



Goal:

Some sorting algorithms

Time complexity analysis

substitution

master theorem

counting steps.

Set Data Structure	Operations $O(\cdot)$				
	Container	Static	Dynamic	Order	
	build(X)	find(k)	insert(x) delete(k)	find_min() find_max()	find_prev(k) find_next(k)
Array	$n$	$n$	$n$	$n$	$n$
Sorted Array	$n \log n$	$\log n$	$n$	1	$\log n$

- But how to construct a sorted array efficiently?

comparision based sorting  
algorithms can not do  
better than  $\Omega(n \log n)$

- Example:  $[2, 3, 1] \rightarrow \{[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]\}$

```

1 def permutation_sort(A):
2     '''Sort A'''
3     for B in permutations(A):
4         if is_sorted(B):
5             return B

```

# O(n!)  
# O(n)  
# O(1)

#### INSERTION-SORT(A)

```

1 for j = 2 to A.length
2     key = A[j]
3     // Insert A[j] into the sorted
      sequence A[1..j-1].
4     i = j - 1
5     while i > 0 and A[i] > key
6         A[i+1] = A[i]
7         i = i - 1
8     A[i+1] = key

```

cost	times
$c_1$	$n$
$c_2$	$n-1$
$c_3$	$n-1$
$c_4$	$n-1$
$c_5$	$\sum_{j=2}^n t_j$
$c_6$	$\sum_{j=2}^n (t_j - 1)$
$c_7$	$\sum_{j=2}^n (t_j - 1)$
$c_8$	$n-1$

#### Selection Sort

- Find a largest number in prefix  $A[:i+1]$  and swap it to  $A[i]$
- Recursively sort prefix  $A[:i]$
- Example:  $[8, 2, 4, 9, 3], [8, 2, 4, 3, 9], [3, 2, 4, 8, 9], [3, 2, 4, 8, 9], [2, 3, 4, 8, 9]$

```

1 def selection_sort(A, i = None):
2     '''Sort A[:i+1]'''
3     if i is None: i = len(A) - 1
4     if i > 0:
5         j = prefix_max(A, i)
6         A[i], A[j] = A[j], A[i]
7         selection_sort(A, i - 1)
8
9 def prefix_max(A, i):
10     '''Return index of maximum in A[:i+1]'''
11     if i > 0:
12         j = prefix_max(A, i - 1)
13         if A[i] < A[j]:
14             return j
15     return i

```

# T(i)  
# O(1)  
# O(1)  
# S(i)  
# O(1)  
# T(i-1)  
# S(i)  
# O(1)  
# S(i-1)  
# O(1)  
# O(1)  
# O(1)

- Example:  $[7, 1, 5, 6, 2, 4, 9, 3], [1, 7, 5, 6, 2, 4, 3, 9], [1, 5, 6, 7, 2, 3, 4, 9], [1, 2, 3, 4, 5, 6, 7, 9]$

```

1 def merge_sort(A, a = 0, b = None):
2     '''Sort A[a:b]'''
3     if b is None: b = len(A)
4     if 1 < b - a:
5         c = (a + b + 1) // 2
6         merge_sort(A, a, c)
7         merge_sort(A, c, b)
8         L, R = A[a:c], A[c:b]
9         merge(L, R, A, len(L), len(R), a, b)
10
11 def merge(L, R, A, i, j, a, b):
12     '''Merge sorted L[:i] and R[:j] into A[a:b]'''
13     if a < b:
14         if (j <= 0) or (i > 0 and L[i-1] > R[j-1]):
15             A[b-1] = L[i-1]
16             i = i - 1
17         else:
18             A[b-1] = R[j-1]
19             j = j - 1
20     merge(L, R, A, i, j, a, b-1)

```

# T(b-a=n)  
# O(1)  
# O(1)  
# O(1)  
# T(n/2)  
# T(n/2)  
# O(n)  
# S(n)  
# S(b-a=n)  
# O(1)  
# O(1)  
# O(1)  
# O(1)  
# O(1)  
# O(1)  
# S(n-1)

**Theorem 4.1 (Master theorem)**

Let  $a \geq 1$  and  $b > 1$  be constants, let  $f(n)$  be a function, and let  $T(n)$  be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n) ,$$

where we interpret  $n/b$  to mean either  $\lfloor n/b \rfloor$  or  $\lceil n/b \rceil$ . Then  $T(n)$  has the following asymptotic bounds:

