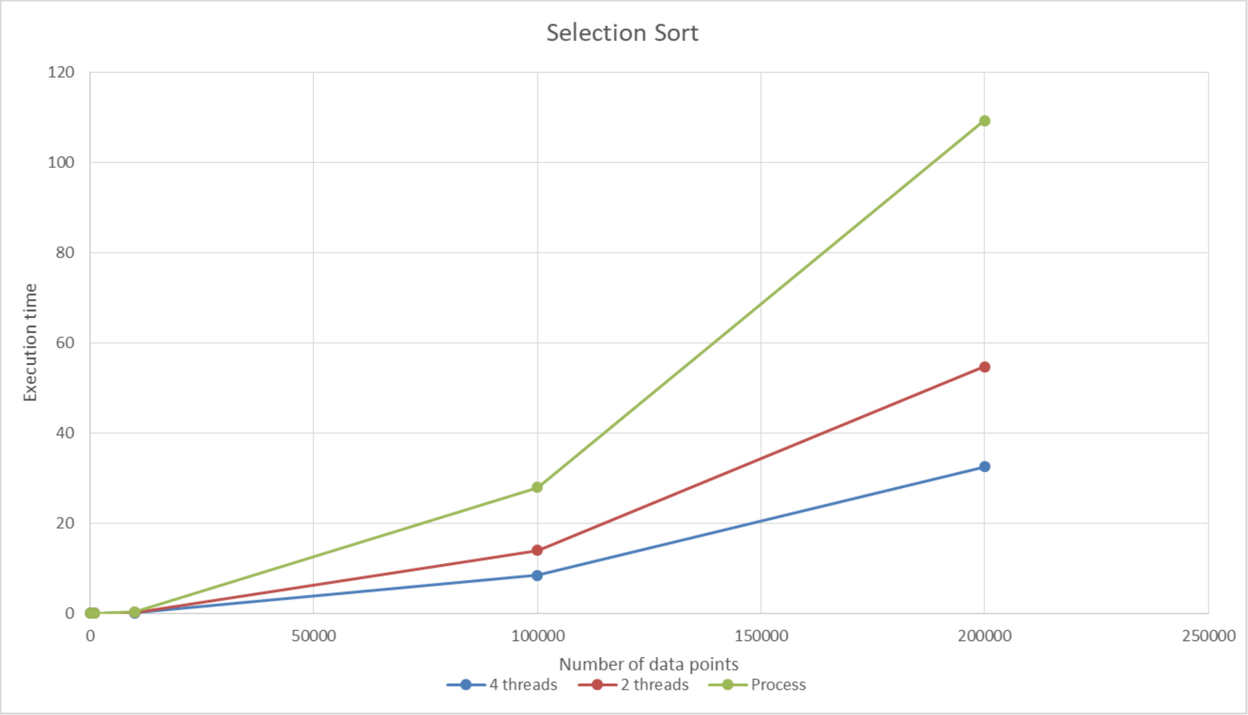
- Bubble Sort has significantly longer execution durations for all dataset sizes.
- It is consistently better to use 4 threads than 2 threads, especially when dealing with larger dataset sizes.
- The graphs' behavior for data points 1000 to 200000 demonstrates that the CPU takes the following order of time to execute a command.

serial process > 2 threaded process > 4 threaded process

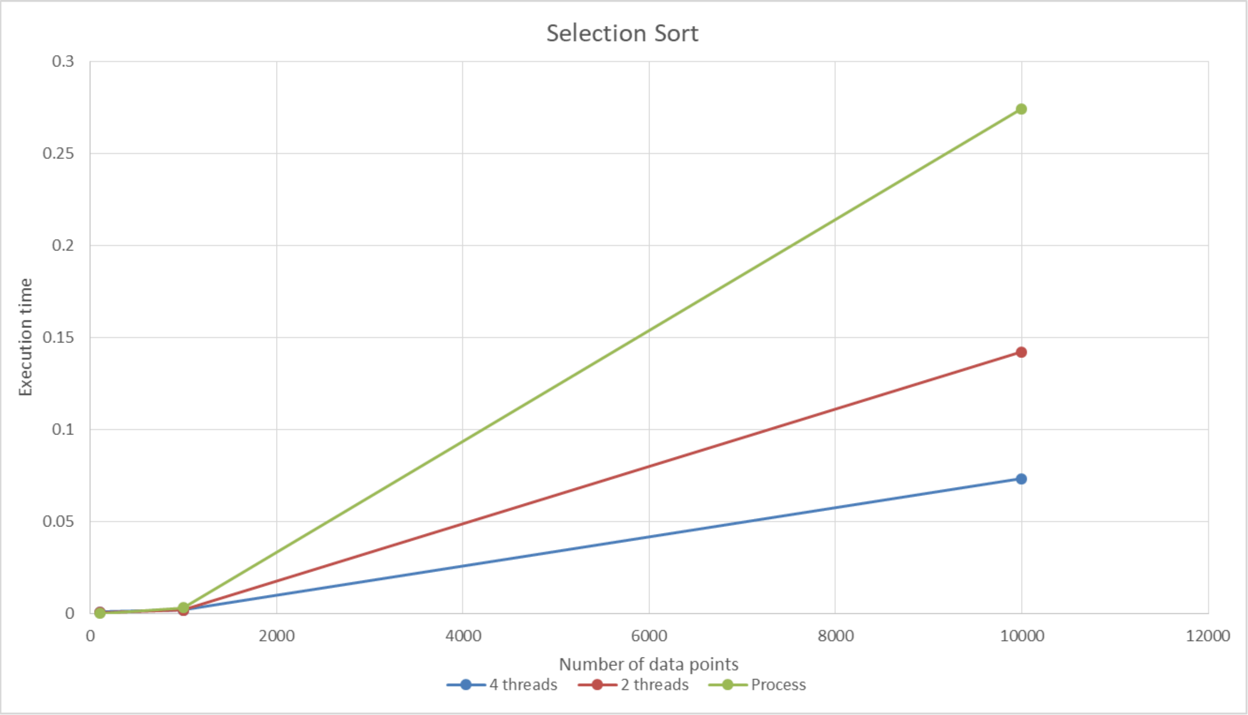
- Additionally, there is a variance in the graph for 100 data points

