Goal:

Some sorting algorithms

- permutation
- insertion
- selection
- merge.

Time complexity analysis

- Substitution
- master theorem
- Counting steps.

Set Data Structure	Operations $O(\cdot)$ Container `build(X)`	Static `find(k)`	Dynamic `insert(x)` `delete(k)`	Order `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?

Comparison based sorting algorithms can not do better than $\Omega(n \lg n)$

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

```python
def permutation_sort(A):
    '''Sort A'''
    for B in permutations(A):                  # O(n!)
        if is_sorted(B):                       # O(n)
            return B                           # O(1)
```

INSERTION-SORT(A)	*cost*	*times*
for $j = 2$ **to** $A.length$	c_1	n
$key = A[j]$	c_2	$n-1$
// Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
$i = j - 1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
$A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] = key$	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:** $[8, 2, 4, 9, 3], [8, 2, 4, 3, 9], [3, 2, 4, 8, 9], [3, 2, 4, 8, 9], [2, 3, 4, 8, 9]$

```python
def selection_sort(A, i = None):             # T(i)
    '''Sort A[:i + 1]'''
    if i is None: i = len(A) - 1             # O(1)
    if i > 0:                                # O(1)
        j = prefix_max(A, i)                 # S(i)
        A[i], A[j] = A[j], A[i]              # O(1)
        selection_sort(A, i - 1)             # T(i - 1)

def prefix_max(A, i):                        # S(i)
    '''Return index of maximum in A[:i + 1]'''
    if i > 0:                                # O(1)
        j = prefix_max(A, i - 1)             # S(i - 1)
        if A[i] < A[j]:                      # O(1)
            return j                         # O(1)
    return i                                 # O(1)
```

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

```python
def merge_sort(A, a = 0, b = None):          # T(b - a = n)
    '''Sort A[a:b]'''
    if b is None: b = len(A)                 # O(1)
    if 1 < b - a:                            # O(1)
        c = (a + b + 1) // 2                 # O(1)
        merge_sort(A, a, c)                  # T(n / 2)
        merge_sort(A, c, b)                  # T(n / 2)
        L, R = A[a:c], A[c:b]                # O(n)
        merge(L, R, A, len(L), len(R), a, b) # S(n)

def merge(L, R, A, i, j, a, b):              # S(b - a = n)
    '''Merge sorted L[:i] and R[:j] into A[a:b]'''
    if a < b:                                # O(1)
        if (j <= 0) or (i > 0 and L[i - 1] > R[j - 1]): # O(1)
            A[b - 1] = L[i - 1]              # O(1)
            i = i - 1                        # O(1)
        else:                                # O(1)
            A[b - 1] = R[j - 1]              # O(1)
            j = j - 1                        # O(1)
        merge(L, R, A, i, j, a, b - 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

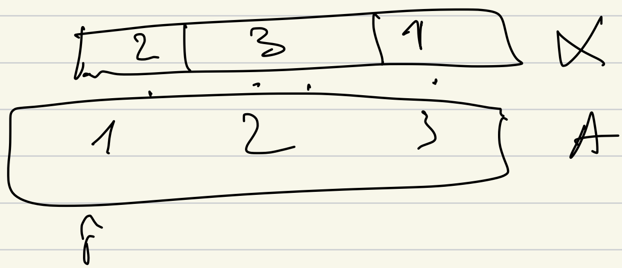
$$T(n) = aT(n/b) + f(n) ,$$

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 $T(n) = \Theta(f(n))$. $\blacksquare$

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

```python
def permutation_sort(A):
    '''Sort A'''
    for B in permutations(A):            # O(n!)
        if is_sorted(B):                 # O(n)
            return B                     # O(1)
```



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

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

$O(n!)$ **not good**

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

we hope to have an algorithm that $O(n \lg n)$

Analysis by counting the steps:

$$\begin{array}{c} \;\;\;0\;\;1\;\;2\;\;3\;\;4 \\ A_1 = [1, 2, 3, 4, 5] \end{array}$$

key 2 at $j = 1$

$i = 0$

Compaire $A[i] > key$

$A[0] > A[1]$

1 2 3

$t_j =$

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

t_j is

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

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

$O(n^2)$

$t_j = O(n)$ 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: $[8, 2, 4, 9, 3]$, $[8, 2, 4, 3, 9]$, $[3, 2, 4, 8, 9]$, $[3, 2, 4, 8, 9]$, $[2, 3, 4, 8, 9]$