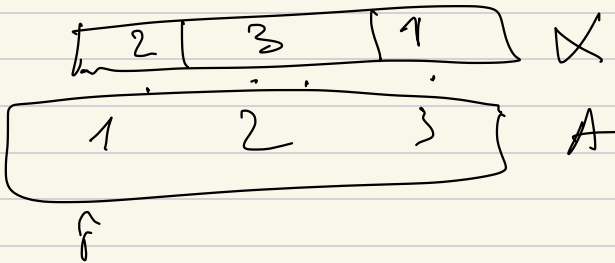
1. If  $f(n) = O(n^{\log_b a - \epsilon})$  for some constant  $\epsilon > 0$ , then  $T(n) = \Theta(n^{\log_b a})$ .
2. If  $f(n) = \Theta(n^{\log_b a})$ , then  $T(n) = \Theta(n^{\log_b a} \lg n)$ .
3. If  $f(n) = \Omega(n^{\log_b a + \epsilon})$  for some constant  $\epsilon > 0$ , and if  $af(n/b) \leq cf(n)$  for some constant  $c < 1$  and all sufficiently large  $n$ , then  $T(n) = \Theta(f(n))$ . ■

- Example: [2, 3, 1]  $\rightarrow$  {[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]}

```

1 def permutation_sort(A):
2     '''Sort A'''
3     for B in permutations(A): ✓                # O(n!)
4         if is_sorted(B): ✓                      # O(n)
5             return B ✓                          # O(1)

```



$$\left. \begin{array}{l} A[1] > A[0] \\ A[2] > A[1] \end{array} \right\} O(n)$$

$$\left. \begin{array}{l} \text{for } 0 < i < n \\ A[i] > A[i-1] \end{array} \right\} O(n)$$

$$O(n!) \quad \underline{\underline{\text{not good}}}$$

# INSERTION-SORT(A)

	cost	times
1 for $j = 2$ to $A.length$	$c_1$	$n$
2 $key = A[j]$	$c_2$	$n - 1$
3     // Insert $A[j]$ into the sorted sequence $A[1 \dots j - 1]$ .	0	$n - 1$
4 $i = j - 1$	$c_4$	$n - 1$
5     while $i > 0$ and $A[i] > key$	$c_5$	$\sum_{j=2}^n t_j$
6 $A[i + 1] = A[i]$	$c_6$	$\sum_{j=2}^n (t_j - 1)$
7 $i = i - 1$	$c_7$	$\sum_{j=2}^n (t_j - 1)$
8 $A[i + 1] = key$	$c_8$	$n - 1$

we hope to have an algorithm that  $O(n \log n)$   
 Analysis by counting the steps:

$$A_1 = \begin{matrix} 0 & 1 & 2 & 3 & 4 \\ [1, & 2, & 3, & 4, & 5] \end{matrix}$$

key 2 at  $j = 1$

$i = 0$

Compare  $A[i] > key$

$A[0] > A[1]$

1 2 3

$t_j =$

$$A_2 = [5, 4, 3, 2, 1]$$

$t_j^0$  is

$$t_j^0 = 1 + 2 + 3 + \dots + n$$

$$t_j^0 = \frac{n(n+1)}{2} - 1$$

$O(n^2)$

$T_j \geq \underline{\underline{O(n)}}$  in the best  
case

↑  
worst  
case.

in the worst case insertion  
sort take  $O(n^2)$

Analysis by counting the steps

## Selection Sort

- Find a largest number in prefix  $A[:i + 1]$  and swap it to  $A[i]$
- Recursively sort prefix  $A[:i]$
- Example:  $[8, 2, 4, 9, 3]$ ,  $[8, 2, 4, 3, 9]$ ,  $[3, 2, 4, 8, 9]$ ,  $[3, 2, 4, 8, 9]$ ,  $[2, 3, 4, 8, 9]$

```

1 def selection_sort(A, i = None):           # T(i)
2     '''Sort A[:i + 1]'''
3     if i is None: i = len(A) - 1         # O(1)
4     if i > 0:                             # O(1)
5         j = prefix_max(A, i)             # S(i)
6         A[i], A[j] = A[j], A[i]          # O(1)
7         selection_sort(A, i - 1)         # T(i - 1)
8
9 def prefix_max(A, i):                     # S(i)
10    '''Return index of maximum in A[:i + 1]'''
11    if i > 0:                             # O(1)
12        j = prefix_max(A, i - 1)         # S(i - 1)
13        if A[i] < A[j]:                   # O(1)
14            return j                      # O(1)
15    return i                             # O(1)

```

*Handwritten notes:*  $O(n^2)$  for selection\_sort,  $O(n)$  for prefix\_max. A bracket groups the recursive call and the swap in selection\_sort.

