GTU Department of Computer Engineering
CSE 222/505 - SPRING 2022
HOMEWORK 6 REPORT

BURAK ÇİÇEK
1901042260

System Requirements

```
private BST [] table;
```
BST Array for Keep Chaining Slots.

```
private int numKeys;
```
Keep the number of Keys.

```
private static final int CAPACITY = 101;
```
Initial Capacity of Chaining HashMap

```
private static final double LOAD_THRESHOLD = 3.0;
```
For rehashing Load threshold.

```
public HashtableChain() { table = new BST[CAPACITY]; }
```

```
public V get(Object key) {
```
Get method of Chaining HashTable

```
public V put(K key, V value) {
```
Put method.

```
/** Returns the number of entries in the map */
public int size() { return numKeys; }

/** Returns true if empty */
public boolean isEmpty() { return numKeys == 0; }
```

```
public V remove(Object key) {
```
Remove method.

```
public void rehash()
```
Rehash.

For Coalesced

```
private LinkedList<EntryNext<K, V>>[] table;
```

```
/** The number of keys */
private int numKeys;
/** The capacity */
private int TABLESIZE;
```

```
public HashtableCoalesced(int tableSizeOfHash)
```
Constructor with size of table.

```
public V get(Object key)
```

```
private int findMaxPrimeNumber(int input)
```

```
public int findMaxPrimeNumber() { return findMaxPrimeNumber((int) (TABLESIZE*0.8)); }
```

```
public int hasher(int key, int ithprobe)
```
hash function

```
public V remove(Object key)
```

```
public V put(K key, V value)
```

```
private void rehash(boolean check)
```

```
public int size() { return  numKeys; }
public boolean isEmpty() { return numKeys == 0; }
public String toString() {
```
For printing the Coalesced Hashtable.

For Merge Sort:

```
public static < T
    extends Comparable < T >> void sort(T[] table)
private static < T
    extends Comparable < T >> void merge
```

For Quick Sort:

```
private static < T
    extends Comparable < T >> void quickSort(T[] table,
                                             int first,
                                             int last) {
private static < T
    extends Comparable < T >> int partition(T[] table,
                                            int first,
                                            int last) {
private static < T
    extends Comparable < T >> void swap(T[] table,
                                        int i, int j) {
public static < T
    extends Comparable < T >> void sort(T[] table) {
  // Sort the whole table.
  quickSort(table, first: 0, last: table.length - 1);
}
```

New Sort:

```
private int findMinIndex(int [] Array, int head, int tail){
private int findMaxIndex(int [] Array, int head, int tail){
public int [] newSort(int [] Array, int head, int tail){
```

