Lecture 8 - January 30

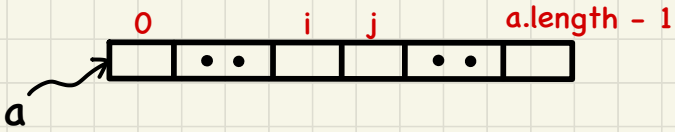
Arrays and Linked Lists

Exercise: Relating Sorting Orders
Selection vs. Insertion Sorts
Singly-Linked List: Quick, Visual Intro.

Announcements/Reminders

- Assignment 1 solution released
- ***splitArrayHarder***: an **extended** version released
- Lecture notes template available
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu
- Contact Information of TAs on common eClass site

Sorting Orders of Arrays

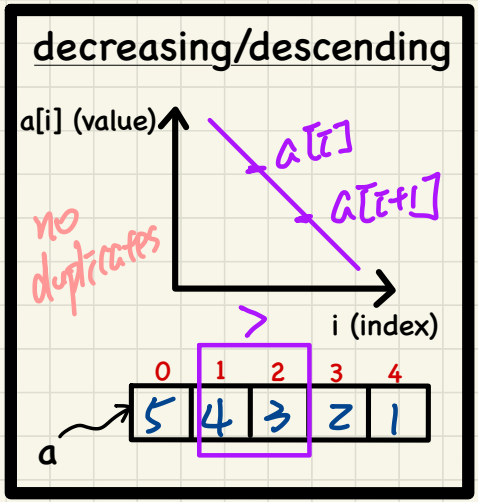


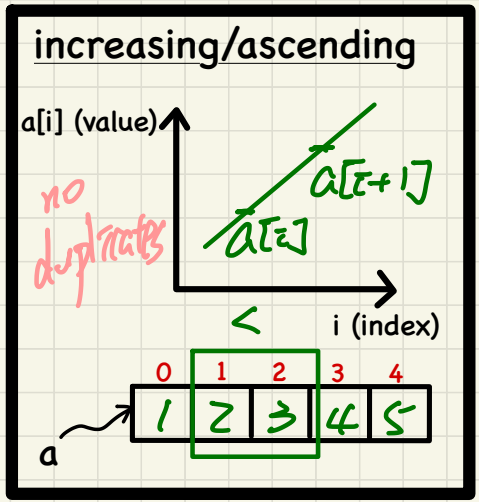
non-descending

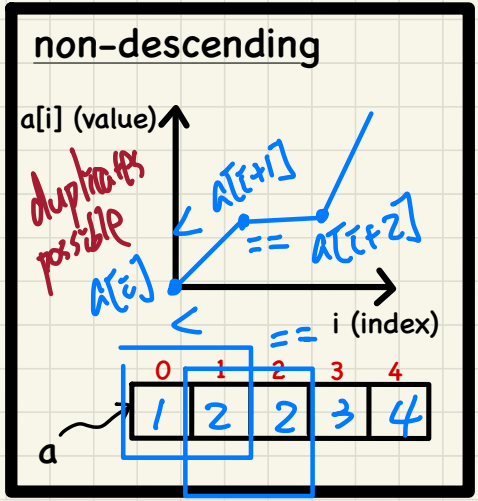
$\triangleq \neg$ (descending)

