

INFO 6205
Program Structures & Algorithms
Summer Full 2018
Assignment 5

In this Assignment, I have to implement a parallel sorting algorithm such that each partition of the array is sorted in parallel.

Key Points:

- ➔ Future Completion class: This class implements the *Future* interface, so we can **use it as a *Future* implementation, but with additional completion logic.**

For example, we can create an instance of this class with a no-arg constructor to represent some future result, hand it out to the consumers and complete it at some time in the future using the *complete* method. We may use the *get* method to block the current thread until this result will be provided.

In the example, below we have a method that creates a *CompletableFuture* instance, then spins off some computation in another thread and returns the *Future* immediately.

When the computation is done, the method completes the *Future* by providing the result to the *complete* method.

- ➔ Here, I am also using a cut-off which I am updating according to user requirement by scanning it. If there are fewer elements to sort than the cut-off, then I am using the system sort instead.

NOTE: Number of Threads used depends on machine which you are using.

When I checked the number of cores available for parallel processing in java. Result was 4.

Out of 4, one is used by main method. Further recursively, two arrays will be formed which will use two threads. It will be divided till end till cut-off value is reached.

Only two parallel threads can run together as in my case I still have one available thread but it can't do any parallel processing as no other core is available.

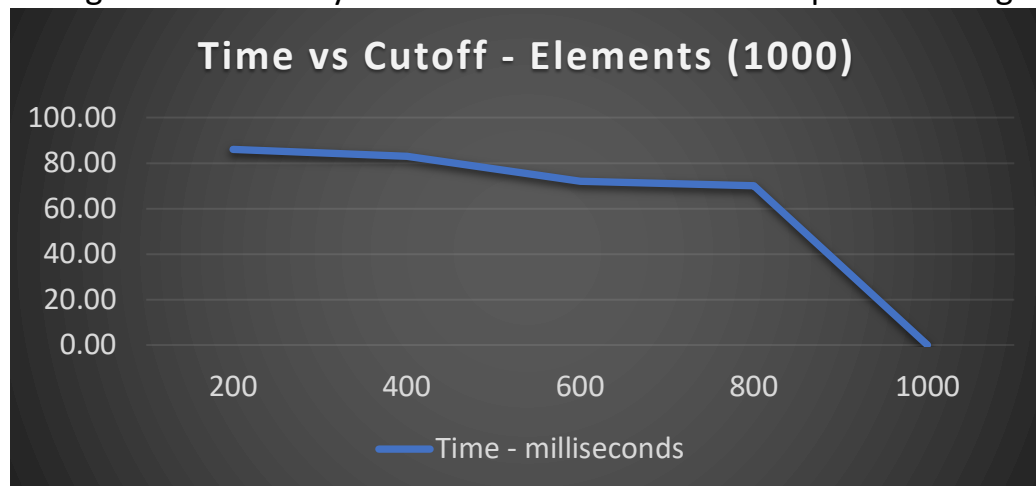
Analysis:

- ➔ **Changing cut-off and keeping the size of array same.**

Cut-off	Number of Elements	Time - milliseconds	Number of Threads Created
200	1000	86.00	13
400	1000	83.00	5
600	1000	72.00	2
800	1000	84.00	1
1000	1000	0.00	0

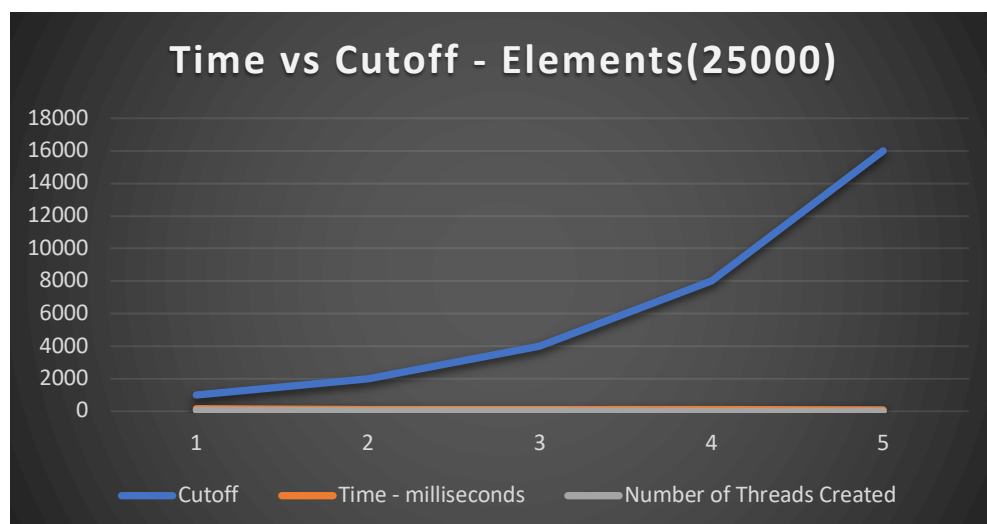
Number of threads created will wait for the cores to complete. They will be in waiting state. As soon as threads gets completed new pair of threads will run.