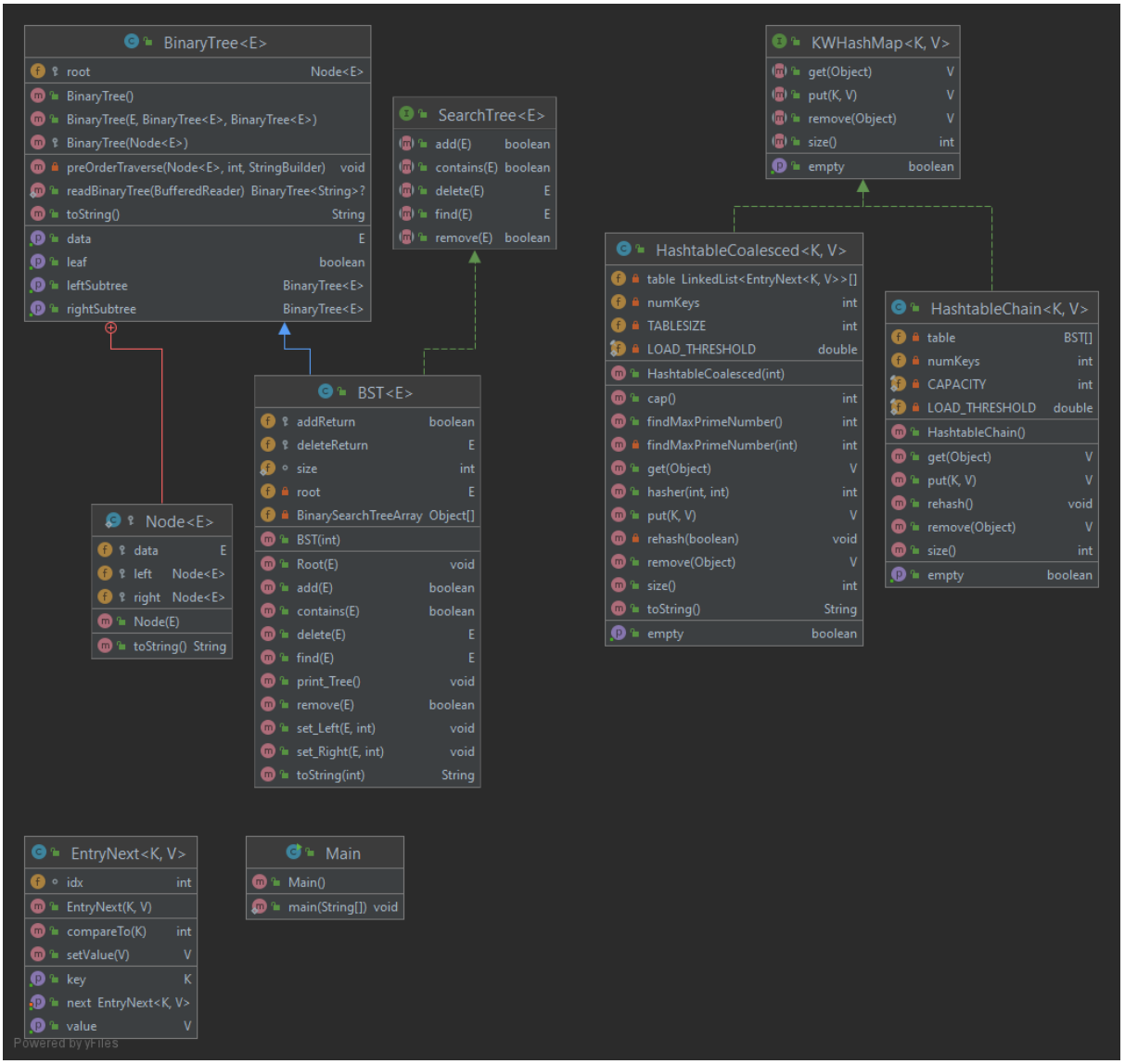
Problem Solution Approach

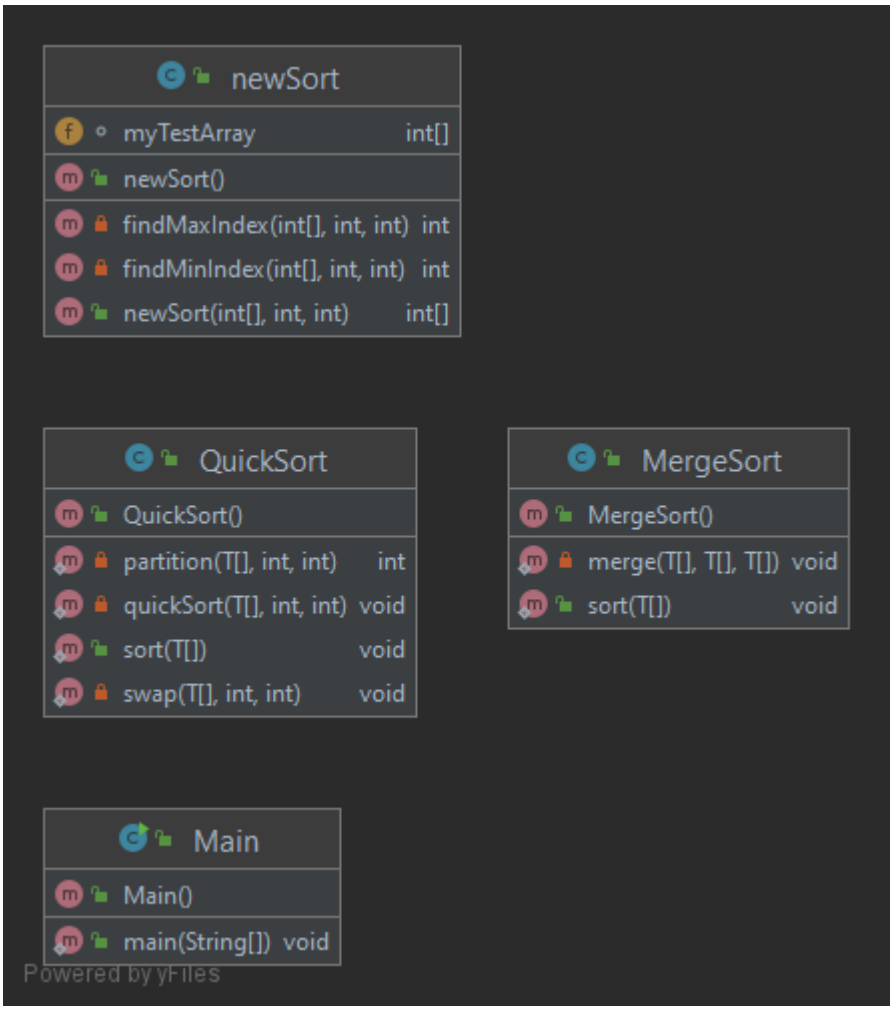
First of all, the first problem I had was to implement BST in the Chaining part. I didn't know which implementation to use. Then, I thought that we implemented Array for BST in the previous assignment and I decided to use it. Then I was able to apply the chaining rule in the desired slots by creating a BST array.

In the coalesced part, the biggest problem I had was assigning the next pointer. I can say that I spent hours and then, thanks to an algorithm that keeps the prev state, I was able to keep the next successfully every time.

Although the algorithm was very simple while writing the New Sort, I got stuck at some point. Since it is written in the pdf, we find the min and max indexes simultaneously and perform the swaps later, but when we go this way, the min and max indexes remain the same after swaps, which can cause some trouble. That's why I found and swapped the first min, then I found Max and proceeded with swapping, and I can say that the problem was solved, but I can say that it took me a long time to find it.

Class Diagrams




newSort
myTestArray int[]
newSort()
findMaxIndex(int[], int, int) int
findMinIndex(int[], int, int) int
newSort(int[], int, int) int[]
QuickSort
QuickSort()
partition(T[], int, int) int
quickSort(T[], int, int) void
sort(T[]) void
swap(T[], int, int) void
MergeSort
MergeSort()
merge(T[], T[], T[]) void
sort(T[]) void
Main
Main()
main(String[]) void
Powered by yFiles

Performance of Coalesced

This strategy is effective, efficient, and very easy to implement. However, sometimes the extra memory use might be prohibitive, and the most common alternative, open addressing, has uncomfortable disadvantages that decrease performance. The primary disadvantage of open addressing is primary and secondary clustering, in which searches may access long sequences of used buckets that contain items with different hash addresses; items with one hash address can thus lengthen searches for items with other hash addresses.

Deletion may be hard.

Coalesced chaining avoids the effects of primary and secondary clustering, and as a result can take advantage of the efficient search algorithm for separate chaining. If the chains are short, this strategy is very efficient and can be highly condensed, memory-wise. As in open addressing, deletion from a coalesced hash table is awkward and potentially expensive, and resizing the table is terribly expensive and should be done rarely, if ever

Performance of Double hashing

The advantage of Double hashing is that it is one of the best form of probing, producing a uniform distribution of records throughout a hash table.
This technique does not yield any clusters.
It is one of effective method for resolving collisions.