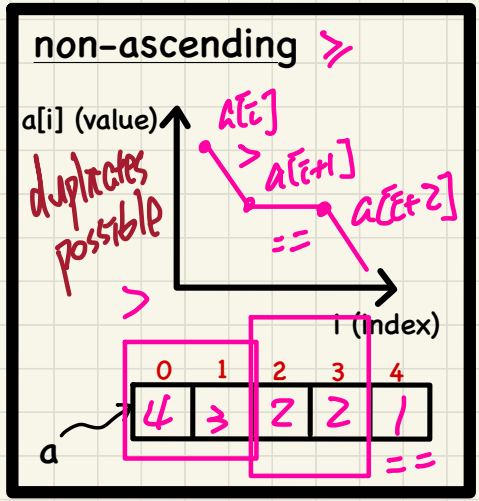
$\equiv \neg (a[i] > a[i+1])$

$\equiv a[i] \leq a[i+1]$









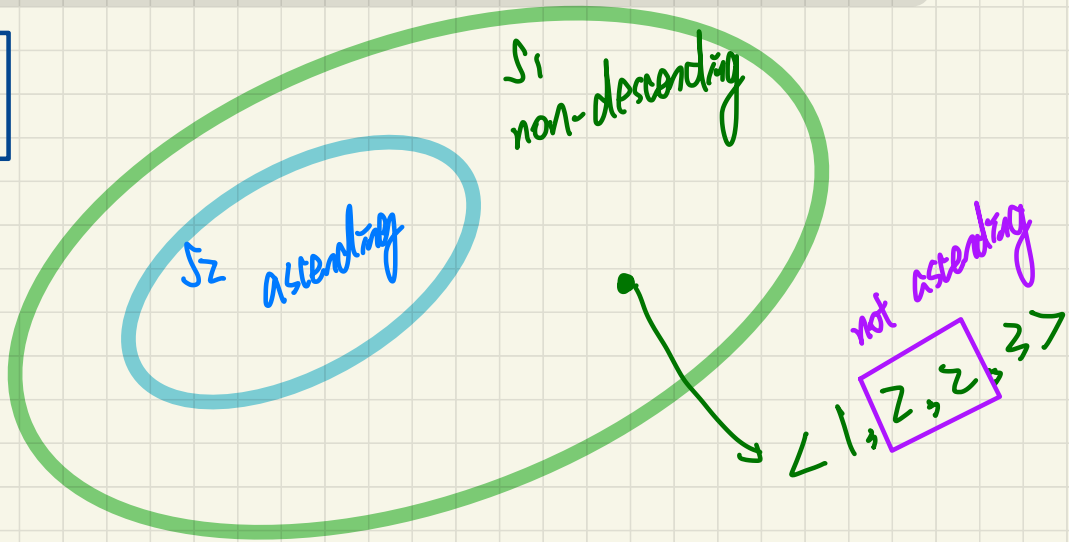
Exercise: Relating Sets of Sorted Arrays

Q. Consider the following two sets:

- S1: all arrays sorted in a **non-descending** order
- S2: all arrays sorted in an **ascending** order.

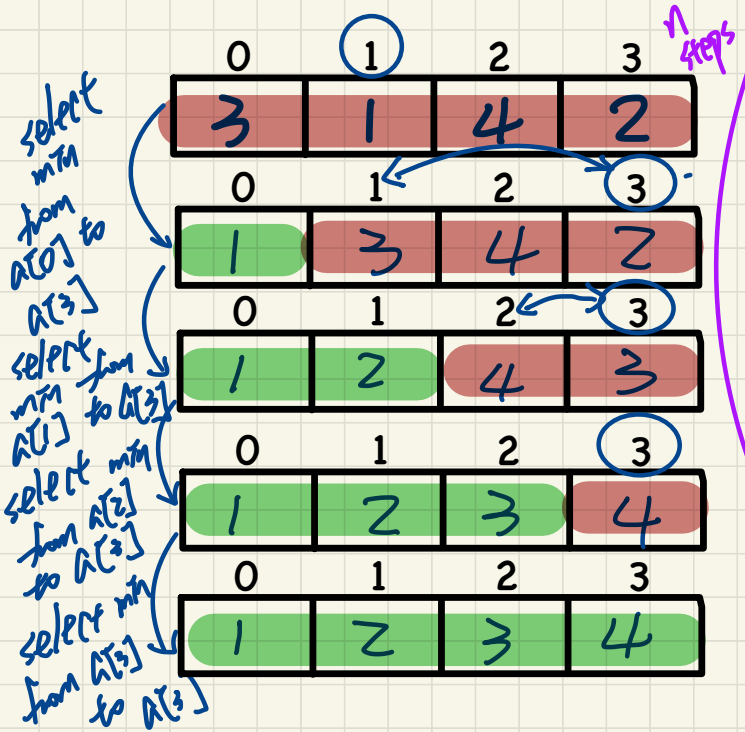
Formulate the relation between these two sets.

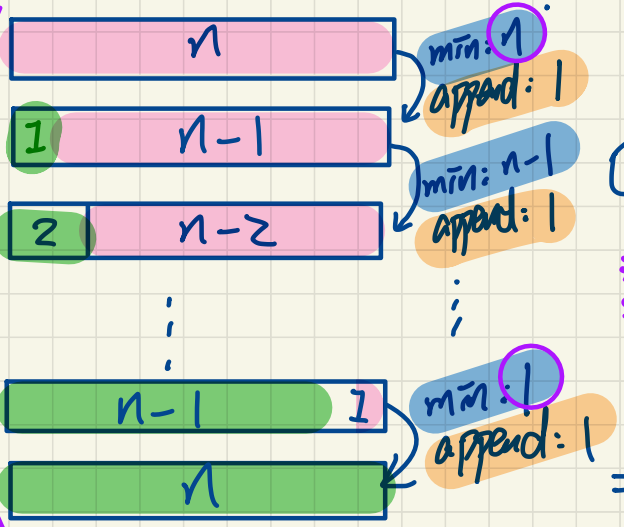
$S_2 \subset S_1$



Selection Sort

Keep **selecting** minimum from the **unsorted** portion and appending it to the end of **sorted** portion.