```
def selection_sort(A, i = None):              # T(i)
    '''Sort A[:i + 1]'''
    if i is None: i = len(A) - 1              # O(1)
    if i > 0:                                 # O(1)
        j = prefix_max(A, i)                  # S(i)
        A[i], A[j] = A[j], A[i]               # O(1)
        selection_sort(A, i - 1)              # T(i - 1)

def prefix_max(A, i):                         # S(i)
    '''Return index of maximum in A[:i + 1]'''
    if i > 0:                                 # O(1)
        j = prefix_max(A, i - 1)              # S(i - 1)
        if A[i] < A[j]:                       # O(1)
            return j                          # O(1)
    return i                                  # O(1)
```

$O(n)$

$O(n^2)$

$O(n)$

A, i

[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)

a

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

Substitution method

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

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

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

$\underbrace{T_{pm}(n) \leq cn}_{\text{definition}}$

$c(n-1) + O(1) \leq cn$

$\cancel{cn} - c + O(1) \leq \cancel{cn}$

$c \geq O(1)$

$c \geq K$) ok

if we had a guess that $T(n) = O(\lg n)$

pm

we should arrive to a contradiction

prefix max

Selection Sort

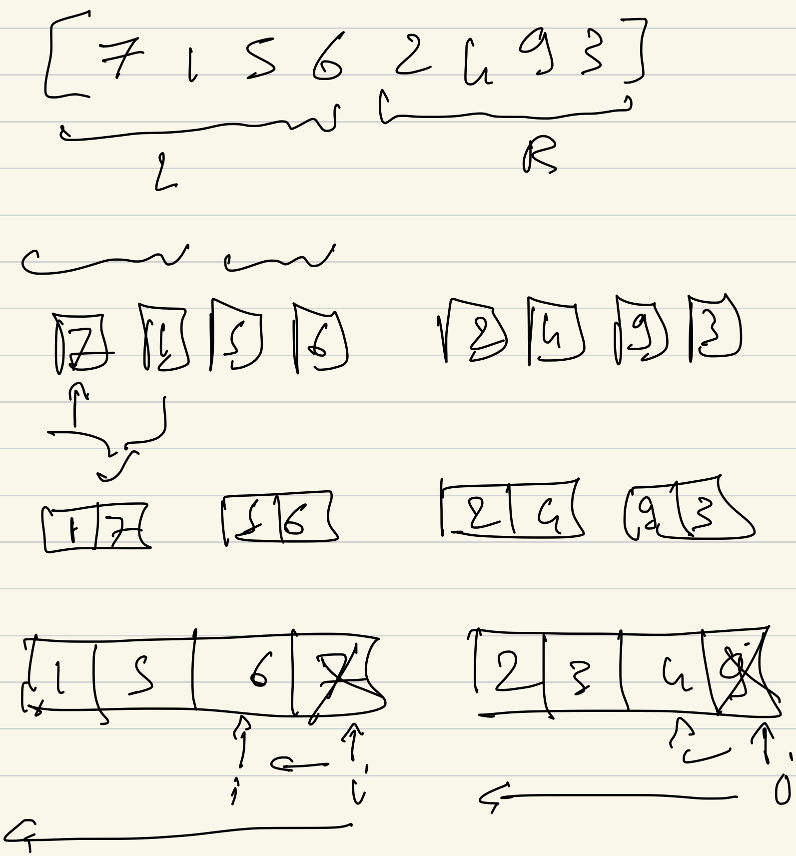
$$T(n) = T(n-1) + O(n)$$

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

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

```python
def merge_sort(A, a = 0, b = None):                  # T(b - a = n)
    '''Sort A[a:b]'''
    if b is None: b = len(A)                          # O(1)
    if 1 < b - a:                                     # O(1)
        c = (a + b + 1) // 2                          # O(1)
        merge_sort(A, a, c)                           # T(n / 2)
        merge_sort(A, c, b)                           # T(n / 2)
        L, R = A[a:c], A[c:b]                         # O(n)
        merge(L, R, A, len(L), len(R), a, b)          # S(n)