The disadvantages of double hashing are as follows:

Double hashing is more difficult to implement than any other.

Double hashing can cause thrashing.

Test Cases

```
HashtableCoalesced<Integer,Integer> lmap1 = new HashtableCoalesced<>( tableSizeOfHash: 100);
System.out.println("Coalesced Version");
System.out.println("************************Map1************************");
//add some elements
start = System.currentTimeMillis();
System.out.println("** ADD 100 ELEMENTS TO COALESCED HASHTABLE **");
for(int i = 0;i< 100;i++){

    System.out.println("Current i (Key) val :"+i+" size before add :" + lmap1.size()+" add current i (Value) val:"+lmap1.put(i, i*i)+" size after add :" + lm
}
end = System.currentTimeMillis();
elapsedTime = end - start;
System.out.println("Total time for this execution : "+ elapsedTime);
```

```
Current i (Key) val :73 size before add :73 add current i (Value) val:73 size after add :74
Current i (Key) val :74 size before add :74 add current i (Value) val:74 size after add :75
Current i (Key) val :75 size before add :75 add current i (Value) val:75 size after add :76
Current i (Key) val :76 size before add :76 add current i (Value) val:76 size after add :77
Current i (Key) val :77 size before add :77 add current i (Value) val:77 size after add :78
Current i (Key) val :78 size before add :78 add current i (Value) val:78 size after add :79
Current i (Key) val :79 size before add :79 add current i (Value) val:79 size after add :80
Current i (Key) val :80 size before add :80 add current i (Value) val:80 size after add :81
Current i (Key) val :81 size before add :81 add current i (Value) val:81 size after add :82
Current i (Key) val :82 size before add :82 add current i (Value) val:82 size after add :83
Current i (Key) val :83 size before add :83 add current i (Value) val:83 size after add :84
Current i (Key) val :84 size before add :84 add current i (Value) val:84 size after add :85
Current i (Key) val :85 size before add :85 add current i (Value) val:85 size after add :86
Current i (Key) val :86 size before add :86 add current i (Value) val:86 size after add :87
Current i (Key) val :87 size before add :87 add current i (Value) val:87 size after add :88
Current i (Key) val :88 size before add :88 add current i (Value) val:88 size after add :89
Current i (Key) val :89 size before add :89 add current i (Value) val:89 size after add :90
Current i (Key) val :90 size before add :90 add current i (Value) val:90 size after add :91
Current i (Key) val :91 size before add :91 add current i (Value) val:91 size after add :92
Current i (Key) val :92 size before add :92 add current i (Value) val:92 size after add :93
Current i (Key) val :93 size before add :93 add current i (Value) val:93 size after add :94
Current i (Key) val :94 size before add :94 add current i (Value) val:94 size after add :95
Current i (Key) val :95 size before add :95 add current i (Value) val:95 size after add :96
Current i (Key) val :96 size before add :96 add current i (Value) val:96 size after add :97
Current i (Key) val :97 size before add :97 add current i (Value) val:97 size after add :98
Current i (Key) val :98 size before add :98 add current i (Value) val:98 size after add :99
Current i (Key) val :99 size before add :99 add current i (Value) val:99 size after add :100
Total time for this execution : 27.0
```

```
HashtableCoalesced<Integer,Integer> lmap2 = new HashtableCoalesced<>( tableSizeOfHash: 1000);
System.out.println("************************Map2************************");
//add some elements
start = System.currentTimeMillis();
System.out.println("** ADD 1000 ELEMENTS TO COALESCED HASHTABLE **");
for(int i = 0;i< 1000;i++){

    System.out.println("Current i (Key) val :"+i+" size before add :" + lmap2.size()+" add current i (Value) val:"+lmap2.put(i, i)+" size after add :" + lma
}
end = System.currentTimeMillis();
elapsedTime = end - start;
System.out.println("Total time for this execution : "+ elapsedTime);
```

```
Current i (Key) val :977 size before add :977 add current i (Value) val:977 size after add :978
Current i (Key) val :978 size before add :978 add current i (Value) val:978 size after add :979
Current i (Key) val :979 size before add :979 add current i (Value) val:979 size after add :980
Current i (Key) val :980 size before add :980 add current i (Value) val:980 size after add :981
Current i (Key) val :981 size before add :981 add current i (Value) val:981 size after add :982
Current i (Key) val :982 size before add :982 add current i (Value) val:982 size after add :983
Current i (Key) val :983 size before add :983 add current i (Value) val:983 size after add :984
Current i (Key) val :984 size before add :984 add current i (Value) val:984 size after add :985
Current i (Key) val :985 size before add :985 add current i (Value) val:985 size after add :986
Current i (Key) val :986 size before add :986 add current i (Value) val:986 size after add :987
Current i (Key) val :987 size before add :987 add current i (Value) val:987 size after add :988
Current i (Key) val :988 size before add :988 add current i (Value) val:988 size after add :989
Current i (Key) val :989 size before add :989 add current i (Value) val:989 size after add :990
Current i (Key) val :990 size before add :990 add current i (Value) val:990 size after add :991
Current i (Key) val :991 size before add :991 add current i (Value) val:991 size after add :992
Current i (Key) val :992 size before add :992 add current i (Value) val:992 size after add :993
Current i (Key) val :993 size before add :993 add current i (Value) val:993 size after add :994
