Lecture 8 - January 30

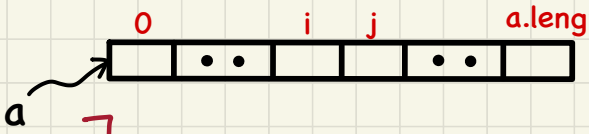
Arrays and Linked Lists

Exercise: Relating Sorting Orders
Selection vs. Insertion Sorts

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

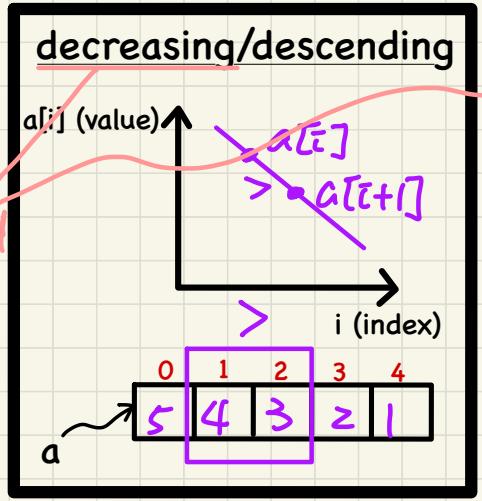


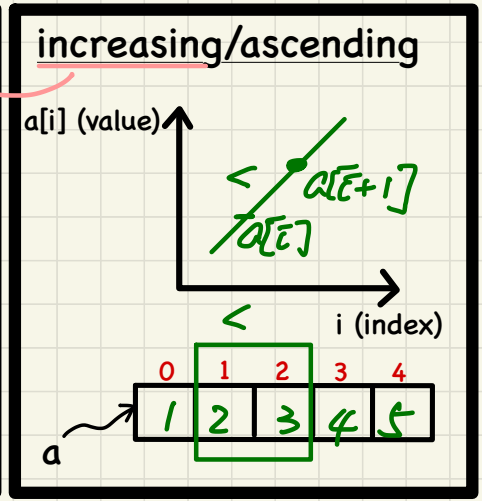
non-descending

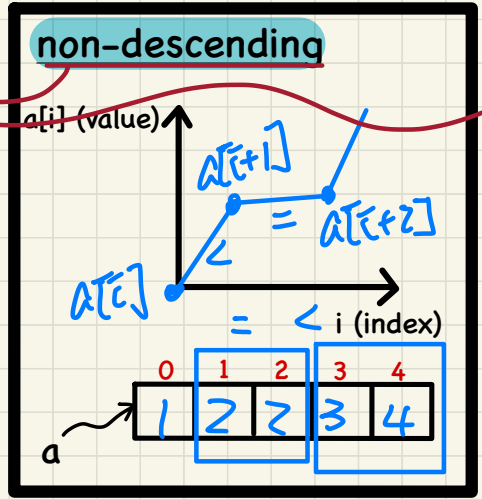