$[8 \ 2 \ 4 \ 9 \ 3]$

$[8 \ 2 \ 4 \ 3 \ 9]$

↑  
what consider

$[3 \ 2 \ 4 \ 8 \ 9]$

↑ sorted.

prefix\_max (n)

$O(1)$

prefix\_max (n-1)

(n-2)

q



prefix max  $T_{pm}(n) = T_{pm}(n-1) + O(1)$

Substitution method

guess  $T_{pm}(n) = O(n)$

assume  $T_{pm}(n) = O(n)$

$$T(n-1) + O(1) \leq Cn$$

$$C(n-1) + O(1) \leq Cn$$

$$T_{pm}(n) \leq Cn$$

definition

$$Cn - C + O(1) \leq Cn$$

$$C \geq O(1)$$

$$C \geq K \quad ) \text{ ok}$$

if we had a guess that  $T(n) = O(\lg n)$   
we should arrive to a contradiction

---

Selection Sort prefix max  
↓

$$T(n) = T(n-1) + O(n)$$

guess  $O(n^2)$  show that it holds. !.!

- Example: [7, 1, 5, 6, 2, 4, 9, 3], [1, 7, 5, 6, 2, 4, 3, 9], [1, 5, 6, 7, 2, 3, 4, 9], [1, 2, 3, 4, 5, 6, 7, 9]

```

1 def merge_sort(A, a = 0, b = None):                                # T(b - a = n)
2     '''Sort A[a:b]'''
3     if b is None: b = len(A)                                     # O(1)
4     if 1 < b - a:                                                # O(1)
5         c = (a + b + 1) // 2                                     # O(1)
6         merge_sort(A, a, c)                                     # T(n / 2)
7         merge_sort(A, c, b)                                     # T(n / 2)
8         L, R = A[a:c], A[c:b]                                    # O(n)
9         merge(L, R, A, len(L), len(R), a, b)                   # S(n)
10
11 def merge(L, R, A, i, j, a, b):                                  # S(b - a = n)
12     '''Merge sorted L[:i] and R[:j] into A[a:b]'''
13     if a < b:                                                    # O(1)
14         if (j <= 0) or (i > 0 and L[i - 1] > R[j - 1]):        # O(1)
15             A[b - 1] = L[i - 1]                                 # O(1)
16             i = i - 1                                           # O(1)
17         else:                                                    # O(1)
18             A[b - 1] = R[j - 1]                                 # O(1)
19             j = j - 1                                           # O(1)
20         merge(L, R, A, i, j, a, b - 1)                         # S(n - 1)

```

[7 1 5 6 2 4 9 3]