Current i (Key) val :994 size before add :994 add current i (Value) val:994 size after add :995
Current i (Key) val :995 size before add :995 add current i (Value) val:995 size after add :996
Current i (Key) val :996 size before add :996 add current i (Value) val:996 size after add :997
Current i (Key) val :997 size before add :997 add current i (Value) val:997 size after add :998
Current i (Key) val :998 size before add :998 add current i (Value) val:998 size after add :999
Current i (Key) val :999 size before add :999 add current i (Value) val:999 size after add :1000
Total time for this execution : 96.0
```

```java
HashtableCoalesced<Integer,Integer> lmap3 = new HashtableCoalesced<>( tableSizeOfHash: 10000);
System.out.println("************************Map3************************");
//add some elements
start = System.currentTimeMillis();
System.out.println("** ADD 10000 ELEMENTS TO COALESCED HASHTABLE **");
for(int i = 0;i< 10000;i++){

    System.out.println("Current i (Key) val :"+i+" size before add :" + lmap3.size()+" add current i (Value) val:"+lmap3.put(i, i)+" size after add :" + lmap
}
end = System.currentTimeMillis();
elapsedTime = end - start;
System.out.println("Total time for this execution : "+ elapsedTime);
```

```
Current i (Key) val :9978 size before add :9978 add current i (Value) val:9978 size after add :9979
Current i (Key) val :9979 size before add :9979 add current i (Value) val:9979 size after add :9980
Current i (Key) val :9980 size before add :9980 add current i (Value) val:9980 size after add :9981
Current i (Key) val :9981 size before add :9981 add current i (Value) val:9981 size after add :9982
Current i (Key) val :9982 size before add :9982 add current i (Value) val:9982 size after add :9983
Current i (Key) val :9983 size before add :9983 add current i (Value) val:9983 size after add :9984
Current i (Key) val :9984 size before add :9984 add current i (Value) val:9984 size after add :9985
Current i (Key) val :9985 size before add :9985 add current i (Value) val:9985 size after add :9986
Current i (Key) val :9986 size before add :9986 add current i (Value) val:9986 size after add :9987
Current i (Key) val :9987 size before add :9987 add current i (Value) val:9987 size after add :9988
Current i (Key) val :9988 size before add :9988 add current i (Value) val:9988 size after add :9989
Current i (Key) val :9989 size before add :9989 add current i (Value) val:9989 size after add :9990
Current i (Key) val :9990 size before add :9990 add current i (Value) val:9990 size after add :9991
Current i (Key) val :9991 size before add :9991 add current i (Value) val:9991 size after add :9992
Current i (Key) val :9992 size before add :9992 add current i (Value) val:9992 size after add :9993
Current i (Key) val :9993 size before add :9993 add current i (Value) val:9993 size after add :9994
Current i (Key) val :9994 size before add :9994 add current i (Value) val:9994 size after add :9995
Current i (Key) val :9995 size before add :9995 add current i (Value) val:9995 size after add :9996
Current i (Key) val :9996 size before add :9996 add current i (Value) val:9996 size after add :9997
Current i (Key) val :9997 size before add :9997 add current i (Value) val:9997 size after add :9998
Current i (Key) val :9998 size before add :9998 add current i (Value) val:9998 size after add :9999
Current i (Key) val :9999 size before add :9999 add current i (Value) val:9999 size after add :10000
Total time for this execution : 41084.0

Process finished with exit code 0
```

```java
System.out.println("Chain Version");
HashtableChain<Integer, Integer> map1 = new HashtableChain<Integer, Integer>();
System.out.println("************************Map1*************************");
//add some elements
start = System.currentTimeMillis();
System.out.println("** ADD 100 ELEMENTS TO CHAIN HASHTABLE **");
for(int i = 0;i< 100;i++){

    System.out.println("Current i (Key) val :"+i+" size before add :" + map1.size()+" add current i (Value) val:"+map1.put(i, i)+" size after add :" + map
}
end = System.currentTimeMillis();
elapsedTime = end - start;
System.out.println("Total time for this execution : "+ elapsedTime);
```

```
Current i (Key) val :89 size before add :89 add current i (Value) val:89 size after add :90
Current i (Key) val :90 size before add :90 add current i (Value) val:90 size after add :91
Current i (Key) val :91 size before add :91 add current i (Value) val:91 size after add :92
Current i (Key) val :92 size before add :92 add current i (Value) val:92 size after add :93
Current i (Key) val :93 size before add :93 add current i (Value) val:93 size after add :94
Current i (Key) val :94 size before add :94 add current i (Value) val:94 size after add :95
Current i (Key) val :95 size before add :95 add current i (Value) val:95 size after add :96
Current i (Key) val :96 size before add :96 add current i (Value) val:96 size after add :97
Current i (Key) val :97 size before add :97 add current i (Value) val:97 size after add :98
Current i (Key) val :98 size before add :98 add current i (Value) val:98 size after add :99
Current i (Key) val :99 size before add :99 add current i (Value) val:99 size after add :100
Total time for this execution : 118.0
```

```java
HashtableChain<Integer, Integer> map2 = new HashtableChain<Integer, Integer>();
System.out.println("************************Map2*************************");
//add some elements
start = System.currentTimeMillis();
System.out.println("** ADD 1000 ELEMENTS TO CHAIN HASHTABLE **");
for(int i = 0;i< 1000;i++){