2 threaded process > 4 threaded process > serial process

Selection Sort:



Zoomed in version from 100 to 10,000 data points:



Data points	4 threads	2 threads	Process
100	0.000624	0.000335	0.000036
1000	0.001726	0.001695	0.00301
10000	0.073278	0.14201	0.274177
100000	8.36733	13.9883	27.9023
200000	32.4899	54.7246	109.339

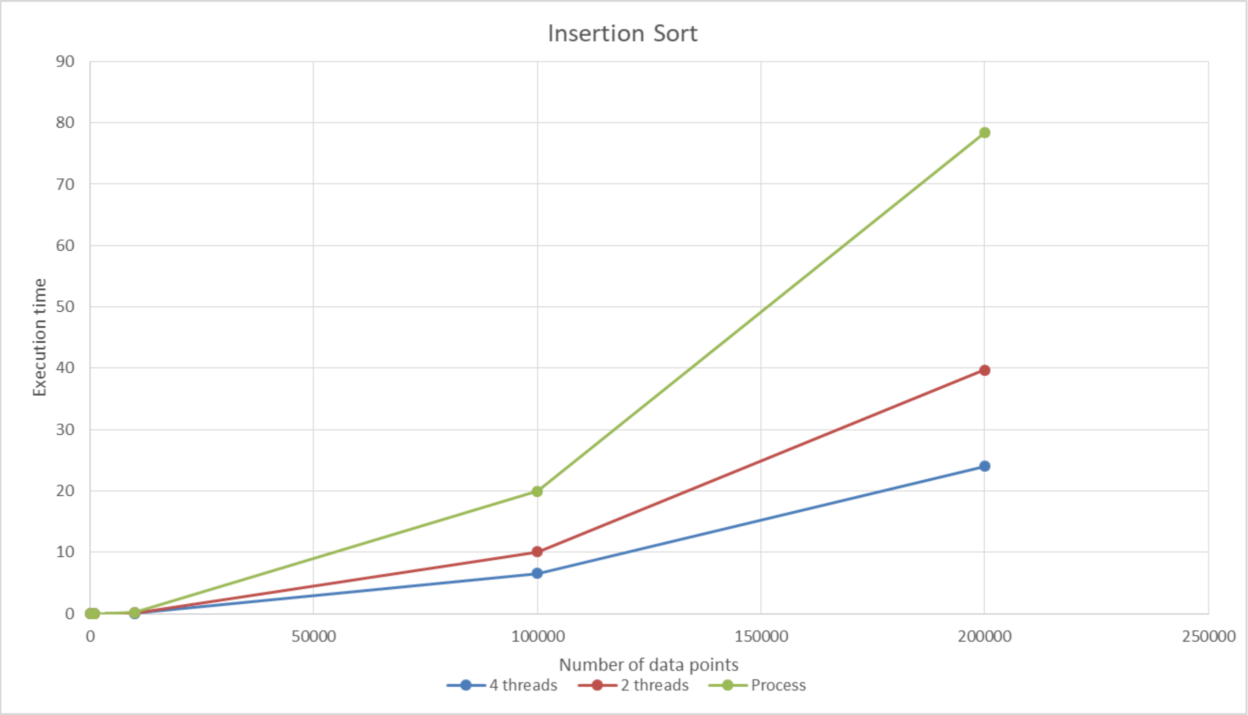
- Selection Sort also shows increased execution times as the dataset size grows.
- The execution times with 4 threads are smaller than with 2 threads for largest dataset sizes.
- There is a variation in start for smallest data points. The graphs' behavior for data points is following:
- For 100 data points:
  4 threaded processes > 2 threaded process > serial processes