L R

~~~~~

[7] [1] [5] [6] [2] [4] [9] [3]

↑

[1 7] [5 6] [2 4] [9 3]

[1] [5] [6] [7]

↑ ↑

i i

[2] [3] [4] [9]

↑ ↑

0 0

←

$$A[j] > A[i] \rightarrow$$

Merge  $O(n)$

$$\forall j \quad j = j - 1$$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

Guess that  $T(n) = O(\log n)$   
and show it by substitution.

**Theorem 4.1 (Master theorem)**

Let  $a \geq 1$  and  $b > 1$  be constants, let  $f(n)$  be a function, and let  $T(n)$  be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + \underline{f(n)},$$

where we interpret  $n/b$  to mean either  $\lfloor n/b \rfloor$  or  $\lceil n/b \rceil$ . Then  $T(n)$  has the following asymptotic bounds:

1. If  $f(n) = O(n^{\log_b a - \epsilon})$  for some constant  $\epsilon > 0$ , then  $T(n) = \Theta(n^{\log_b a})$ .
2. If  $f(n) = \Theta(n^{\log_b a})$ , then  $T(n) = \Theta(n^{\log_b a} \lg n)$ .
3. If  $f(n) = \Omega(n^{\log_b a + \epsilon})$  for some constant  $\epsilon > 0$ , and if  $af(n/b) \leq cf(n)$  for some constant  $c < 1$  and all sufficiently large  $n$ , then  $T(n) = \Theta(f(n))$ . ■

$$T(n) = aT\left(\frac{n}{b}\right) + \underline{f(n)}$$

Merge sort  $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

$$a = 2$$

$$= 2T\left(\frac{n}{2}\right) + kn$$

$$b = 2$$

$$f(n) = O(n) = kn$$

$$\text{if } km = \Theta(n^4) = \Theta(m)$$

$$C_2 m \leq km \leq C_1 m$$

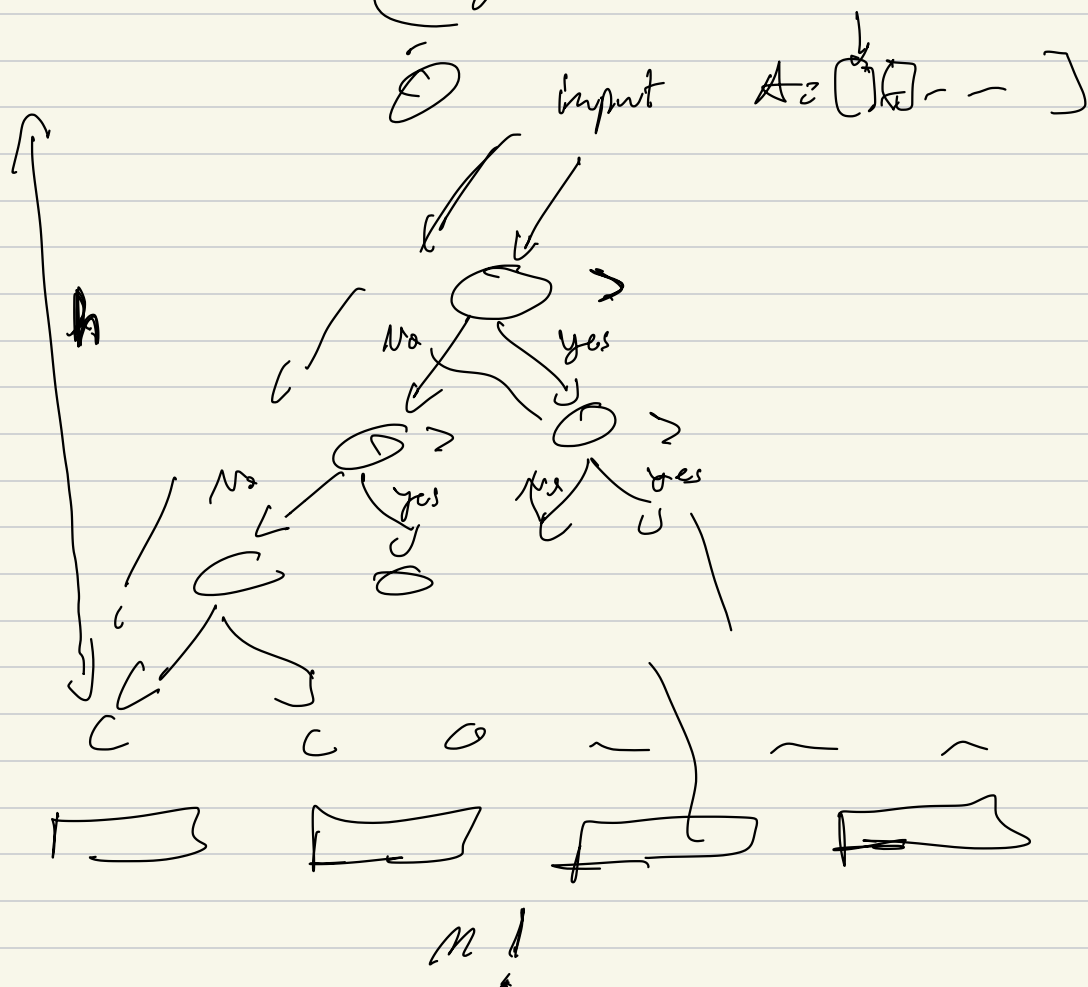
$$C_2 = 3 \quad k = 4 \quad C_1 = 5$$

$$\underline{\underline{OK}}$$

Menge take  $\Theta(\text{ulgn})$

Any comparison based sorting algorithm

$$\Omega(n \lg n)$$



$$2^h \geq n! \quad h \geq \lg(n!) \approx n \lg n$$