LEC3 Sets and Sorting

https://github.com/GUMI-21/MIT6.006_note
3.17

Review of last LEC

- Interface
  Collection of operations(e.g.,sequence & set)
- Data structure
  Way to store data that supports a set of operations
  efficiency, memory usage

Set Interface

Container
build(A) give an iterable A, build sequence from items in A
len() return the number of stored items
Static
find(k) return the stored item with key k
Dynamic
insert(x) add x to set(replace item with key x. key if one already exists)
delete(k) remove nad return the stored item with key k
Order
iter_ord() return the stored items on-by-on in key order
find_min() return smallest key item
find_max() return largest key item
find_next(k) k+1 index item
find_prev(k) k-i index item

Data struct

- *a big array*
  all *interface* take linear time, $O(n)$
- *Sorted array*
  find item by using binary search.

Today we focus on Sort

Data Structure	Container `build(A)`	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	$\log n$

Sorting

- Destructive: Overwrites the input array
- In place: Use $O(1)$ extra space, it means the memory space doesn't grow by length of sort
  Input: Array of n numbers/keys A
  output: Sorted array B

Algorithms of sort

Permutation Sort

```
def permutation_sort(A):
    '''sort A'''
    for B in permutations(A):
        if is_sorted(B)
            return B
```

1. enumerate all permutaions: omiga(n!) -> means time >= n!, *lower boundary of run time*
2. Check if permutation is sorted, maybe `for i = i to n-1: B[i] == B[i+1]` O(n) so final $omiga(n! * n)$ time, it's very worse.

Selection Sort

- Example
  8 2 4 9 3 -> 8 2 4 3 | 9 -> 2 4 3 | 8 9 -> 2 3 | 4 8 9 -> 2 | 3 4 8 9 done.
  just keeping choose the biggest item.
  In 6.006, wo concern with proving correctness, proving efficiency, so we always write algorithm by recursive(Induction).
- *Algorithm*

1. Found biggest with index <= i
2. Swap

3. Sort 1,....,i-1 recursivly

- *Let's show step1, it's called prefix_max:*

```
def prefix_max(A,i):
    '''Return index of maximum in A[:i+1]'''
    if i > 0:
        j = prefix_max(A, i - 1) # find the biggest iten in 0 to i-1 is J
        if A[i] < A[j]:          # if A[i] < A[j], it's case2
            return j
    return i  # if A[i] > A[j] it's case 1

'''There are only two case of biggest element: 1.it is at index i, 2.it is at index < i; '''
```

- *Proof of alogrithm Step 1*:
  Base case: 1 element
  Induction step: case1 & case2 in i+1
- *run time of prefix_max*
  $S(1) = \theta(1), S(n) = S(n-1) + \theta(1)$ -> so runtime is $O(n)$ see 6.042J alg analysis

```
cn = c(n-1) + theta(1)
c = theta(1) □
```

- *Step 2 selection_sort & Step3 recursive*

```
def selection_sort(A, i = None):
    '''Sort A[:i+1]'''
    if i is None: i = len(A) - 1
    if i > 0:
        j = prefix_max(A, i)  # find biggest item index <= i
        A[i], A[j] = A[j], A[i] # swap i j
        selection_sort(A, i-1) # recursive function
```

runtime: $T(n) = T(n-1) + \theta(n) = \theta(n^2)(1+2+...+n = \theta(n^2))$ use *Plug&Chug*
So the selection sort takes O(n^2) time

Merge Sort

- Example
  7 1 5 6 2 4 9 3 -> |7 1|, |5 6|, |2 4|, |9 3| -> |1 7|, |5 6|, |2 4|, |3 9| ->