$\triangleq \neg(\text{descending})$

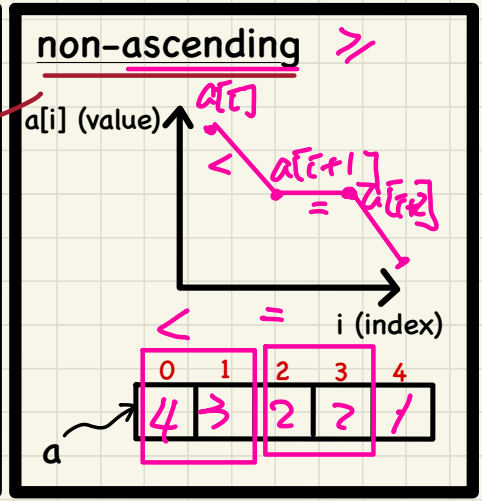
$\equiv \neg(a[i] > a[i+1])$

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









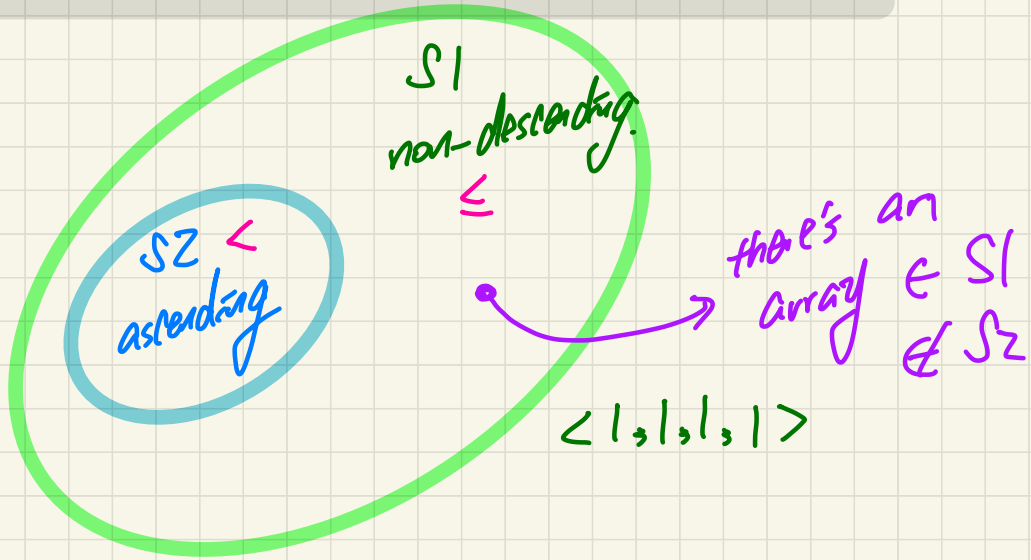
Exercise: Relating Sets of Sorted Arrays

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.