$$O(\underbrace{n}_{\#\text{steps}} \cdot \underbrace{1}_{\text{swap}} + [\overbrace{n}^{\text{1st selec.}} + (n-1) + \dots + \underbrace{1}_{\text{last selec.}}])$$

$$= O(n + n^2) = O(n^2)$$

Insertion Sort

$O(1)$

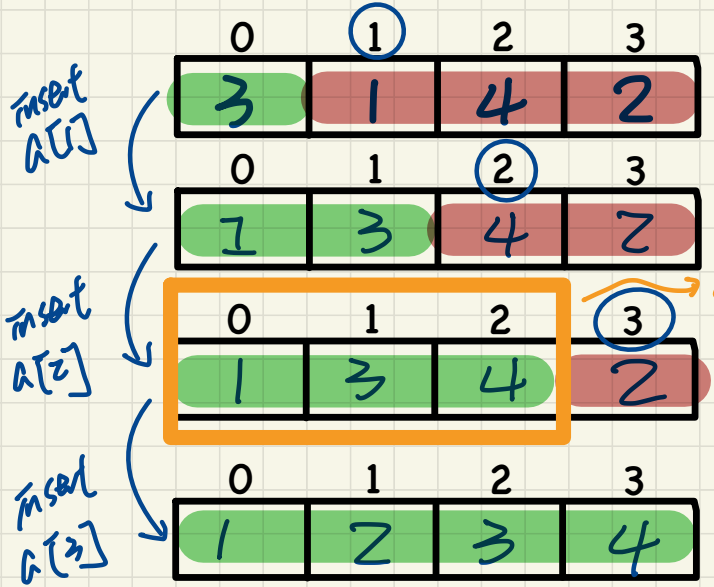
$O\left((n-1)\cdot 1 + [1+2+3+\cdots+(n-1)]\right)$

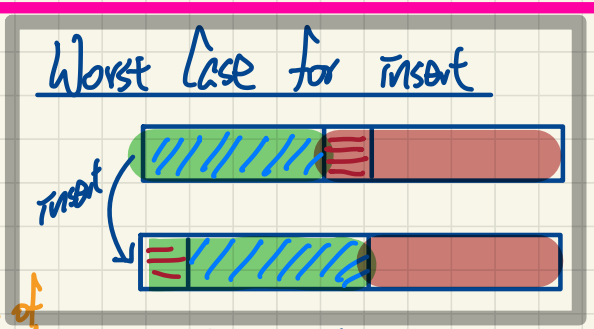
steps · choosing 1st unsorted item

$= O(n + n^2)$

$= O(n^2)$

Keep getting 1st element from the **unsorted** portion and **inserting** it to the **sorted** portion.





Step	choose	size of sorted portion	for insert max cost
1	a[1]	1	1
2	a[2]	2	2
3	a[3]	3	3
⋮	⋮	⋮	⋮
n-1	a[n-1]	n-1	n-1

Selection Sort: Deriving Asymptotic Upper Bound

```
1  void selectionSort(int[] a, int n)
2    for (int i = 0; i <= (n - 2); i ++)
3      int minIndex = i;
4      for (int j = i; j <= (n - 1); j ++)
5        if (a[j] < a[minIndex]) { minIndex = j; }
6      int temp = a[i];
7      a[i] = a[minIndex];
8      a[minIndex] = temp;
```

a.length

1

1

1

exec. according to values of i

exec. acc. to values (i, j)

i: 0, 1, 2, ..., n-2 → [0, n-2] → n-1 iterations

i	j	
0	1 2 ... n-1	n
1	1 2 ... n-1	n-1
2	2 ... n-1	n-2
⋮	⋮	⋮
n-2	n-1	2

$$O\left((n-1)\cdot 1 + \underbrace{[n+(n-1)+\cdots+2]}_{\#(i,j)}\cdot 1\right)$$

(n-1): # it. of outer loop; 1: Lb-Ub

$$\frac{(n+2)\cdot(n-1)}{2} \approx n^2$$

$$= O(n-1+n^2)$$

$$= O(n^2)$$

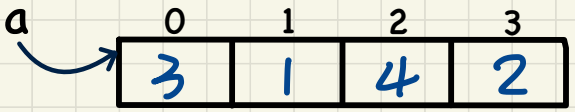
Insertion Sort: Deriving Asymptotic Upper Bound

```
1  void insertionSort(int[] a, int n)
2    for (int i = 1; i < n; i ++)
3      int current = a[i];
4      int j = i;
5      while (j > 0 && a[j - 1] > current)
6        a[j] = a[j - 1];
7        j --;
8      a[j] = current;
```