- For 1000 data points:
  serial processes > 4 threaded process > 2 threaded processes

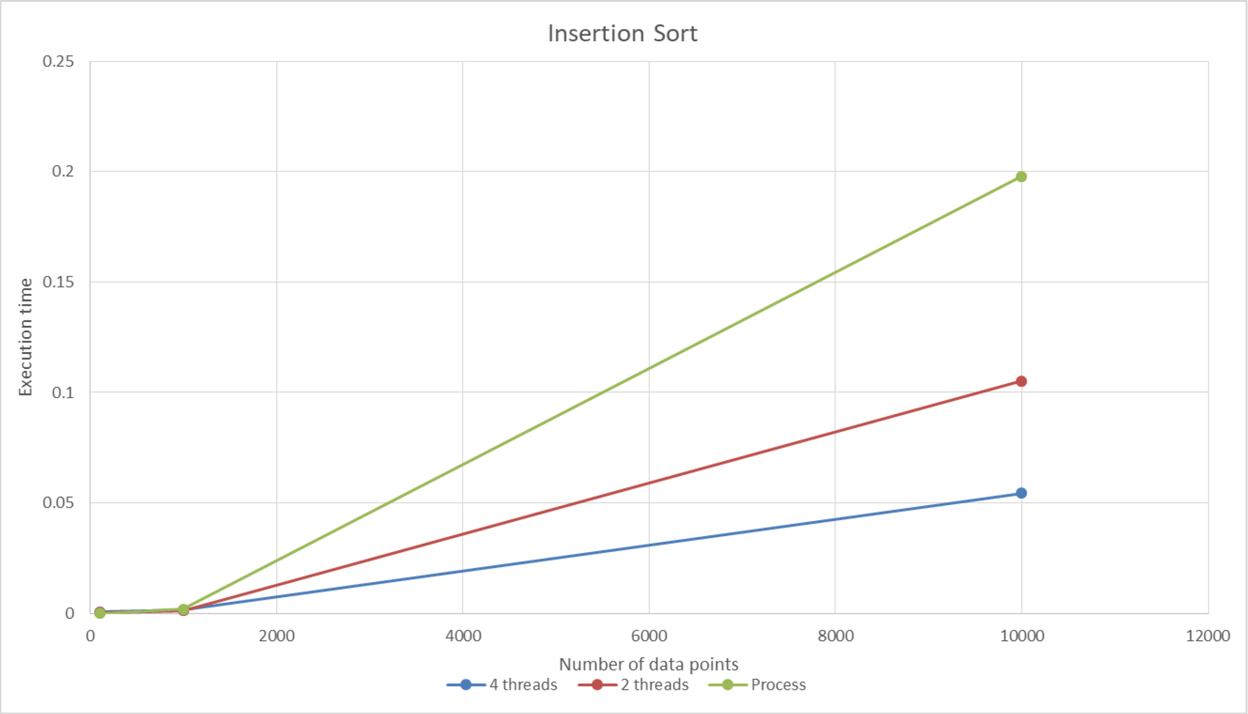
- For the rest of the data sets:
  serial processes > 2 threaded process > 4 threaded processes

- Despite the variations, using more threads tends to lead to reduced execution times, especially for larger datasets.

Insertion Sort:



Zoomed in version from 100 to 10,000 data points:



Data points	4 threads	2 threads	Process
100	0.000519	0.000423	0.000022
1000	0.001343	0.001273	0.001954
10000	0.054278	0.10511	0.197998

100000	6.55457	10.0964	19.9595
200000	23.9871	39.6758	78.3515

- With a small dataset (100 and 1000 data points), the execution times are relatively low, and the differences between using 2 threads and 4 threads are minimal.
- However, as the dataset size increases (to 10000 and 100000 data points), the execution times increase significantly. Despite this, using 4 threads consistently shows improvement over using only 2 threads.

Conclusion:

Through our project, we have observed distinct trends in the efficiency of threading across different sorting algorithms. Notably, threading does not enhance the efficiency of Merge Sort, hence, implementing it as a serial process is more effective. Conversely, we've found that the benefits of threading are more pronounced with larger datasets, since with smaller datasets, the overhead of managing multiple threads can outweigh the potential gains from parallel execution. This observation underscores the importance of considering dataset size when deciding whether to employ threading. Additionally, as the number of threads increases, particularly with larger datasets, we observe a decrease in execution time, indicating enhanced efficiency. Therefore, it is advisable to apply threading selectively, focusing primarily on larger datasets to maximize performance gains.