**NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES-FAST**

**KARACHI CAMPUS**



**SUPERVISOR NAME:**  
**MISS. ANAUM HAMID**

**CO SUPERVISOR NAME:  
RABIA ANSARI**

**PROJECT NAME:**

**COMPARISON BETWEEN PROCESS AND THREADS**

**SECTION: C**

**GROUP MEMBERS:**

**Laiba Zaheer(k180311)**

**Abeer Fatima(k180304)**

**Ruba Waqar(k181099)**

**OBJECTIVE:**

Our project will mainly focus on the differences between multithreading and multiprocessing. The performance comparison will be observed by the implementation of merge sort, count sort and Insertion sort and radix sort.

**INTRODUCTION:**

We will show the comparison by implementing sorting algorithm using process and threads. We will compare the performance of this sorting algorithm with respect to time, number of inputs in relation with multiple threads and multiple processes.

**HOW PROJECT STARTED:**  
1. Sorting Algorithms to be implemented were selected.  
2. Then project was divided into 2 parts   
- Algorithms for Threads were implemented.  
- Algorithms for Process were implemented.

3. Comparison was observed between Threads and Process.

**PROBLEMS FACED:**

Merge sort was divided in multithreads into two threads. Merge Sort was throwing errors for different number of input, whereas it was working perfectly fine for few inputs.

In count sort after taking input, returning array was creating unknown error.

**SOLUTION TO THE PROBLEMS:**

In Merge sort to resolve error, the array was divided into 2 parts. First divided part is implemented in 1 thread. And other part of array was implemented in other thread.

Error of count sort was resolved by calling SHM library, which enables the option to access shared array and updates that array after every operation.

**METHODOLOGY ( PROJECT DESCRIPTION ):**

* **MERGE SORT:**

For a single thread and a single process, we implemented a single of merge sort.

For multithreading, we have used the library of pthread to create multiple threads and have joined the threads after merging of split parts while applying merge sort, whereas, for process, we have used the ipc and shm library with fork function to create 2 child processes then we called merge sort for both the processes on equal halves of number of elements attaching shared memory so that both processes concurrently work on the same memory space. Their sorted results were then merged and the shared memory was detached.

to split the number of elements in equal halves to make them sorted through merge sort technique with shared memory implemented through ipc and shm library functions. We tested out our codes on the following data and the results are tabulated as follows:

|  |  |  |
| --- | --- | --- |
| Number of elements | Time (in seconds) taken by | |
| Process | Thread |
| 1000 | 0.000104 | 0.000268 |
| 7500 | 0.000264 | 0.0001719 |
| 10000 | 0.000168 | 0.003984 |
| 75000 | 0.000688 | 0.020428 |
| 100000 | 0.000751 | 0.027651 |
| 750000 | 0.005112 | 0.214430 |
| 1000000 | 0.006117 | 0.288281 |

**Code:**



* **Count Sort:**

For threading in count sort, we have used the library of pthread to create single thread while applying count sort, whereas, for process, we have used the ipc and shm library with fork function to create 1 child process and 1 parent process then we called count sort for both the processes attaching shared memory so that both processes concurrently work on the same memory space. After getting the desired results the shared memory was detached.

The second part was sorted through count sort technique with shared memory implemented through ipc and shm library functions. We tested out our codes on the following data and the results are tabulated as follows:

|  |  |  |
| --- | --- | --- |
| Number of elements | Time (in seconds) taken by | |
| Process | Thread |
| 50 | 0.000054 | 0.00098 |
| 100 | 0.000231 | 0.00107 |
| 100001 | 0.000131 | 0.001271 |
| 1000001 | 0.000072 | 0.011287 |

**Code:**



* **Insertion Sort:**

For a single thread and a single process, we implemented a single of insertion sort.