In above result, when size of array is 1000 and cut-off is 600. Only two threads are generated which is taking very less time as machine configuration has only three cores available for multiple threading.



Similarly, in the next experiment I am taking very high number of elements (25000) and doubling the cut off.

Cut-off	Number of Elements	Time - milliseconds	Number of Threads Created
1000	25000	157.00	62
2000	25000	97.00	30
4000	25000	96.00	14
8000	25000	92.00	6
16000	25000	85.00	2

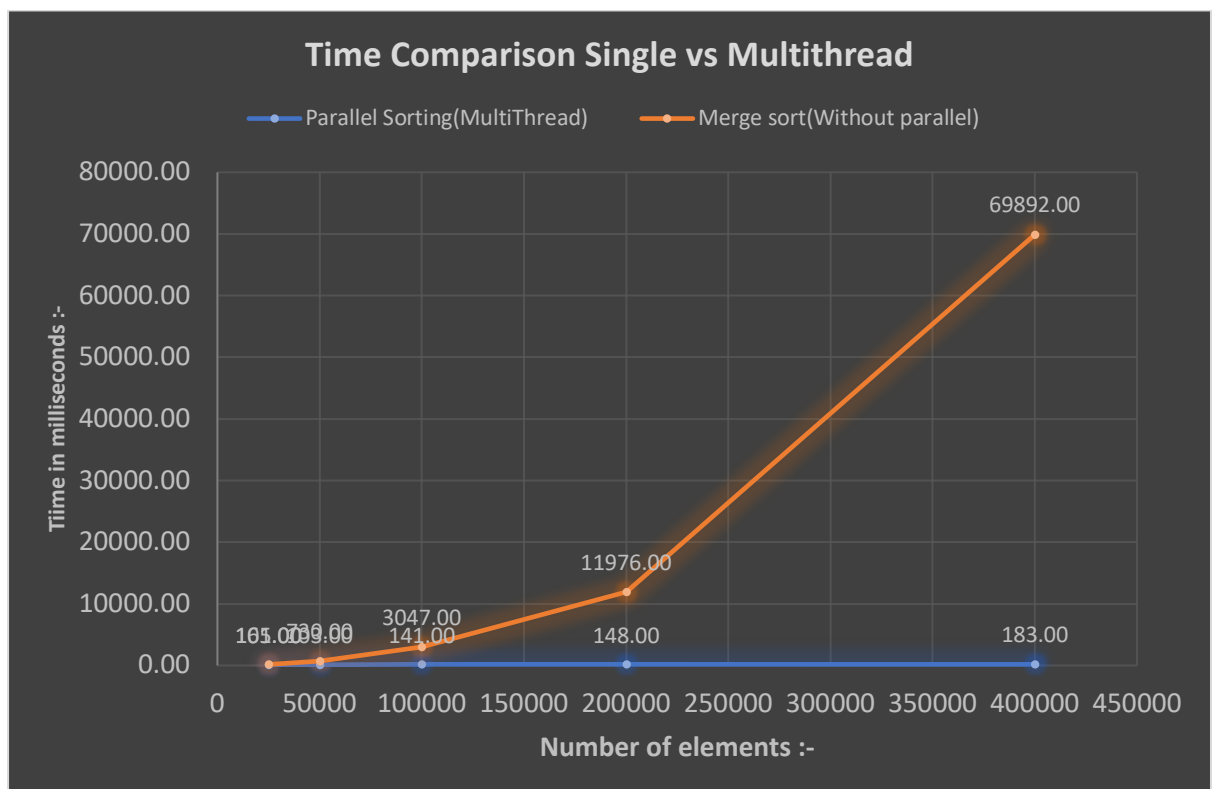


3. In this analysis , doubling the number of elements and cut-off in each step to achieve the time analysis between the single thread and multiple thread sorting.

