## Q2

**n random unsorted numbers:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithm | n=5 | n=10 | n=50 | n=100 | n=500 | n=1000 | n=10000 |
| Selection Sort | 8.18494 | 20.81892 | 11.43213 | 13.0558 | 49.81161 | 116.92278 | 1703.62903 |
| Insertion Sort | 0.09548 | 0.12515 | 0.4453 | 2.28715 | 34.08061 | 61.20072 | 1792.21928 |
| Merge Sort | 0.20406 | 0.34279 | 2.16947 | 1.61143 | 7.97337 | 12.49275 | 128.74144 |
| Quick Sort | 0.13865 | 0.23557 | 0.34798 | 1.9622 | 6.77097 | 10.21593 | 43.52293 |

手机屏幕截图

描述已自动生成

**n random sorted numbers in ascending(increasing) order:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithm | n=5 | n=10 | n=50 | n=100 | n=500 | n=1000 | n=10000 |
| Selection Sort | 0.07431 | 0.16909 | 3.09277 | 3.29725 | 53.87743 | 97.79655 | 1753.07329 |
| Insertion Sort | 0.06849 | 0.0801 | 0.08382 | 0.10571 | 0.33963 | 0.70986 | 6.83453 |
| Merge Sort | 0.1961 | 0.2559 | 0.76144 | 1.39256 | 7.53997 | 12.47577 | 62.27649 |
| Quick Sort | 0.07725 | 0.15246 | 0.16608 | 0.36361 | 0.77268 | 1.45553 | 71.86234 |

手机屏幕截图

描述已自动生成

**n random sorted numbers in descending(decreasing) order:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithm | n=5 | n=10 | n=50 | n=100 | n=500 | n=1000 | n=10000 |
| Selection Sort | 0.07972 | 0.172 | 1.98737 | 3.10357 | 46.71075 | 118.40056 | 1753.45876 |
| Insertion Sort | 0.08319 | 0.20924 | 0.9055 | 2.60239 | 73.32035 | 107.80084 | 2849.55394 |
| Merge Sort | 0.18205 | 0.34931 | 0.76303 | 1.3621 | 12.21739 | 16.78406 | 51.91658 |
| Quick Sort | 0.1102 | 0.17365 | 0.2529 | 0.39084 | 2.47204 | 4.47844 | 30.74935 |

手机屏幕截图

描述已自动生成

**Discussion:**

Selection Sort: O(n^2)

Insertion Sort: O(n^2)

Merge Sort: O(nlogn)

Quick Sort: O(n^2) expected O(nlogn)

Comparing the sortedness of the input array, when the input array is sorted, merge sort will spend less time than input an unsorted array. Especially, if the sorted array and insertion sort algorithm have the same direction for example, both are ascending, then this algorithm will greatly reduce the running time, because it does not need to swap. For other two algorithms, the sortedness hardly affected on them.

As for the influence of the array size “n” suppose the input array is unsorted, when n is small, selection sort and insertion sort may faster than merge and quick sort. However, when n is big enough, merge sort and quick sort are much faster than selection and insertion sort.

I expected quick sort is faster than merge sort when the array is sorted, because they at that time both big-O is O(nlog(n)) and merge sort spends more space memory. But the test shows that quick sort is slower than merge when array is in ascending order (I set both algorithm in sorting ascending) but faster in descending.

**Code:**

public static void main (String[] args) {

int size = 10; //SIZE input(5, 10, 50, 100, 500, 1000, 10000)

ArrayList<Integer> list = new ArrayList<>(size+1);

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

list.add(i);

}

Integer[] a = new Integer[size];

for (int count = 0; count < size; count++){

a[count] = list.remove((int)(Math.random() \* list.size()));

}

// output a sorted list, true is des and false is asc

//SortingAlgorithms.quickSort(a, true);

Integer[] selection = new Integer[size];

Integer[] insertion = new Integer[size];

Integer[] merge = new Integer[size];

Integer[] quick = new Integer[size];

System.arraycopy(a, 0, selection, 0, a.length);

System.arraycopy(a, 0, insertion, 0, a.length);

System.arraycopy(a, 0, merge, 0, a.length);

System.arraycopy(a, 0, quick, 0, a.length);

double startTime=System.nanoTime();

SortingAlgorithms.selectionSort(selection, false);

double endTime=System.nanoTime();

System.out.println("selection： "+(endTime. startTime)/100000+"ms");

startTime=System.nanoTime();

SortingAlgorithms.insertionSort(insertion, false);

endTime=System.nanoTime();

System.out.println("insertion： "+(endTime-startTime)/100000+"ms");

startTime=System.nanoTime();

SortingAlgorithms.mergeSort(merge, false);

endTime=System.nanoTime();

System.out.println("merge： "+(endTime-startTime)/100000+"ms");

startTime=System.nanoTime();

SortingAlgorithms.quickSort(quick, false);

endTime=System.nanoTime();

System.out.println("quick： "+(endTime-startTime)/100000+"ms");