InsertionSort

n	tid
80 000	845 ms
160 000	3 463 ms
240 000	7 511 ms
320 000	13 394 ms
400 000	21 467 ms

$O(n^2)$ eftersom tiden ökar med ungefär 4 gånger när n dubblass.

DrakeSort

n	tid
40 000 000	273 ms
80 000 000	475 ms
120 000 000	527 ms
160 000 000	784 ms
200 000 000	892 ms

$O(n)$ eftersom tiden ökar konstant när n dubblas. Tiden dubblas.

QuickSort

n	tid
40 000 000	3 035 ms
80 000 000	6 321 ms
120 000 000	9 099 ms
160 000 000	12 950 ms
200 000 000	15 457 ms

$O(n)$ eftersom tiden ökar konstant när n dubblas. Tiden dubblas.

JavaAPISort

n	tid
8 000 000	2 033 ms
16 000 000	3 600 ms
24 000 000	8 122 ms
32 000 000	10 088 ms
40 000 000	17 423 ms

$O(n \log n)$ skulle jag säga. När man söker upp JavaAPISort tidskomplexitet online säger den att det borde vara O(n log(n)).

Min JavaAPISort:

```
public List<SpeciesId> sort(List<SpeciesId> s) {
s.sort(Comparator.comparing(SpeciesId::getOId).reversed().thenComparing(Species
Id::getVId));
  return s;
}
```

O(n log(n)).

Min insersionSort:

```
   int n = unsorted.length;
  for (int i = 1; i < n; ++i) {
     int key = unsorted[i];
     int j = i - 1;
     while (j >= 0 && unsorted[j] > key) {
        unsorted[j + 1] = unsorted[j];
        j = j - 1;
     }
     unsorted[j + 1] = key;
  }
  return unsorted;
}
```

I min implementation borde tidskomplexiteten bli $O(n^2)$. Antalet gånger for loopen körs bestäms av n och i loopen finns en while loop. Eftersom det är en loop i en loop och den utre loopen bestäms av n skulle jag säga att tidskomplexiteten är $O(n^2)$.

Min drakeSort:

```
public int[] sort(int[] unsorted) {
  if (null == unsorted || unsorted.length < 2) {
     return unsorted; // nothing to sort
  }
  int max = unsorted[0];
  int min = unsorted[0];

  for (int i = 1; i < unsorted.length; i++) {
     if (min > unsorted[i]) {
        min = unsorted[i];
     } else if (max < unsorted[i]) {
        max = unsorted[i];
     }
  }
  int[] freq = new int[max - min + 1];
  for (int j : unsorted) {
     freq[j - min] += 1;
  }
  int[] sorted = new int[unsorted.length];
  for (int i = 0, k = 0; i < freq.length; i++) {
     while (freq[i] > 0) {
        sorted[k++] = min + i; // populate sorted array
        freq[i]--;
     }
  }
  return sorted;
}
```

Jag tror att tidskomplexiteten blir $O(2n)$. Jag fick den här tidskomplexiteten eftersom den första forloopen bestäms av freq.length vilket är mitt n. Min whileloop i forloopen bestäms inte av n på samma sätt så jag är inte säker om tidskomplexiteten är $O(2n)$ eller om den faktiskt är $O(n^2)$.