    System.out.println("Current i (Key) val :"+i+" size before add :" + map2.size()+" add current i (Value) val:"+map2.put(i, i)+" size after add :"
}
end = System.currentTimeMillis();
elapsedTime = end - start;
System.out.println("Total time for this execution : "+ elapsedTime);
```

```
Current i (Key) val :978 size before add :978 add current i (Value) val:978 size after add :979
Current i (Key) val :979 size before add :979 add current i (Value) val:979 size after add :980
Current i (Key) val :980 size before add :980 add current i (Value) val:980 size after add :981
Current i (Key) val :981 size before add :981 add current i (Value) val:981 size after add :982
Current i (Key) val :982 size before add :982 add current i (Value) val:982 size after add :983
Current i (Key) val :983 size before add :983 add current i (Value) val:983 size after add :984
Current i (Key) val :984 size before add :984 add current i (Value) val:984 size after add :985
Current i (Key) val :985 size before add :985 add current i (Value) val:985 size after add :986
Current i (Key) val :986 size before add :986 add current i (Value) val:986 size after add :987
Current i (Key) val :987 size before add :987 add current i (Value) val:987 size after add :988
Current i (Key) val :988 size before add :988 add current i (Value) val:988 size after add :989
Current i (Key) val :989 size before add :989 add current i (Value) val:989 size after add :990
Current i (Key) val :990 size before add :990 add current i (Value) val:990 size after add :991
Current i (Key) val :991 size before add :991 add current i (Value) val:991 size after add :992
Current i (Key) val :992 size before add :992 add current i (Value) val:992 size after add :993
Current i (Key) val :993 size before add :993 add current i (Value) val:993 size after add :994
Current i (Key) val :994 size before add :994 add current i (Value) val:994 size after add :995
Current i (Key) val :995 size before add :995 add current i (Value) val:995 size after add :996
Current i (Key) val :996 size before add :996 add current i (Value) val:996 size after add :997
Current i (Key) val :997 size before add :997 add current i (Value) val:997 size after add :998
Current i (Key) val :998 size before add :998 add current i (Value) val:998 size after add :999
Current i (Key) val :999 size before add :999 add current i (Value) val:999 size after add :1000
Total time for this execution : 1877.0

Process finished with exit code 0
```

```java
start = System.currentTimeMillis();
System.out.println("** ADD 10000 ELEMENTS TO CHAIN HASHTABLE **");
for(int i = 0;i< 10000;i++){

    System.out.println("Current i (Key) val :"+i+" size before add :" + map3.size()+" add current i (Value) val:"+map3.put(i, i)+" size after add :" + map3.s
}
end = System.currentTimeMillis();
elapsedTime = end - start;
System.out.println("Total time for this execution : "+ elapsedTime);
```

```
Current i (Key) val :9974 size before add :9974 add current i (Value) val:9974 size after add :9975
Current i (Key) val :9975 size before add :9975 add current i (Value) val:9975 size after add :9976
Current i (Key) val :9976 size before add :9976 add current i (Value) val:9976 size after add :9977
Current i (Key) val :9977 size before add :9977 add current i (Value) val:9977 size after add :9978
Current i (Key) val :9978 size before add :9978 add current i (Value) val:9978 size after add :9979
Current i (Key) val :9979 size before add :9979 add current i (Value) val:9979 size after add :9980
Current i (Key) val :9980 size before add :9980 add current i (Value) val:9980 size after add :9981
Current i (Key) val :9981 size before add :9981 add current i (Value) val:9981 size after add :9982
Current i (Key) val :9982 size before add :9982 add current i (Value) val:9982 size after add :9983
Current i (Key) val :9983 size before add :9983 add current i (Value) val:9983 size after add :9984
Current i (Key) val :9984 size before add :9984 add current i (Value) val:9984 size after add :9985
Current i (Key) val :9985 size before add :9985 add current i (Value) val:9985 size after add :9986
Current i (Key) val :9986 size before add :9986 add current i (Value) val:9986 size after add :9987
Current i (Key) val :9987 size before add :9987 add current i (Value) val:9987 size after add :9988
Current i (Key) val :9988 size before add :9988 add current i (Value) val:9988 size after add :9989
Current i (Key) val :9989 size before add :9989 add current i (Value) val:9989 size after add :9990
Current i (Key) val :9990 size before add :9990 add current i (Value) val:9990 size after add :9991
Current i (Key) val :9991 size before add :9991 add current i (Value) val:9991 size after add :9992
Current i (Key) val :9992 size before add :9992 add current i (Value) val:9992 size after add :9993
Current i (Key) val :9993 size before add :9993 add current i (Value) val:9993 size after add :9994
Current i (Key) val :9994 size before add :9994 add current i (Value) val:9994 size after add :9995
Current i (Key) val :9995 size before add :9995 add current i (Value) val:9995 size after add :9996
Current i (Key) val :9996 size before add :9996 add current i (Value) val:9996 size after add :9997
Current i (Key) val :9997 size before add :9997 add current i (Value) val:9997 size after add :9998
Current i (Key) val :9998 size before add :9998 add current i (Value) val:9998 size after add :9999
Current i (Key) val :9999 size before add :9999 add current i (Value) val:9999 size after add :10000
Total time for this execution : 1418.0
```