For multithreading, we have used the library of pthread to create multiple threads while applying insertion sort with the help of merge sort , whereas, for process, we have used the ipc and shm library with fork function to create 1 child processes and 1 parent process then we called insertion sort and merged them for both the processes on equal halves of number of elements attaching shared memory so that both processes concurrently work on the same memory space. Their sorted results were then merged and the shared memory was detached.

|  |  |  |
| --- | --- | --- |
| Number of elements | Time (in seconds) taken by | |
| Process | Thread |
| 1000 | 0.000105 | 0.000963 |
| 7500 | 0.000108 | 0.018931 |
| 10000 | 0.000150 | 0.068995 |
| 75000 | 0.00086 | 3.707761 |
| 100000 | 0.00074 | 6.552257 |
| 120000 | 0.000142 | 9.449598 |
| 200000 | 0.000157 | 26.351427 |

**Code:**

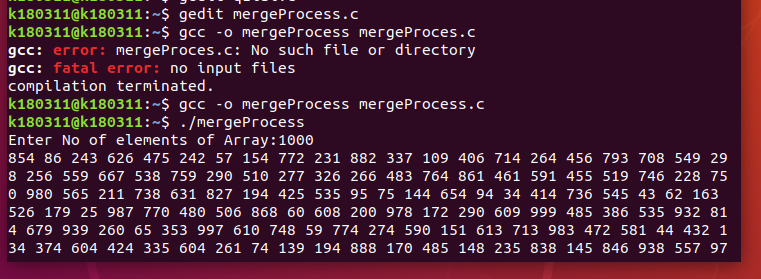


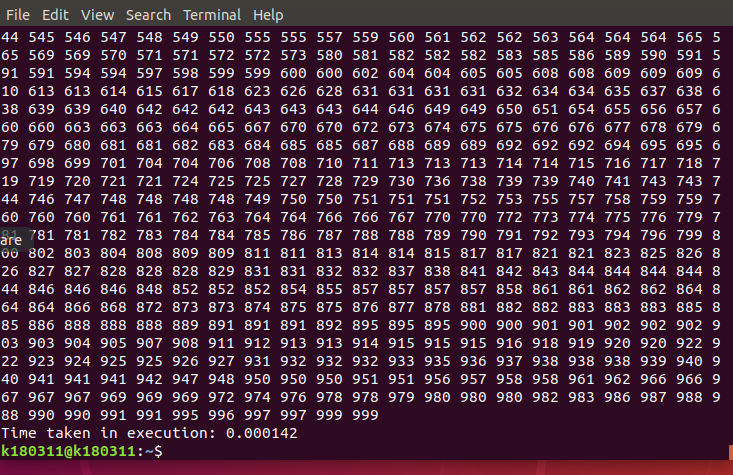
**PROJECT ASPECT IN FUTURE TECHNOLOGY:**

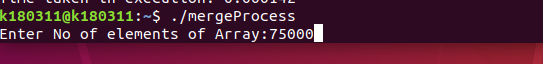
Since modern architectures support both process and threads in parallel. This project is simply checking the time required for process and thread to run an algorithm and sort elements. The algorithm’s better performance in threads can be implemented in future architectures using threads. The algorithm’s having efficiency using process can be implemented using via process.

