	Best case	**Worst case**	**Average case**
Insertion sort	O(n)	O(n^2)	O(n^2)
Selection sort	O(n^2)	O(n^2)	O(n^2)
Heap sort	O(nlogn)	O(nlogn)	O(nlogn)
Mergesort	O(nlogn)	O(nlogn)	O(nlogn)
Quicksort	O(nlogn)	O(n^2)	O(nlogn)
Bucket sort	O(n+k)	O(n^2)	O(n+k)
Radix sort	O(nk)	O(nk)	O(nk)

Radix sort:
Input: 478, 537, 9, 721, 3, 38, 123, 67

One's place

0	1	2	3	4	5	6	7	8	9
	721		3 123				537 67	478 38	9

Intermediate order: 721, 3, 123, 537, 67, 478, 38, 9

Ten's place

0	1	2	3	4	5	6	7	8	9
3 9		721 123	537 38			67	478		

Intermediate order: 3, 9, 721, 123, 537, 38, 67, 478

Hundred's place

0	1	2	3	4	5	6	7	8	9
3 9 38 67	123			478	537		721		

Final: 3, 9, 38, 67, 123, 478, 537, 721

For each of the following situations, name the best sorting algorithm we studied. (For one or two questions, there may be more than one answer deserving full credit, but you only need to give one answer for each.)

(a) The array is mostly sorted already (a few elements are in the wrong place). **Insertion sort**

(b) You need an O(n log n) sort even in the worst case and you cannot use any extra space except for a few local variables. **Heap sort**

(c) The data to be sorted is too big to fit in memory, so most of it is on disk. **Merge Sort**

(d) You have many data sets to sort separately, and each one has only around 10 elements. **Insertion/selection sort**

(e) You have a large data set, but all the data has only one of about 10 values for sorting purposes (e.g., the data is records of elementary-school students and the sort is by age in years). **Bucket sort**

(f) Instead of sorting the entire data set, you only need the k smallest elements where k is an input to the algorithm but is likely to be much smaller than the size of the entire data set. S**election sort is the simplest most natural choice and fine if k is truly small – this is the expected answer. If k is not a small constant, we might prefer heap sort or a variant of quicksort with a cut-off like we used on a homework problem.**

O(N2) average, worst case:
– **Selection Sort, Bubblesort, Insertion Sort**

O(N log N) average case:
– **Heapsort**: In-place, not stable.
– **BST Sort**: O(N) extra space (including tree pointers, possibly poor memory locality), stable.
– **Mergesort**: O(N) extra space, stable.
– **Quicksort**: claimed fastest in practice, but O(N2) worst case. Recursion/stack requirement. Not stable.

O(N log N) worst and average case:
– **Any comparison-based sorting algorithm**

O(N)
– **Radix Sort**: fast and stable. Not comparison based. Not inplace. Poor memory locality can undercut performance.
– **Bucket Sort:** O(n+k): fast, stable, not comparison based. Not inplace, poor memory locality