def merge(L, R, A, i, j, a, b):                       # S(b - a = n)
    '''Merge sorted L[:i] and R[:j] into A[a:b]'''
    if a < b:                                         # O(1)
        if (j <= 0) or (i > 0 and L[i - 1] > R[j - 1]): # O(1)
            A[b - 1] = L[i - 1]                       # O(1)
            i = i - 1                                 # O(1)
        else:                                         # O(1)
            A[b - 1] = R[j - 1]                       # O(1)
            j = j - 1                                 # O(1)
        merge(L, R, A, i, j, a, b - 1)                # S(n - 1)
```



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

79 $j = j - 1$

Merging $O(n)$

$$\underset{ms}{T}(n) = \underline{2}\, T\left(\frac{n}{\underline{2}}\right) + O(n)$$

Guess that $T(n) = O(n \lg n)$
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

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

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 $T(n) = \Theta(f(n))$. ■

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

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

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

$a = 2$

$b = 2$

$f(n) = O(n) = Kn$

if $kn = \Theta(n^4) = \Theta(n)$

$$c_2 n \leq kn \leq c_1 n$$

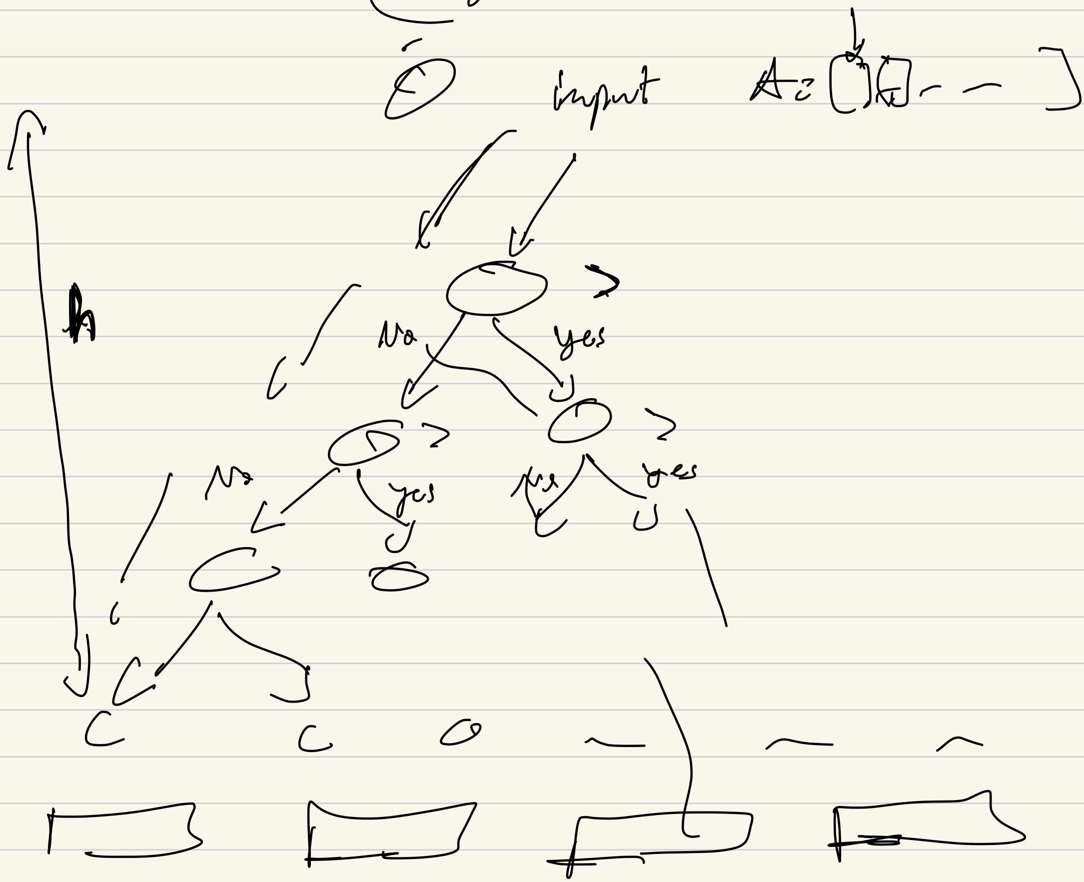
$c_2 = 3 \quad k = 4 \quad c_1 = 5$

ok

Merge take $\Theta(n \lg n)$

Any comparison based sorting algorithm

$\Omega(n \lg n)$



$n!$

$2^h \geq n! \qquad h \geq \lg(n!) = n \lg n$