Selection sort

```
for i = 0 to n-1
(a) choose the minimum index from a[i], a[i+1], ..., a[n-1]
(b) swap a[i] and a[min_index]
```

#swap: O(n)
#comparison: O(n^2)
time: O(n^2)
space: O(1)---in-place sorting

```
          1
      1       2
   5    1    4    2
 5 7 1 3 4 6 8 2
```

tournament sort

```
build a min-winner tree (O(n) in time)
for i = 0 to n-1
(a) choose the minimum index from a[i], a[i+1], ..., a[n-1]
      with a min-winner tree
(b) swap a[i] and a[min_index]
(c) update i-th leaf of the tree and min_index-th leaf of the
     tree (O(log n) in time)
```

space: O(n)
time: O(n log n)

bubble sort

```
for(i=0;i<len;i++){
  int changed = 0;
  for(j=0;j<len-i-1;j++){
    printf("%d %d: ", i, j);
    if (arr[j] > arr[j+1]){
      swap(arr+j, arr+j+1);
      changed = 1;
    }
    show(arr, len);
  }
  if (!changed)
    break;
}
```

#swap: O(n^2)
#comparison: O(n^2)
space: O(1)---in-place
can early stop if a sorted

insertion sort
for i = 0 to n-1
(a) consider a[i]
(b) find the postion in a[0], a[1], ..., a[i-1]
(c) insert a[i] into the position

space: O(1)
time: O(n^2)
almost sorted: almost O(n)

usually,
insertion better than bubble;
selection better than bubble;
insertion faster than selection in practice

winner tree => merge tree
(1, 2, 3, 4, 5, 6, 7, 8) : O(n) time
(1, 3, 5, 7) (2, 4, 6, 8): O(n) time
(5,7) (1,3) (4,6) (2, 8): O(n) time
 5 7   1 3   4 6   8 2

merge sort
(1) build a merge tree
(2) output the root

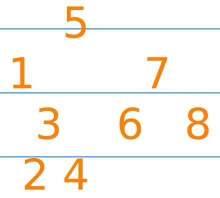
time: O(n log n)
space: O(n log n), can be down to O(n)

heap sort
convert a to a max heap
for i = 0 to n-1
(a) swap a[0] with a[n-1-i]
(b) maintain heap property for new a[0] (O(log (n-i)) time)

time: O(n log n)
space: O(1)

BST sort
(a) build a BST from a: time O(n^2) worst case, space O(n)
(b) in-order traversal on the BST: time O(n), space O(h)

quick sort: BST sort without building a BST
5, 7, 1, 3, 4, 6, 8, 2



5, 7, 1, 3, 4, 6, 8, 2
1, 3, 4, 2, 5, 7, 6, 8

1, 3, 4, 2, 5, 6, 7, 8

1, 2, 3, 4, 5, 6, 7, 8