Exercise

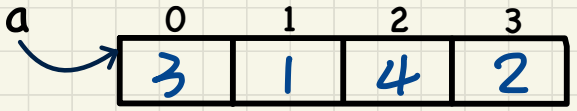
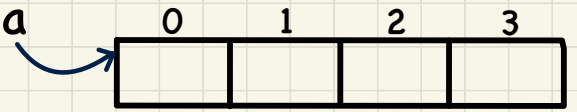
Selection Sort in Java

```
1  void selectionSort(int[] a, int n)
2    for (int i = 0; i <= (n - 2); i ++)
3      int minIndex = i;
4      for (int j = i; j <= (n - 1); j ++)
5        if (a[j] < a[minIndex]) { minIndex = j; }
6      int temp = a[i];
7      a[i] = a[minIndex];
8      a[minIndex] = temp;
```



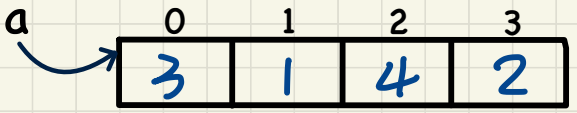
Inner Loop: select the next min from a[i] to a[n - 1] and put it to the end of the sorted region.

Outer Loop:
At the end of each iteration of the for-loop,
a is sorted from a[0] to a[i].

i	inner loop: j from ? to ?	midIndex at L6	after L6 - L8, a becomes?
			
			

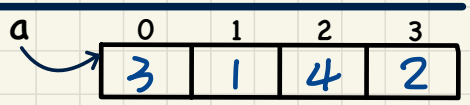
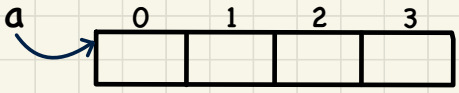
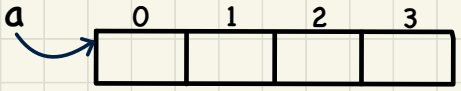
Insertion Sort in Java

```
1  void insertionSort(int[] a, int n)
2    for (int i = 1; i < n; i ++)
3      int current = a[i];
4      int j = i;
5      while (j > 0 && a[j - 1] > current)
6        a[j] = a[j - 1];
7        j --;
8      a[j] = current;
```

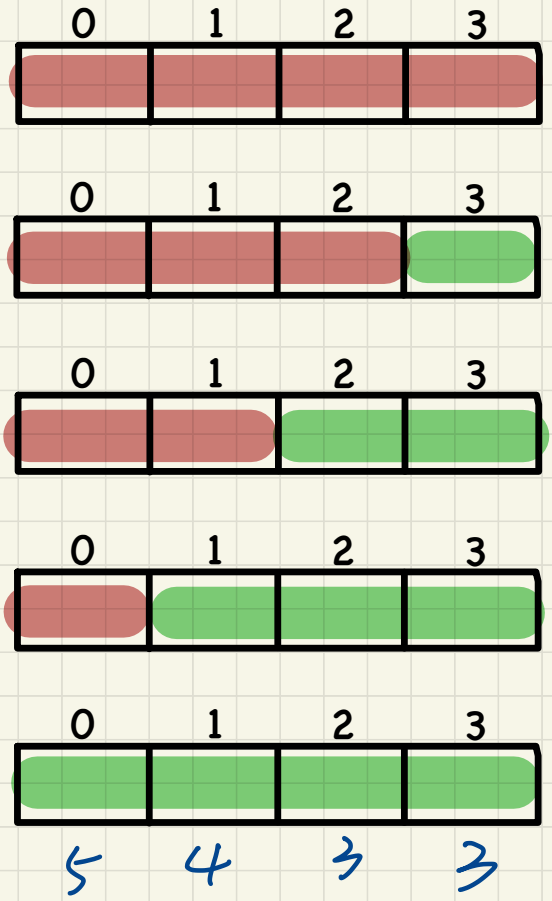


Inner Loop: find out where to insert current into a[0] to a[i] s.t. that part of a becomes sorted.

Outer Loop:
At the end of each iteration of the for-loop,
a is sorted from a[0] to a[i].

i	current after L3	j at L8	after L8, a becomes?
			
			
			

Exercise: Selection Sort vs. Insertion Sort



(1) sorted portion on the R

(2) non-ascending

Singly-Linked Lists (SLL): Visual Introduction

- A chain of connected nodes (via aliasing)
- Each node contains:
  - + reference to a data object
  - + reference to the next node
- Head vs. Tail
- The chain may grow or shrink dynamically.
- Accessing a position in a linear collection:
  - + Array uses absolute indexing: O(1)
  - + SLL uses relative positioning: O(n)

n: 1st node
n.data == "Alan"

n.next != null
n.next.data == "Mark"

n.next.next != null
n.next.next.data == "Tom"

n.next.next.next.data
↓
NullPointerException