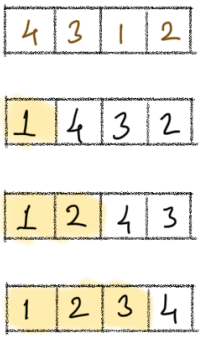
sorting

selection sort

select the smallest element from the array, put it at the beginning
↳ then start from second element
↳ repeat this n-1 times



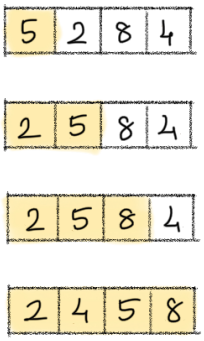
best	aver	worst
$O(n^2)$	$O(n^2)$	$O(n^2)$

```
void selection_sort ( int ar[], int n){
  for(int i=0; i<n-1;i++) {
    int min = i;
    for(int j=i+1; j<n ;j++){
      if (ar[min] > ar[j])
          min=j;
    }
    int temp = ar[min];
    ar[min] = ar[i];
    ar[i] = temp;
  }
}
```

insertion sort

in each pass, the first elem of the unsorted part is inserted to sorted part

- appropriate for small inputs

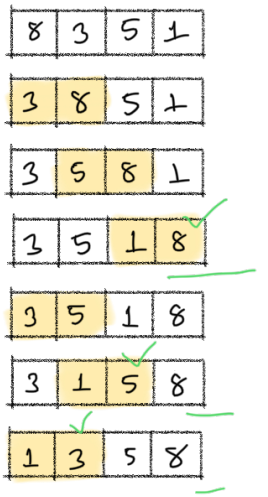


best	aver	worst
$O(n)$	$O(n^2)$	$O(n^2)$

```
void insertion_sort ( int ar[], int n){
  for(int i=1; i<n ;i++) {
    int insert = ar[i];
    int j=i;                    j = 4
    for(; j>0 && (arr[j-1]>insert); j--)
      arr[j] = arr[j-1];
    arr[j] = insert;
  }
}
```

bubble sort

repeatedly swap adjacent elements

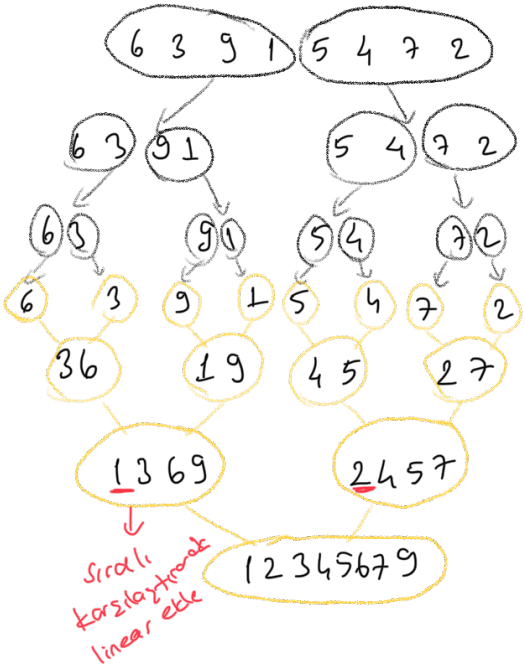



```
void bubble_sort ( int ar[], int n){
  bool sorted = false;
  int last= n-1;
  for(int i=0; (i<last) && !sorted ; i++){
    sorted = true;
    for( int j=0; j<last; j++){
      if (ar[j] > ar[j+1]){
        swap (ar[j], ar[j+1])
        sorted = false;        } } }
```

best	aver	worst
$o(n)$	$o(n^2)$	$o(n^2)$

merge sort

↳ divide the list into halves, sort each part recursively, then merge
↳ divide and conquer




```
merge_sort ( A, p, r){
if (p < r)
    then  q ← [(p+r)/2]      } O(logn)
          merge sort (A,p,q)
          merge sort (A,q+1,r)
          merge(A,p,q,r)      → O(n)
```

best	aver	worst
$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

- not memory efficient

* only merge sort is not inplace, other algorithms arranges given array, but merge sorting uses extra strage.

stable sort = when the order of elements is maintained when values are same. (only heap sort is not stable)

quick sort

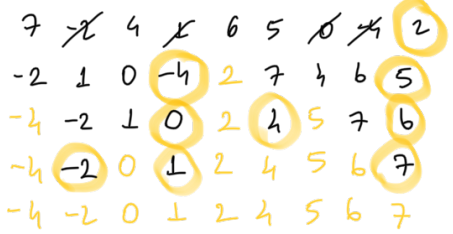
select a pivot element, then divide the elements with respect to it as smalles and greates, sort the parts recursively, then bring all together

↳ divide and conquer

↳ no use of extra memory

→ most popular sorting algorithm

pivot = select a random elem

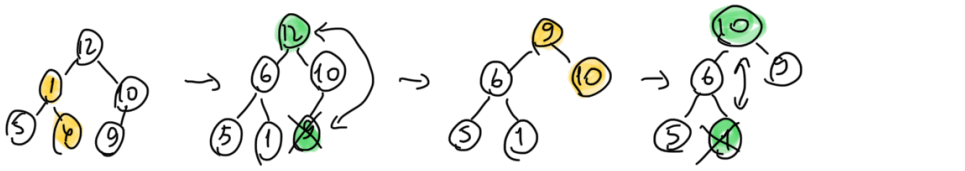


best	aver	worst
$O(n \log n)$	$O(n \log n)$	$O(n^2)$

heap sort

1 12 10 5 6 9 →

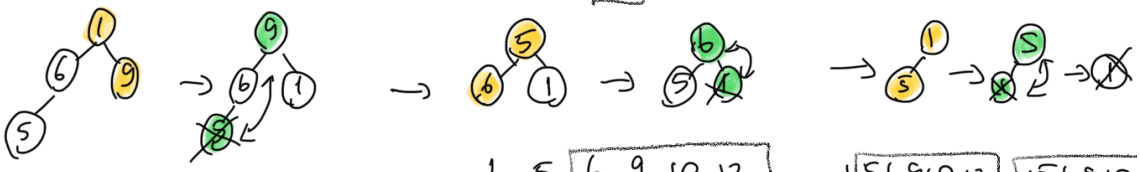
max heap:



12 6 10 5 1 9

9 6 10 5 1 12

1 6 9 5 10 12



1 5 6 9 10 12

1 5 6 9 10 12

5 6 1 9 10 12

best	aver	worst
$O(n \log n)$	$O(n \log n)$	$O(n \log n)$