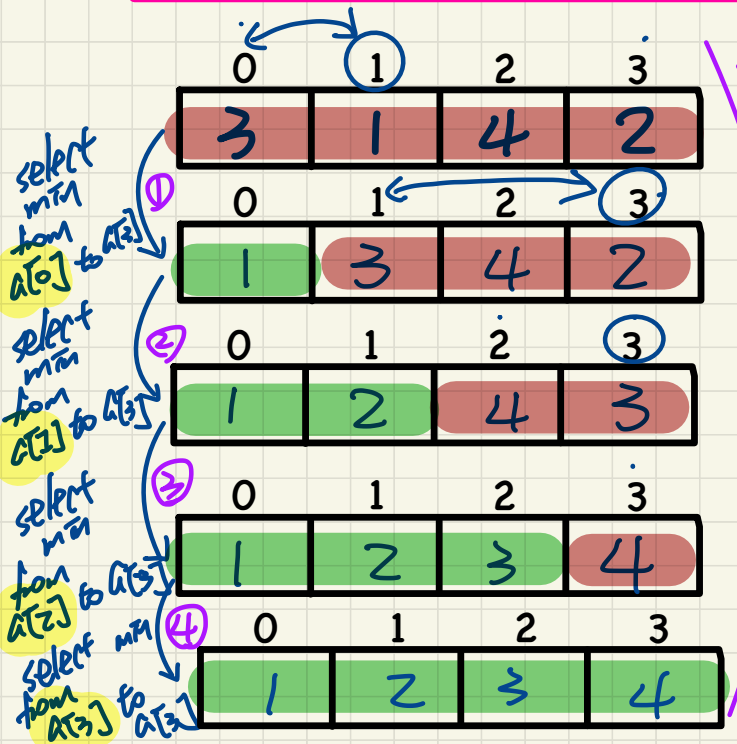
$S_2 \subset S_1$



Selection Sort

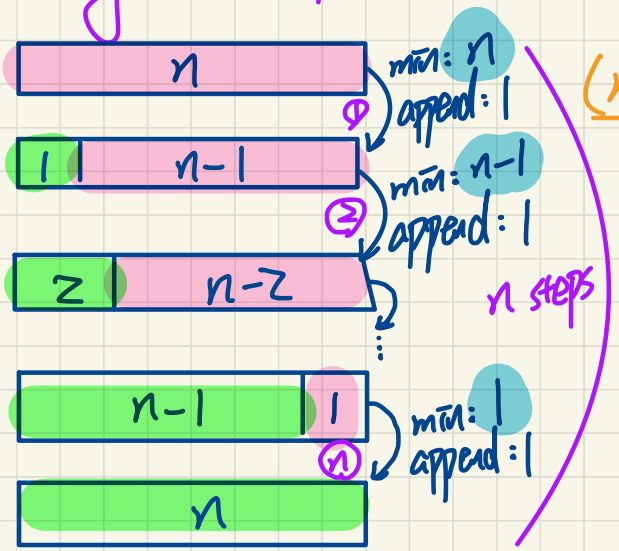
$O(\underbrace{n}_{\#\text{ steps}} \cdot \underbrace{1}_{\text{swap}} + [\underbrace{n}_{\text{1st sel.}} + (n-1) + \cdots + \underbrace{1}_{\text{last sel.}}]) = O(n + n^2) = O(n^2)$

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



steps? size of the ultimate sorted portion n

Decreasing Costs of select



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

Insertion Sort

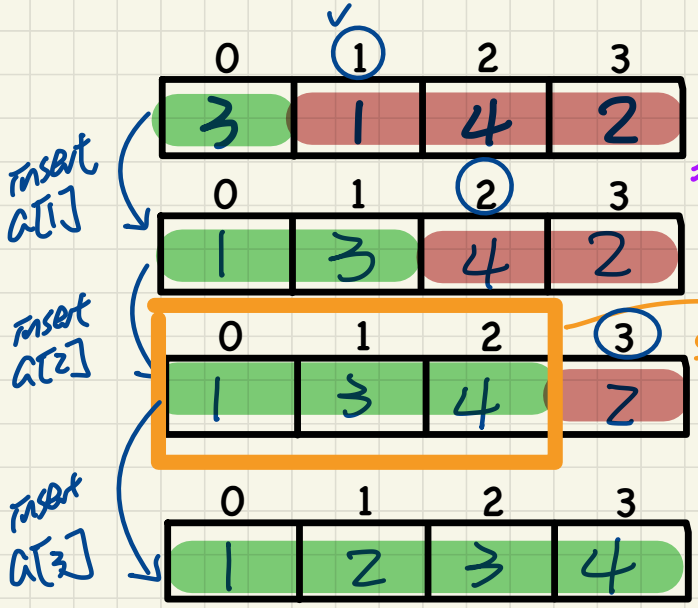
$O(1)$

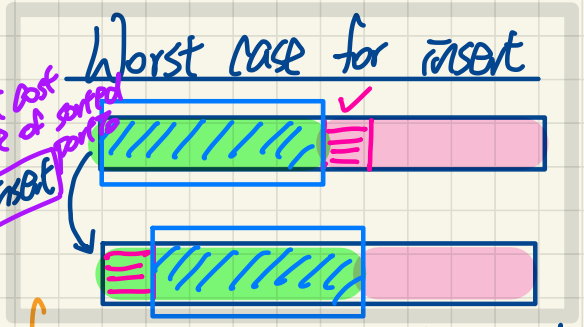
$O((n-1) \cdot 1 + [1+2+\cdots+(n-1)]) =$