Cut-off value will be used only in case of parallel sorting (Multiple threading).

Single thread Merge sort is the traditional approach running on the single thread.

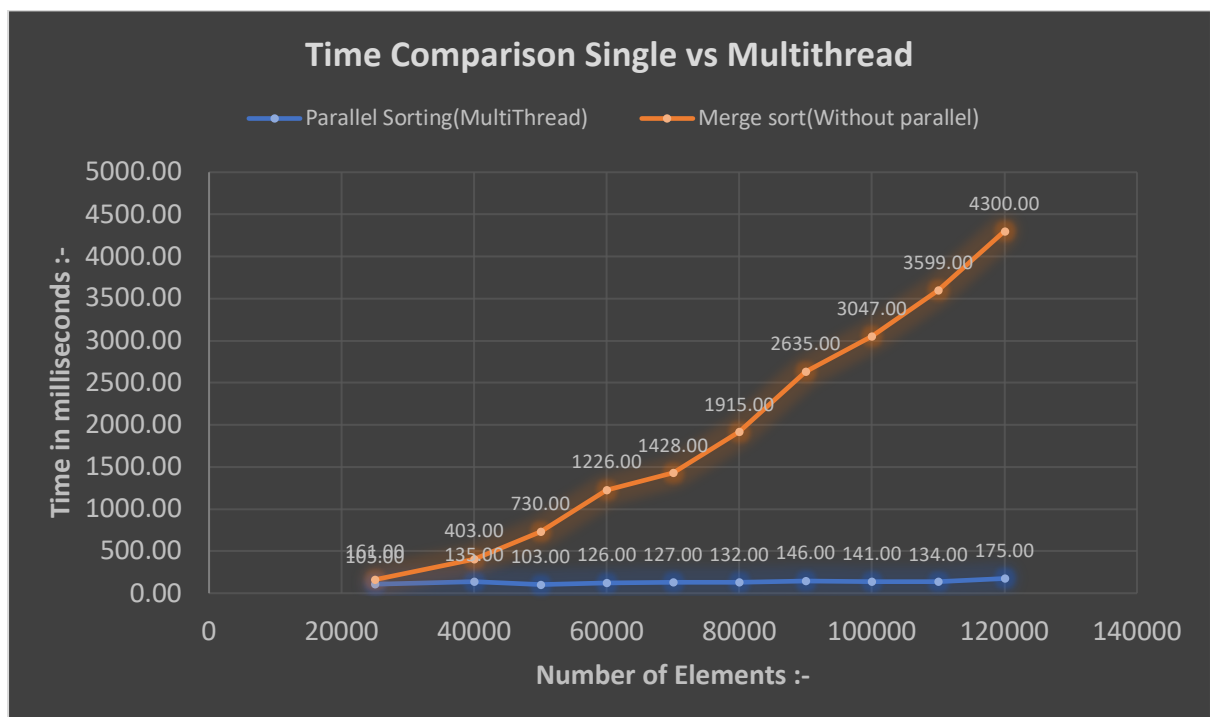
Cut-off	Number of Elements	Time - milliseconds	
		Parallel Sorting(Multi-Thread)	Single Thread Merge Sort
500	25000	105.00	161.00
1000	50000	103.00	730.00
2000	100000	141.00	3047.00
4000	200000	148.00	11976.00
8000	400000	183.00	69892.00
16000	800000	253.00	384592.00



Here, we can prove that the time taken by parallel sorting is way much lower than the single thread sorting. As we increase the number of elements the time taken by the linear i.e., single sorting i.e., non- parallel sorting is taking very large time as compared to Parallel sorting.

2.

Cut-off	Number of Elements	Time - milliseconds	
		Parallel Sorting(Multi-Thread)	Single Thread Merge sort
500	25000	105.00	161.00
750	40000	135.00	403.00
1000	50000	103.00	730.00
1250	60000	126.00	1226.00
1500	70000	127.00	1428.00
1500	80000	132.00	1915.00
1750	90000	146.00	2635.00
2000	100000	141.00	3047.00
2250	110000	134.00	3599.00
2500	120000	175.00	4300.00



Here, we can prove that the time taken by parallel sorting is way much lower than the single thread sorting. As we increase the number of elements the time taken by the linear i.e., non-parallel sorting is taking very large time as compared to Parallel sorting.

