Selection sort, seleciona o primeiro elemento como mínimo, compara TODOS os elementos a direita, pega o menor se houve e troca para a primeira posição se necessário, depois pega o segundo elemento já que a esquerda o elemento está na posição ordenada, compara um por um a direita até pegar o novo mínimo se houver, e trocar para a segunda posição, sendo agora a primeira e segunda posição, ordenada.

Comparações(sempre igual, cálculável), trocas depende(pior caso calculável = numero de comparações).   
Comparações: n(n-1)/2 ou n²/2 + n/2 mesmo ordenado ele vai sempre a mesma quantidade de comparações O(n²)

EX:  
23 |42 04 16 08 15  
04 |42 23 16 08 15  
04 08| 23 16 42 15  
04 08 15| 16 42 23  
04 08 15 16 23 42|

Insertion Sort, dividi em porção ordenada e não ordenada, primeiro elemento é considerado o começo e o final da parte ordenada, depois se olha o segundo número, caso seja maior, ele entra na parte ordenada,

No pior caso n(n-1) /2 = O (n²), no melhor caso N comparações.

23 |42 04 16 08 15  
23 42| 04 16 08 15  
23 04 42 16 08 15  
04 23 42 |16 08 15  
04 23 16 42 08 15  
04 16 23 42 |08 15  
04 16 23 08 42 15  
04 16 08 23 42 15  
04 08 16 23 42 |15  
04 08 16 23 15 42  
04 08 16 15 23 42  
04 08 15 16 23 42|

Bubble sort, para cada elemento da lista, deve se comparar ao diretamente a sua direita se eles são fora de ordem um em relação ao outro, há a troca, faz isso pra cada elemento da lista, e conclui uma etapa, deve passar pela quantidade de etapas necessárias para ficar arrumada, no pior caso (elementos ao contrario do que se quer), n-1 etapas.  
otimizando: caso não haja trocas, está arrumada a lista;  
otimizando: a cada interação, é certo que o maior elemento estará na ultima posição, portanto é inútil olhar para aquele número. Para cada interação do conjunto inteiro, deve-se diminuir em n em 1.  
primeira iteração n-0, para se passar por todo o arranjo, depois n-1 e assim por diante

Tempo de execução nxn (n²), não recomendado para muitos elementos porque aumenta exponencialmente

3 2 9 6 5  
2 3 9 6 5 – 3 e 9 é comparado, já está arrumado, portanto compara 4 e 5º elem  
2 3 6 9 5   
2 3 6 5 9 – primeira rodada  
2 3 6 5 9 – 2 é menor que 3, 3 é menor que 6.  
2 3 5 6 9

Merge Sort separa em uma n listas sendo n o numero de elementos, compara de 2 em 2, 4 em 4 e assim por diante até arrumar, merge sort precisa de um espaço adicional para ser mesclada, velocidade: O(n log n), muito melhor para listas menores de 50.

quick sort, primeiro pega o ultimo elemento e o chama de pivô, cria a parede a esquerda do primeiro índice onde todos os elementos que entrarão ali serão menor que o pivô, ao chegar no pivô se percebe que o lugar dele é a direita do elemento a esquerda da parede não necessariamente sendo seu lugar final, depois repete o processo a esquerda e a direita da parede, caso já esteja ordenado a esquerda, move-se o índice da parede +1 e o ultimo elemento é o novo pivô, por exemplo ao arranjar a direita da parede, para cada caso em que o elemento que se está olhando seja menor que o pivô, pega-se este elemento e o troca com o primeiro elemento a direita da parede até que todos os elementos sejam pivôs ou subarranjos de matriz 1.

O(n log n)