Part 2:

```
    for(int i=0;i<100;i++){
    mytestarr[i] = 100-i;
    }
        System.out.println("** Sorting 100 ELEMENTS With Quick Sort Algorithm **");
        for(int i=0;i<100;i++){
System.out.print(mytestarr[i] + " ");        }
        System.out.println();
        System.out.println();
        start = System.currentTimeMillis();
    mytest.sort(mytestarr);

        for(int i=0;i<100;i++){
            System.out.print(mytestarr[i] + " ");        }
        end = System.currentTimeMillis();
        elapsedTime = end - start;
        System.out.println();
        System.out.println("Total execution time of 100 element in Quick Sort is: " + elapsedTime + "ms.");
    }
```

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

1 2 3 4 5 6 7 8 9 10 11 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 93 94 95 96 97 98 99 100
Total execution time of 100 element in Quick Sort is: 2.0ms.
```

```
System.out.println("** Sorting 100 ELEMENTS With Merge Sort Algorithm **");
for(int i=0;i<100;i++){
    System.out.print(mytestarr[i] + " ");        }
System.out.println();
System.out.println();
start = System.currentTimeMillis();
mytestMerge.sort(mytestarr);

for(int i=0;i<100;i++){
    System.out.print(mytestarr[i] + " ");        }
end = System.currentTimeMillis();
elapsedTime = end - start;
System.out.println();
System.out.println("Total execution time of 100 element in Merge Sort is: " + elapsedTime + "ms.");
```

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

1 2 3 4 5 6 7 8 9 10 11 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 93 94 95 96 97 98 99 100
Total execution time of 100 element in Merge Sort is: 1.0ms.
```

```
    System.out.println("** Sorting 100 ELEMENTS With  New Sort Algorithm **");
    for(int i=0;i<100;i++){
        mytestarr2[i] = 100-i;
    }
    for(int i=0;i<100;i++){
        System.out.print(mytestarr2[i] + " ");        }
    System.out.println();
    System.out.println();
    start = System.currentTimeMillis();
    myNewSortTest.newSort(mytestarr2, head: 0, tail: 99);

    for(int i=0;i<100;i++){
        System.out.print(mytestarr2[i] + " ");        }
    end = System.currentTimeMillis();
    elapsedTime = end - start;
    System.out.println();
    System.out.println("Total execution time of 100 element in New Sort Sort is: " + elapsedTime + "ms.");
```

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

1 2 3 4 5 6 7 8 9 10 11 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 93 94 95 96 97 98 99 100
Total execution time of 100 element in New Sort Sort is: 1.0ms.
```

```
  for(int i=0;i<1000;i++){
  mytestarr[i] = 1000-i;
  }
      System.out.println("** Sorting 1000 ELEMENTS With Quick Sort Algorithm **");
      for(int i=0;i<1000;i++){
ystem.out.print(mytestarr[i] + " ");        }
      System.out.println();
      System.out.println();
      start = System.currentTimeMillis();
  mytest.sort(mytestarr);

      for(int i=0;i<1000;i++){
          System.out.print(mytestarr[i] + " ");        }
      end = System.currentTimeMillis();
      elapsedTime = end - start;
      System.out.println();
      System.out.println("Total execution time of 1000 element in Quick Sort is: " + elapsedTime + "ms.");
```

```
634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688
689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743
744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798
799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853
854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908
909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963
964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000
Total execution time of 1000 element in Quick Sort is: 12.0ms.
```

```
      for(int i=0;i<1000;i++){
          mytestarr[i] = 1000-i;
      }
      System.out.println("** Sorting 1000 ELEMENTS With Merge Sort Algorithm **");
      for(int i=0;i<1000;i++){
          mytestarr[i] = 1000-i;
      }

      for(int i=0;i<1000;i++){
          System.out.print(mytestarr[i] + " ");        }
      System.out.println();
      System.out.println();
      start = System.currentTimeMillis();
      mytestMerge.sort(mytestarr);

      for(int i=0;i<1000;i++){
          System.out.print(mytestarr[i] + " ");        }
      end = System.currentTimeMillis();
      elapsedTime = end - start;
      System.out.println();
      System.out.println("Total execution time of 1000 element in Merge Sort is: " + elapsedTime + "ms.");
```

```
469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523
524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578
579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633
634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688
689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743
744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798
799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853
854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908
909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963
964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000
Total execution time of 1000 element in Merge Sort is: 4.0ms.
```

```
        System.out.println("** Sorting 1000 ELEMENTS With  New Sort Algorithm **");
        for(int i=0;i<1000;i++){
            mytestarr2[i] = 1000-i;
        }
        for(int i=0;i<1000;i++){
            System.out.print(mytestarr2[i] + " ");        }
        System.out.println();
        System.out.println();
        start = System.currentTimeMillis();
        myNewSortTest.newSort(mytestarr2, head: 0, tail: 999);

        for(int i=0;i<1000;i++){
            System.out.print(mytestarr2[i] + " ");        }
        end = System.currentTimeMillis();
        elapsedTime = end - start;
        System.out.println();
        System.out.println("Total execution time of 1000 element in New Sort Sort is: " + elapsedTime + "ms.");
```

```
469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523
524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578
579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633
634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688
689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743
744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798
799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853
854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908
909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963
964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000
Total execution time of 1000 element in New Sort Sort is: 14.0ms.
```

```
    for(int i=0;i<10000;i++){
    mytestarr[i] = 10000-i;
    }
        System.out.println("** Sorting 10000 ELEMENTS With Quick Sort Algorithm **");
        for(int i=0;i<10000;i++){
System.out.print(mytestarr[i] + " ");        }
        System.out.println();
        System.out.println();
        start = System.currentTimeMillis();
    mytest.sort(mytestarr);

        for(int i=0;i<10000;i++){
            System.out.print(mytestarr[i] + " ");        }
        end = System.currentTimeMillis();
        elapsedTime = end - start;
        System.out.println();
        System.out.println("Total execution time of 10000 element in Quick Sort is: " + elapsedTime + "ms.");
```