Conclusion:

- I have noticed that it is better to do multithreading only when we have large number of elements.
- If you do multithreading with small numbers then time taken is larger than the normal sorting.
- If the number of elements is very large then, multithreading consumes very small time as compared to the normal non-parallel sorting.
- It is observed that for small arrays system sort works optimum due to the thread creation time being skipped.
- For large arrays this is not the same. The threads help in optimization. Recursion depth or available threads is calculated based on the threads created during parallel sort and the ideal number for this parameter is calculated just before system sort is called.

So, No. of elements very large:

$$\text{Time(Parallel)} < \text{Time (Non – parallel sort)}$$

If no. of elements small

$$\text{Time(Parallel)} > \text{Time (Non – parallel sort)}$$

Optimized Value:

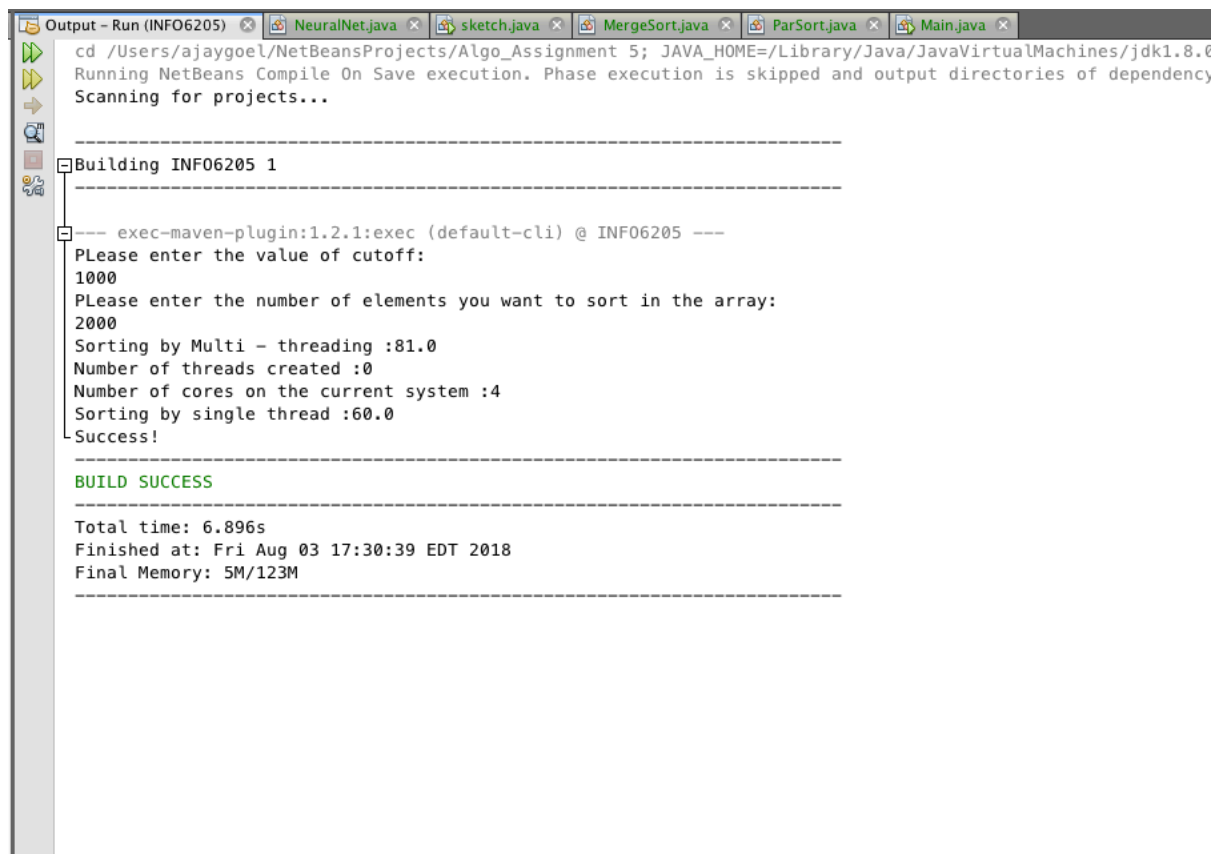
Cut-off	Number of Elements	Time - milliseconds	Number of Threads Created	Cut-off/Array
200	1000	86.00	13	0.2
400	1000	83.00	5	0.4
600	1000	72.00	2	0.6
800	1000	70.00	1	0.8
1000	1000	0.00	0	1

