sorting
selection sort

for i = 0 to len-1
(a) find the minimum index in a[i], a[i+1], ..., a[len-1]
(b) swap a[min_index] with a[i]

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

tournament sort
build a min-winner tree with a
for i = 0 to len-1
(a) find the minimum index in a[i], a[i+1], ..., a[len-1]
    with the min-winner tree
(b) swap a[min_index] with a[i]
(c) update leaf min_index and leaf i in the winner tree

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

bubble sort

```
 for(i=0;i<len;i++){
   changed = 0;
   for(j=1;j<len-i;j++){
     if (arr[j] < arr[j-1]){
       changed = 1;
       swap(arr+j, arr+j-1);
     }
   }
   if (changed == 0)
     break;
 }
```

time: O(n^2) in comparison, O(n^2) in swap
space: O(1)---in-place
can early stop (changed == 0) when a ordered

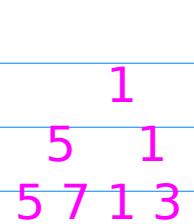
insertion sort

for i = 0 to len-1
(a) pick card a[i]
(b) insert a[i] into a[0], a[1], ..., a[i-1]

time: O(n^2)
space: O(1)
can be fast when a almost ordered

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

winner tree



5 7 1 3 4 6 8 2

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

(a) build a merge tree from a

(b) output the root

space: O(n log n),

can go down to O(n)

[hard to go down to O(1)]

time: O(n log n)

heapsort

modify a to a max-heap

for i = 0 to len-1

(a) max_index = 0 because of max-heap

(b) swap a[max_index] with a[len-i-1]

(c) heapify a[max_index] (maintain)

space: O(1)

time: O(n log n) for modify, O(n log n) for maintain