```
9507 9508 9509 9510 9511 9512 9513 9514 9515 9516 9517 9518 9519 9520 9521 9522 9523 9524 9525 9526 9527 9528 9529 9530 9531 9532 9533 9534 9535 9536 9537 9538 9539 9540 9541 9542 9543 9544 9545 9546 9547 9548 9549 9550
9551 9552 9553 9554 9555 9556 9557 9558 9559 9560 9561 9562 9563 9564 9565 9566 9567 9568 9569 9570 9571 9572 9573 9574 9575 9576 9577 9578 9579 9580 9581 9582 9583 9584 9585 9586 9587 9588 9589 9590 9591 9592 9593 9594
9595 9596 9597 9598 9599 9600 9601 9602 9603 9604 9605 9606 9607 9608 9609 9610 9611 9612 9613 9614 9615 9616 9617 9618 9619 9620 9621 9622 9623 9624 9625 9626 9627 9628 9629 9630 9631 9632 9633 9634 9635 9636 9637 9638
9639 9640 9641 9642 9643 9644 9645 9646 9647 9648 9649 9650 9651 9652 9653 9654 9655 9656 9657 9658 9659 9660 9661 9662 9663 9664 9665 9666 9667 9668 9669 9670 9671 9672 9673 9674 9675 9676 9677 9678 9679 9680 9681 9682
9683 9684 9685 9686 9687 9688 9689 9690 9691 9692 9693 9694 9695 9696 9697 9698 9699 9700 9701 9702 9703 9704 9705 9706 9707 9708 9709 9710 9711 9712 9713 9714 9715 9716 9717 9718 9719 9720 9721 9722 9723 9724 9725 9726
9727 9728 9729 9730 9731 9732 9733 9734 9735 9736 9737 9738 9739 9740 9741 9742 9743 9744 9745 9746 9747 9748 9749 9750 9751 9752 9753 9754 9755 9756 9757 9758 9759 9760 9761 9762 9763 9764 9765 9766 9767 9768 9769 9770
9771 9772 9773 9774 9775 9776 9777 9778 9779 9780 9781 9782 9783 9784 9785 9786 9787 9788 9789 9790 9791 9792 9793 9794 9795 9796 9797 9798 9799 9800 9801 9802 9803 9804 9805 9806 9807 9808 9809 9810 9811 9812 9813 9814
9815 9816 9817 9818 9819 9820 9821 9822 9823 9824 9825 9826 9827 9828 9829 9830 9831 9832 9833 9834 9835 9836 9837 9838 9839 9840 9841 9842 9843 9844 9845 9846 9847 9848 9849 9850 9851 9852 9853 9854 9855 9856 9857 9858
9859 9860 9861 9862 9863 9864 9865 9866 9867 9868 9869 9870 9871 9872 9873 9874 9875 9876 9877 9878 9879 9880 9881 9882 9883 9884 9885 9886 9887 9888 9889 9890 9891 9892 9893 9894 9895 9896 9897 9898 9899 9900 9901 9902
9903 9904 9905 9906 9907 9908 9909 9910 9911 9912 9913 9914 9915 9916 9917 9918 9919 9920 9921 9922 9923 9924 9925 9926 9927 9928 9929 9930 9931 9932 9933 9934 9935 9936 9937 9938 9939 9940 9941 9942 9943 9944 9945 9946
9947 9948 9949 9950 9951 9952 9953 9954 9955 9956 9957 9958 9959 9960 9961 9962 9963 9964 9965 9966 9967 9968 9969 9970 9971 9972 9973 9974 9975 9976 9977 9978 9979 9980 9981 9982 9983 9984 9985 9986 9987 9988 9989 9990
9991 9992 9993 9994 9995 9996 9997 9998 9999 10000
Total execution time of 10000 element in Quick Sort is: 223.0ms.
```

```java
for(int i=0;i<10000;i++){
    mytestarr[i] = 10000-i;
}
System.out.println("** Sorting 10000 ELEMENTS With Merge Sort Algorithm **");
for(int i=0;i<10000;i++){
    mytestarr[i] = 10000-i;
}

for(int i=0;i<10000;i++){
    System.out.print(mytestarr[i] + " ");        }
System.out.println();
System.out.println();
start = System.currentTimeMillis();
mytestMerge.sort(mytestarr);

for(int i=0;i<10000;i++){
    System.out.print(mytestarr[i] + " ");        }
end = System.currentTimeMillis();
elapsedTime = end - start;
System.out.println();
System.out.println("Total execution time of 10000 element in Merge Sort is: " + elapsedTime + "ms.");
```

```
9551 9552 9553 9554 9555 9556 9557 9558 9559 9560 9561 9562 9563 9564 9565 9566 9567 9568 9569 9570 9571 9572 9573 9574 9575 9576 9577 9578 9579 9580 9581 9582 9583 9584 9585 9586 9587 9588 9589 9590 9591 9592 9593 9594
9595 9596 9597 9598 9599 9600 9601 9602 9603 9604 9605 9606 9607 9608 9609 9610 9611 9612 9613 9614 9615 9616 9617 9618 9619 9620 9621 9622 9623 9624 9625 9626 9627 9628 9629 9630 9631 9632 9633 9634 9635 9636 9637 9638
9639 9640 9641 9642 9643 9644 9645 9646 9647 9648 9649 9650 9651 9652 9653 9654 9655 9656 9657 9658 9659 9660 9661 9662 9663 9664 9665 9666 9667 9668 9669 9670 9671 9672 9673 9674 9675 9676 9677 9678 9679 9680 9681 9682
9683 9684 9685 9686 9687 9688 9689 9690 9691 9692 9693 9694 9695 9696 9697 9698 9699 9700 9701 9702 9703 9704 9705 9706 9707 9708 9709 9710 9711 9712 9713 9714 9715 9716 9717 9718 9719 9720 9721 9722 9723 9724 9725 9726
9727 9728 9729 9730 9731 9732 9733 9734 9735 9736 9737 9738 9739 9740 9741 9742 9743 9744 9745 9746 9747 9748 9749 9750 9751 9752 9753 9754 9755 9756 9757 9758 9759 9760 9761 9762 9763 9764 9765 9766 9767 9768 9769 9770
9771 9772 9773 9774 9775 9776 9777 9778 9779 9780 9781 9782 9783 9784 9785 9786 9787 9788 9789 9790 9791 9792 9793 9794 9795 9796 9797 9798 9799 9800 9801 9802 9803 9804 9805 9806 9807 9808 9809 9810 9811 9812 9813 9814
9815 9816 9817 9818 9819 9820 9821 9822 9823 9824 9825 9826 9827 9828 9829 9830 9831 9832 9833 9834 9835 9836 9837 9838 9839 9840 9841 9842 9843 9844 9845 9846 9847 9848 9849 9850 9851 9852 9853 9854 9855 9856 9857 9858
9859 9860 9861 9862 9863 9864 9865 9866 9867 9868 9869 9870 9871 9872 9873 9874 9875 9876 9877 9878 9879 9880 9881 9882 9883 9884 9885 9886 9887 9888 9889 9890 9891 9892 9893 9894 9895 9896 9897 9898 9899 9900 9901 9902
9903 9904 9905 9906 9907 9908 9909 9910 9911 9912 9913 9914 9915 9916 9917 9918 9919 9920 9921 9922 9923 9924 9925 9926 9927 9928 9929 9930 9931 9932 9933 9934 9935 9936 9937 9938 9939 9940 9941 9942 9943 9944 9945 9946
9947 9948 9949 9950 9951 9952 9953 9954 9955 9956 9957 9958 9959 9960 9961 9962 9963 9964 9965 9966 9967 9968 9969 9970 9971 9972 9973 9974 9975 9976 9977 9978 9979 9980 9981 9982 9983 9984 9985 9986 9987 9988 9989 9990
9991 9992 9993 9994 9995 9996 9997 9998 9999 10000
Total execution time of 10000 element in Merge Sort is: 187.0ms.
```