It is observed that 0.8 is the optimum value of threads/depth for cut-off/array size for arrays

Proof:

Screenshot when cut-off =1000 and elements in array = 2000

Goel, Ajay: Nu Id (001897443)



```
Output - Run (INFO6205) x NeuralNet.java x sketch.java x MergeSort.java x ParSort.java x Main.java x
cd /Users/ajaygoel/NetBeansProjects/Algo_Assignment 5; JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0
Running NetBeans Compile On Save execution. Phase execution is skipped and output directories of dependency
Scanning for projects...

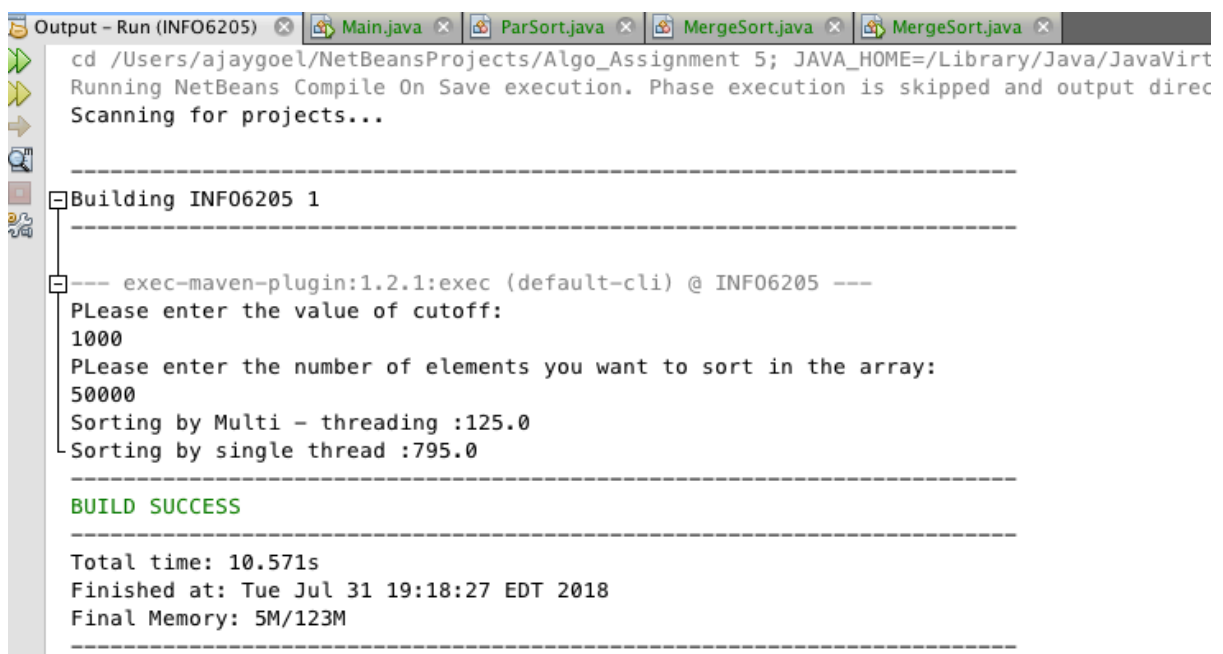
-----
Building INFO6205 1
-----
--- exec-maven-plugin:1.2.1:exec (default-cli) @ INFO6205 ---
Please enter the value of cutoff:
1000
Please enter the number of elements you want to sort in the array:
2000
Sorting by Multi - threading :81.0
Number of threads created :0
Number of cores on the current system :4
Sorting by single thread :60.0
Success!

BUILD SUCCESS

Total time: 6.896s
Finished at: Fri Aug 03 17:30:39 EDT 2018
Final Memory: 5M/123M
```

Time taken is large by parallel sorting when array size is small.