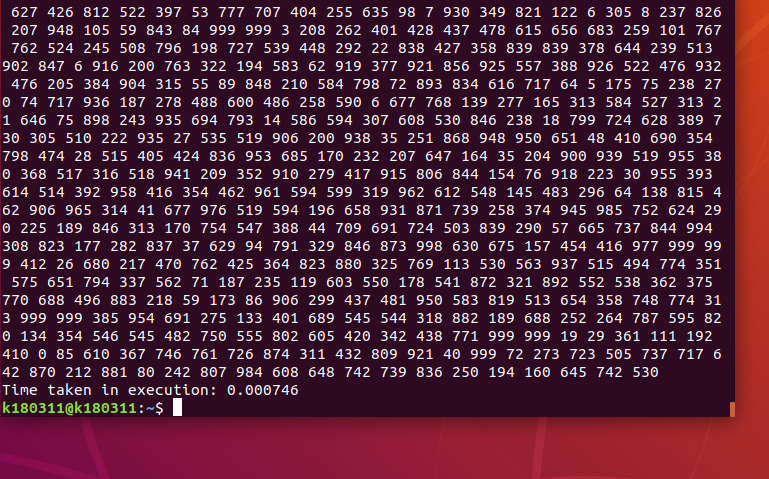
**OUTPUT SNIPPETS:**

* **Merge Sort:**

**PROCESS:**

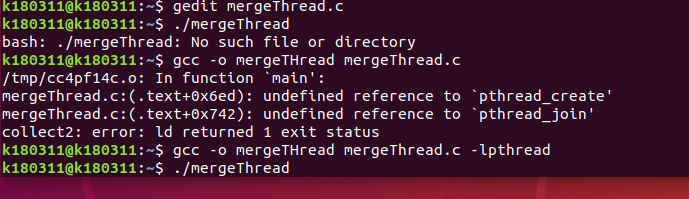
****

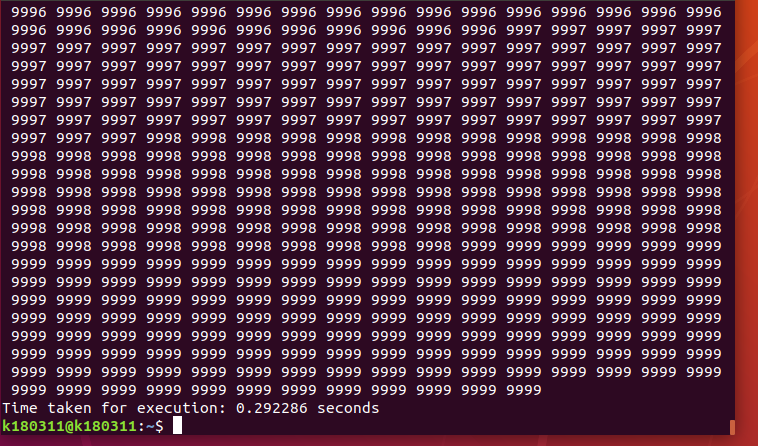
****

****

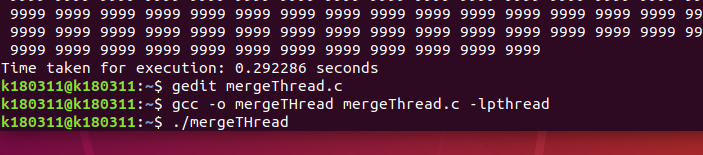
**THREADS:**

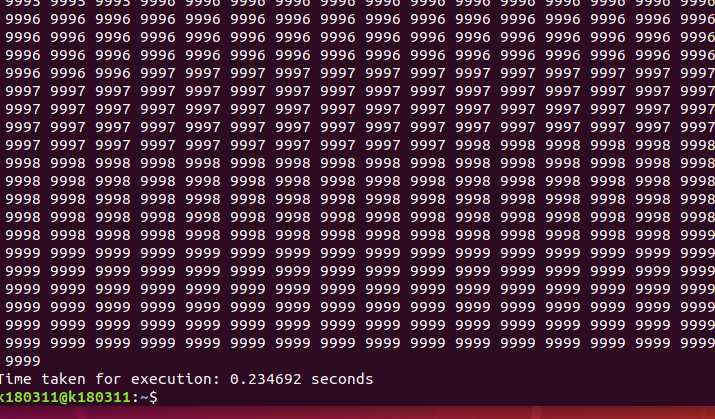
**For No of elements:**100000

****

****

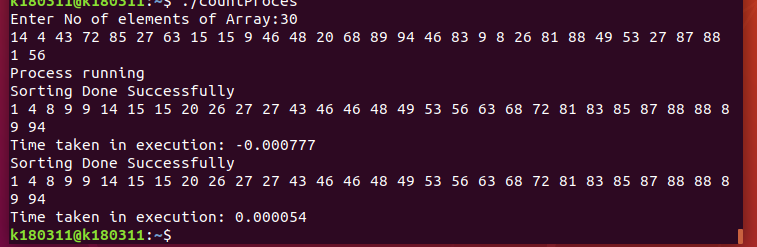
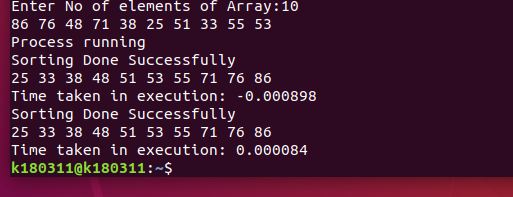
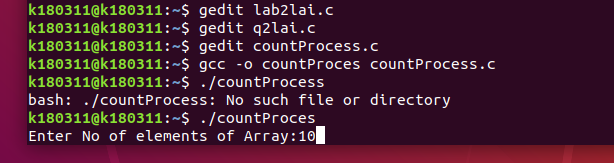
**For No of elements** :75000