```java
System.out.println("** Sorting 10000 ELEMENTS With  New Sort Algorithm **");
for(int i=0;i<10000;i++){
    mytestarr2[i] = 10000-i;
}
for(int i=0;i<10000;i++){
    System.out.print(mytestarr2[i] + " ");        }
System.out.println();
System.out.println();
start = System.currentTimeMillis();
myNewSortTest.newSort(mytestarr2, head: 0, tail: 9999);

for(int i=0;i<10000;i++){
    System.out.print(mytestarr2[i] + " ");        }
end = System.currentTimeMillis();
elapsedTime = end - start;
System.out.println();
System.out.println("Total execution time of 10000 element in New Sort Sort is: " + elapsedTime + "ms.");
```

```
9463 9464 9465 9466 9467 9468 9469 9470 9471 9472 9473 9474 9475 9476 9477 9478 9479 9480 9481 9482 9483 9484 9485 9486 9487 9488 9489 9490 9491 9492 9493 9494 9495 9496 9497 9498 9499 9500 9501 9502 9503 9504 9505 9506
9507 9508 9509 9510 9511 9512 9513 9514 9515 9516 9517 9518 9519 9520 9521 9522 9523 9524 9525 9526 9527 9528 9529 9530 9531 9532 9533 9534 9535 9536 9537 9538 9539 9540 9541 9542 9543 9544 9545 9546 9547 9548 9549 9550
9551 9552 9553 9554 9555 9556 9557 9558 9559 9560 9561 9562 9563 9564 9565 9566 9567 9568 9569 9570 9571 9572 9573 9574 9575 9576 9577 9578 9579 9580 9581 9582 9583 9584 9585 9586 9587 9588 9589 9590 9591 9592 9593 9594
9595 9596 9597 9598 9599 9600 9601 9602 9603 9604 9605 9606 9607 9608 9609 9610 9611 9612 9613 9614 9615 9616 9617 9618 9619 9620 9621 9622 9623 9624 9625 9626 9627 9628 9629 9630 9631 9632 9633 9634 9635 9636 9637 9638
9639 9640 9641 9642 9643 9644 9645 9646 9647 9648 9649 9650 9651 9652 9653 9654 9655 9656 9657 9658 9659 9660 9661 9662 9663 9664 9665 9666 9667 9668 9669 9670 9671 9672 9673 9674 9675 9676 9677 9678 9679 9680 9681 9682
9683 9684 9685 9686 9687 9688 9689 9690 9691 9692 9693 9694 9695 9696 9697 9698 9699 9700 9701 9702 9703 9704 9705 9706 9707 9708 9709 9710 9711 9712 9713 9714 9715 9716 9717 9718 9719 9720 9721 9722 9723 9724 9725 9726
9727 9728 9729 9730 9731 9732 9733 9734 9735 9736 9737 9738 9739 9740 9741 9742 9743 9744 9745 9746 9747 9748 9749 9750 9751 9752 9753 9754 9755 9756 9757 9758 9759 9760 9761 9762 9763 9764 9765 9766 9767 9768 9769 9770
9771 9772 9773 9774 9775 9776 9777 9778 9779 9780 9781 9782 9783 9784 9785 9786 9787 9788 9789 9790 9791 9792 9793 9794 9795 9796 9797 9798 9799 9800 9801 9802 9803 9804 9805 9806 9807 9808 9809 9810 9811 9812 9813 9814
9815 9816 9817 9818 9819 9820 9821 9822 9823 9824 9825 9826 9827 9828 9829 9830 9831 9832 9833 9834 9835 9836 9837 9838 9839 9840 9841 9842 9843 9844 9845 9846 9847 9848 9849 9850 9851 9852 9853 9854 9855 9856 9857 9858
9859 9860 9861 9862 9863 9864 9865 9866 9867 9868 9869 9870 9871 9872 9873 9874 9875 9876 9877 9878 9879 9880 9881 9882 9883 9884 9885 9886 9887 9888 9889 9890 9891 9892 9893 9894 9895 9896 9897 9898 9899 9900 9901 9902
9903 9904 9905 9906 9907 9908 9909 9910 9911 9912 9913 9914 9915 9916 9917 9918 9919 9920 9921 9922 9923 9924 9925 9926 9927 9928 9929 9930 9931 9932 9933 9934 9935 9936 9937 9938 9939 9940 9941 9942 9943 9944 9945 9946
9947 9948 9949 9950 9951 9952 9953 9954 9955 9956 9957 9958 9959 9960 9961 9962 9963 9964 9965 9966 9967 9968 9969 9970 9971 9972 9973 9974 9975 9976 9977 9978 9979 9980 9981 9982 9983 9984 9985 9986 9987 9988 9989 9990
9991 9992 9993 9994 9995 9996 9997 9998 9999 10000
Total execution time of 10000 element in New Sort Sort is: 254.0ms.
```

In every Statistics the performance is the sort of this methods are

Merge Sort > Quick Sort > New Sort