2. When cut-off is 1000 and number of elements is 50000



```
Output - Run (INFO6205) x Main.java x ParSort.java x MergeSort.java x MergeSort.java x
cd /Users/ajaygoel/NetBeansProjects/Algo_Assignment 5; JAVA_HOME=/Library/Java/JavaVirt
Running NetBeans Compile On Save execution. Phase execution is skipped and output direc
Scanning for projects...

-----
Building INFO6205 1
-----
--- exec-maven-plugin:1.2.1:exec (default-cli) @ INFO6205 ---
Please enter the value of cutoff:
1000
Please enter the number of elements you want to sort in the array:
50000
Sorting by Multi - threading :125.0
Sorting by single thread :795.0

BUILD SUCCESS

Total time: 10.571s
Finished at: Tue Jul 31 19:18:27 EDT 2018
Final Memory: 5M/123M
```

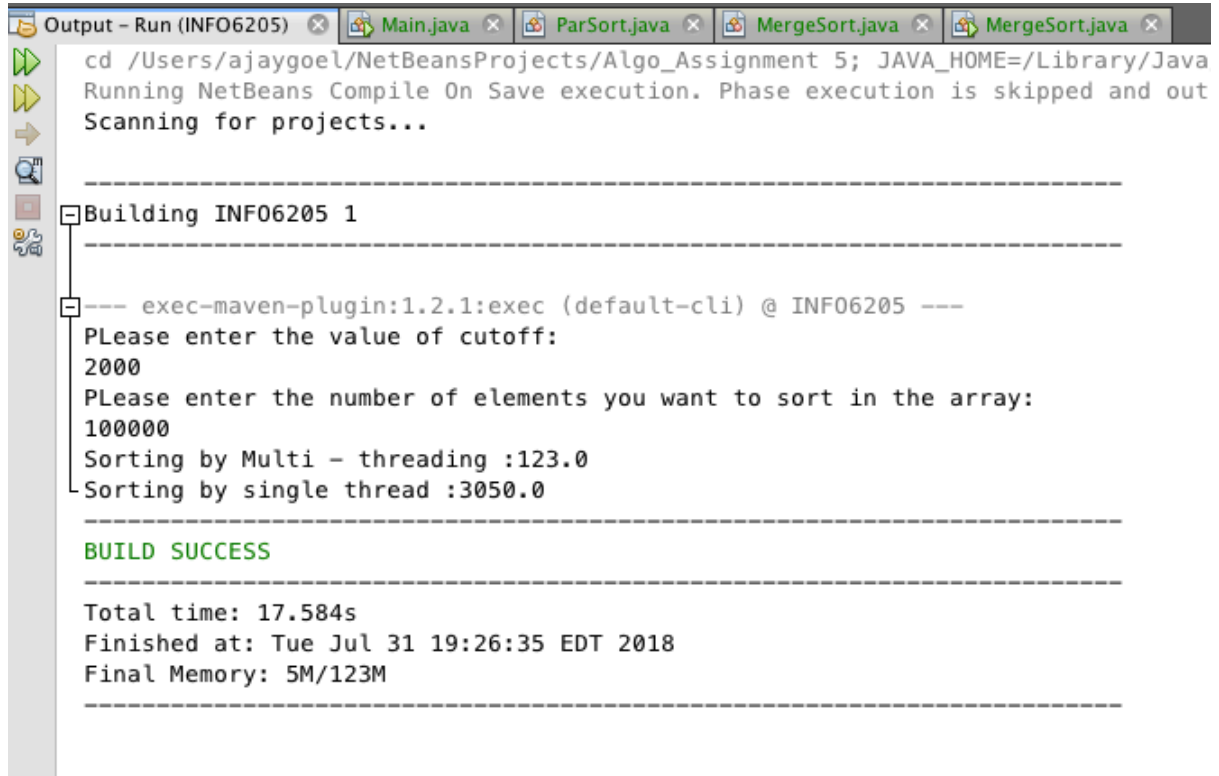
Time taken by parallel sort is less as the number of elements are very large.

SCREENSHOTS:

Running the code for 100000 elements and cutoff is 4000

Goel, Ajay: Nu Id (001897443)

Please find screenshot below:



```
Output - Run (INFO6205) x Main.java x ParSort.java x MergeSort.java x MergeSort.java x
cd /Users/ajaygoel/NetBeansProjects/Algo_Assignment 5; JAVA_HOME=/Library/Java
Running NetBeans Compile On Save execution. Phase execution is skipped and out
Scanning for projects...

-----
Building INF06205 1
-----
--- exec-maven-plugin:1.2.1:exec (default-cli) @ INF06205 ---
Please enter the value of cutoff:
2000
Please enter the number of elements you want to sort in the array:
100000
Sorting by Multi - threading :123.0
Sorting by single thread :3050.0
-----

BUILD SUCCESS
-----

Total time: 17.584s
Finished at: Tue Jul 31 19:26:35 EDT 2018
Final Memory: 5M/123M
-----
```