****

****

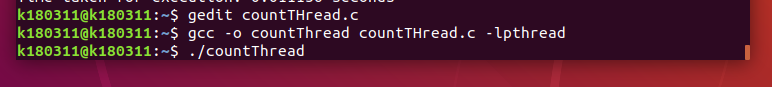
* **Count Sort:**

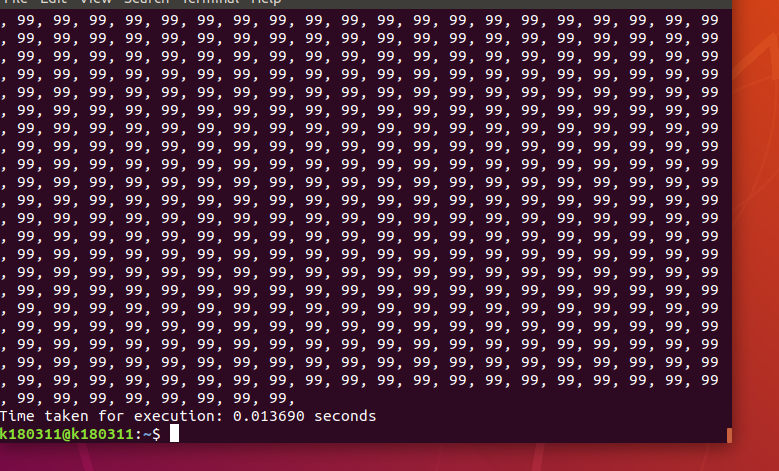
**PROCESS:**

****

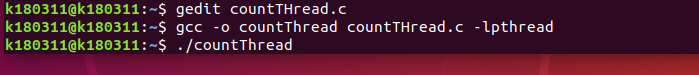
**THREADS:**

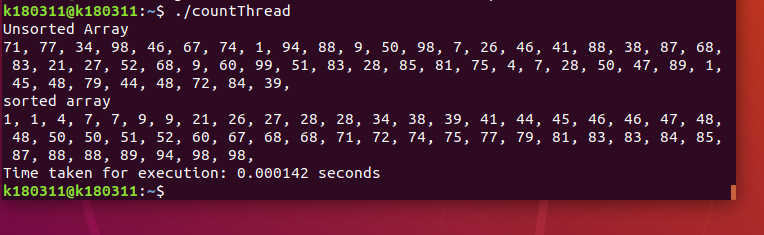
**For Input:1000**

****

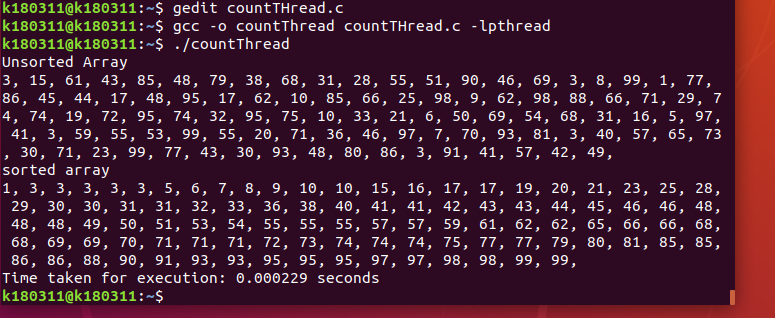
****

**For No of elements:50**

****

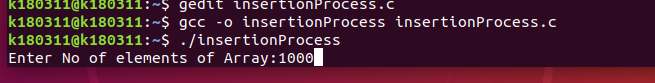
****

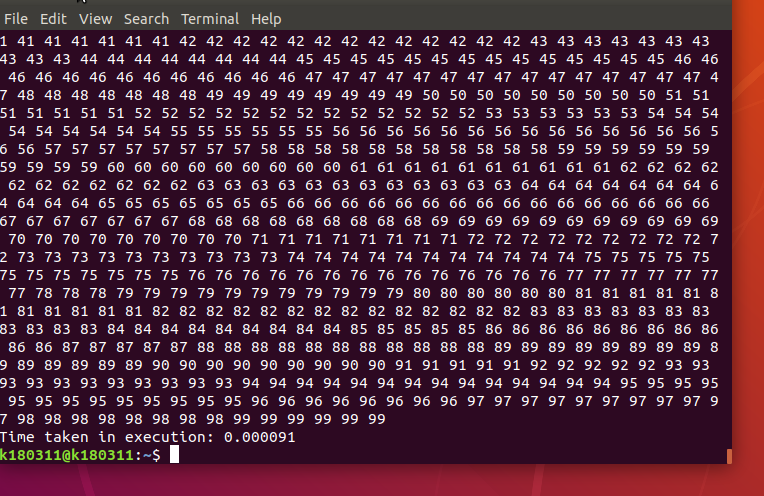
**For No of elements:100**

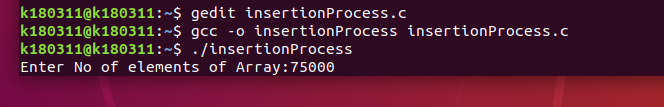
****

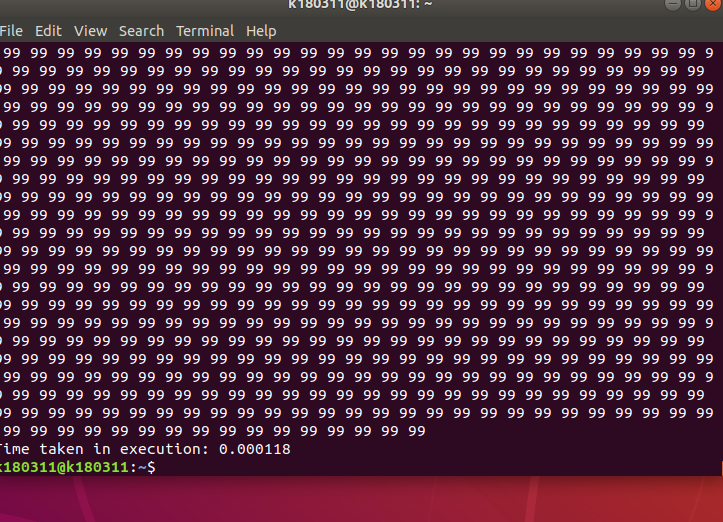
* **Insertion Sort:**