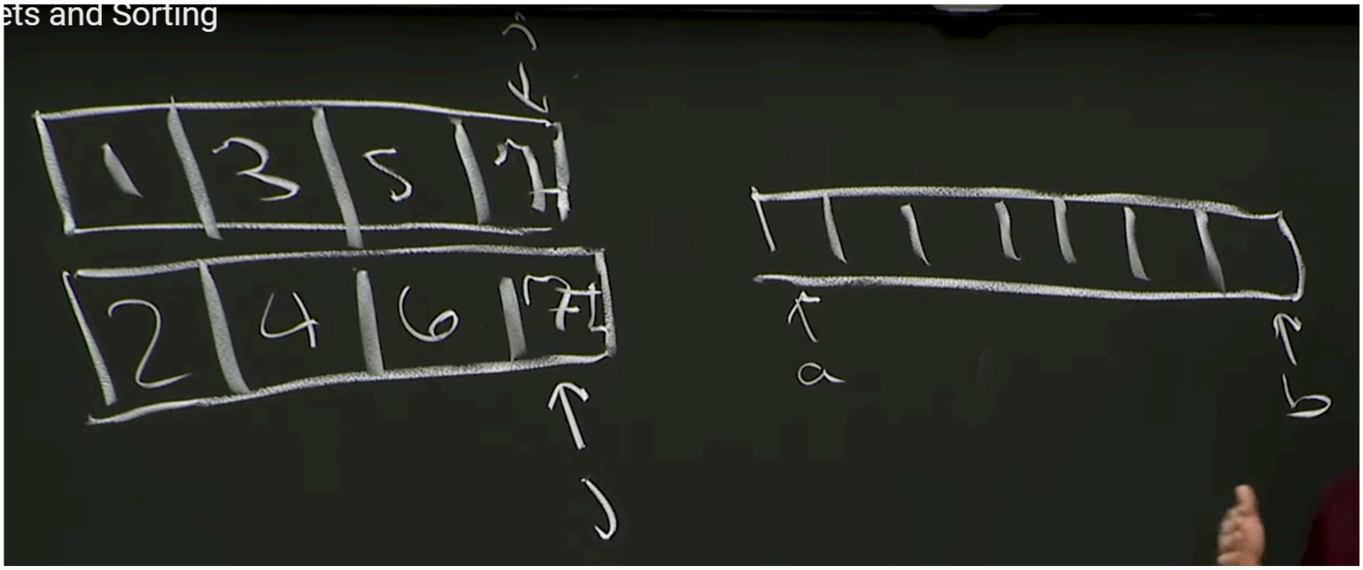
|1 7 . 5 6|, |2 4 . 3 9| *on .'s two sides, numbers has sorted* -> | 15 6 7 . 2 3 4 9 |

->merge method: *two figners alogrithm.*

```
1 5 6 7*
2 3 4 9*   -> 9
1 5 6 7*
2 3 4*     -> 7 9
1 5 6*
2 3 4*     -> 6 7 9 ....
```

```python
# destructive
def merge_sort(A, a=0, b=None):
    '''sort A[a:b]'''
    if b is None:
        b = len(A)
    if 1 < b - a:
        c = (a + b + 1) // 2   # middle of array
        merge_sort(A, a, c)
        merge_sort(A, c, b)
        L, R = A[a:c], A[c:b]
        merge(L,R,A, len(L) - 1, len(R) - 1, a, b)  # a & b is index of sorted

def merge(L, R, A, i, j, a, b):  # input biggest element of i & j into b
    '''recursive call: either make i or j into the last index of array'''
    if a < b:
        if (j <= 0) or (i > 0 and L[i] > R[j]):  # j array end or i > j
            A[b-1] = L[i]
            i = i - 1
        else:
            A[b-1] = R[j]
            j = j - 1
        merge(L, R, A, i, j, a, b - 1)
```



if element j >= element i, set j in b and call j - 1 & b - 1, or if i > j, set i in b and call i - 1 & b - 1, until a = b.

- *run time analysis*
  can see in note of mit6.042j - algorithm analysis
  hypothsis the length of array is always 2^n
  $T(1) = \theta(1), T(n) = 2T(n/2) + \theta(n), T(n) = \theta(nlogn)$
  vertify:
  $cnlogn = 2c(n/2)\log(n/2) + \theta(n) = cn(logn - log2) + \theta(n)$, then
  $\theta(n) = cnlog2 = cn$ *the runtime of comparsion of two fingers is cn*