steps — get left-most of unsorted — $\frac{(1+(n-1))\cdot(n-1)}{2}$

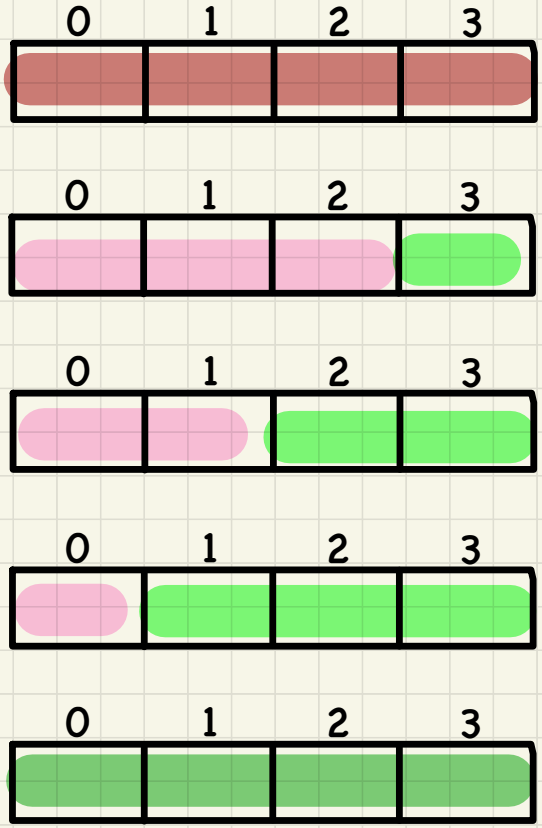
$O(n-1+n^2) = O(n^2)$

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





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



(1) Sorted portion is maintained on the right

(2) order: non-ascending.

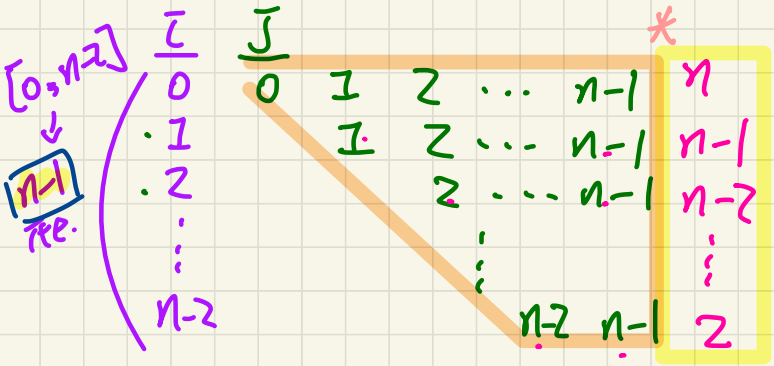
Selection Sort: Deriving Asymptotic Upper Bound

```
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

exec. according to values of i

exec. according to values of (i, j)



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

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

(n-1): # it. outer loop

1: L3, L6~L8

[ ]·1: L5

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

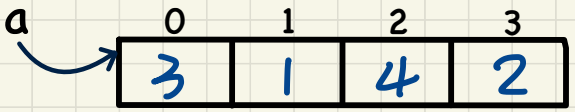
Insertion Sort: Deriving Asymptotic Upper Bound

```
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)

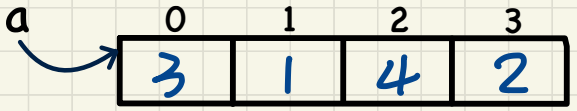
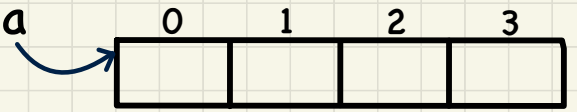
Selection Sort in Java

```
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;
```



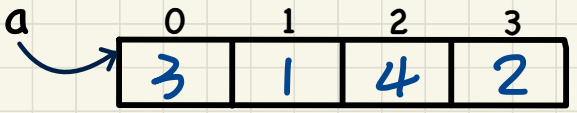
Inner Loop: select the next min from a[i] to a[n - 1] and put it to the end of the sorted region.

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?