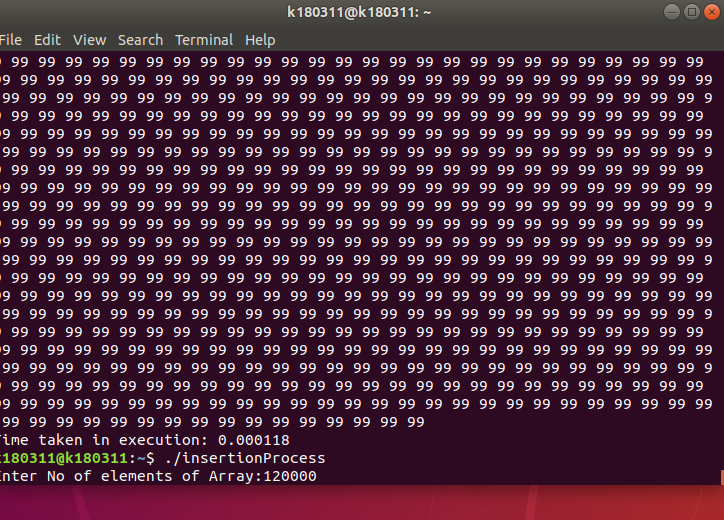
**PROCESS:**

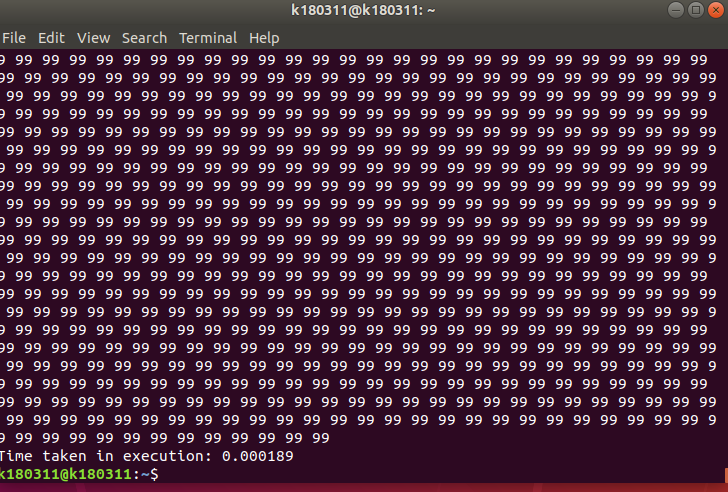
****

****

****

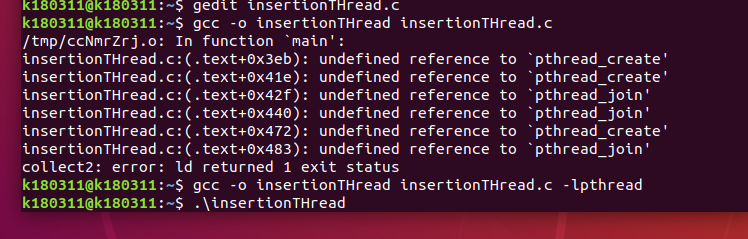
****

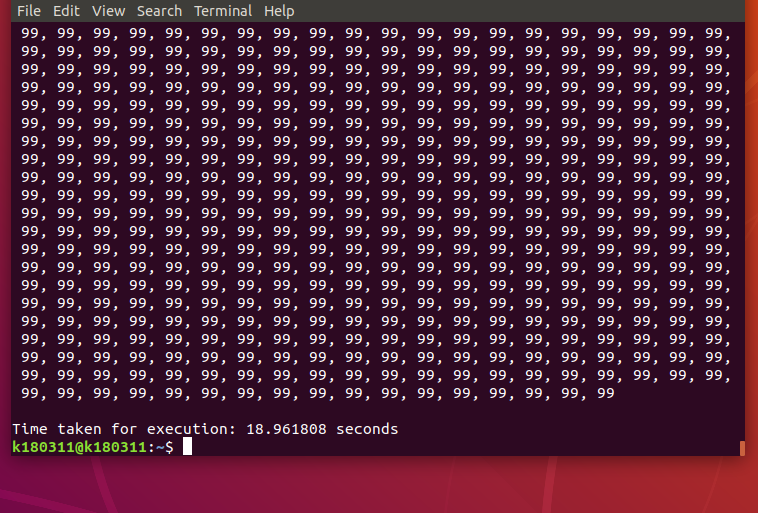
****

****

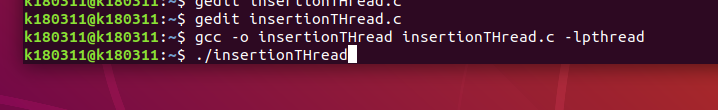
**THREADS:**

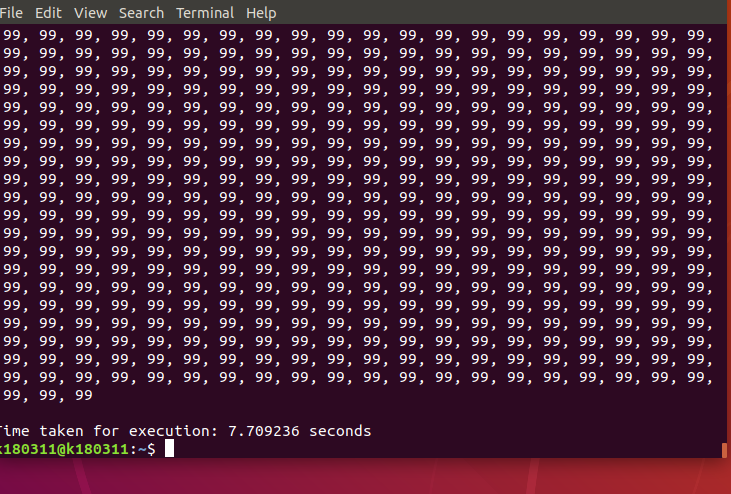
**For No of elements:200000**

****

****

**For No of elements:12000**

****

****

**CONCLUSION:**

The conclusion which can be drawn from the Comparison Chart is that:

* Merge sort is working faster in Multiprocessing as compared to multithreading. The reason is that, multithreading is creating an overhead of synchronization of the threads, secondly as merge sort works on divide and conquer rule so we have to merge all the parts at the end which is time consuming and multithreading works faster on heavy computations.
* Count Sort is working faster in process as compared to threads because process does not require heavy computations and also require less memory.
* Insertion Sort overall is working faster in process as compared to threads. It is because it's quicker for the OS to stop one thread and start running another than do the same with two processes

.