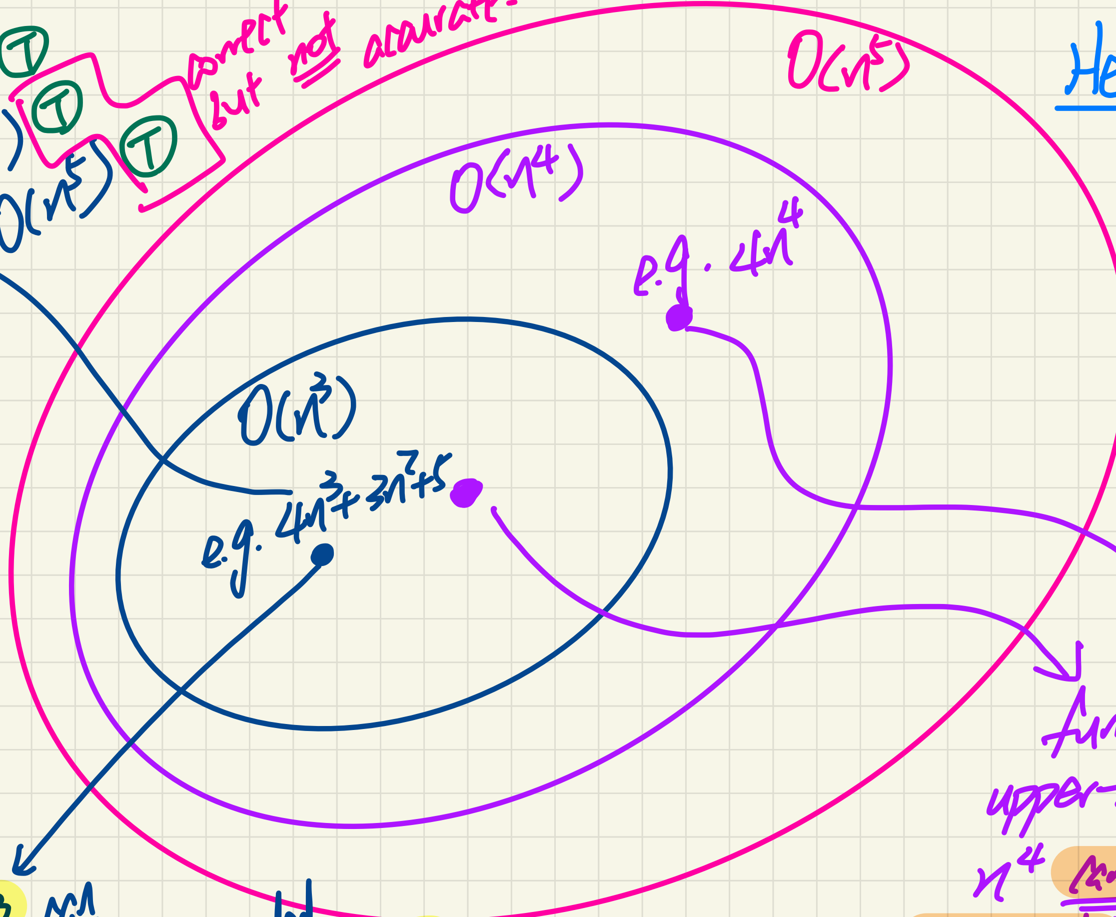
Lecture 1

Part E

Asymptotic Upper Bounds of Implemented Algorithms

is $O(n^3)$ T
$O(n^4)$ T
$O(n^5)$ T

correct but ~~not~~ accurate.



Hello World program

→ print("H.W.");

↳ $O(2^n)$

correct but incorrect

functions upper-bounded by n^4 cannot necessarily be upper-bounded by n^3

functions upper-bounded by n^3 can also be upper-bounded by n^4 and n^5

$4n^2 + 6n + 9$ is $O(n^2)$ ✓

$4n^2 + 6n + 9$ is $O(\cancel{4}n^2 + \cancel{3n})$ ✗

Determining the Asymptotic Upper Bound (1)

```
int maxOf (int x, int y) {
  int max = x;    O(1)
  if (y > x) {    O(1)
    max = y;      O(1)
  }
  return max;     O(1)
}
```

$$O(\underbrace{1+1+1+1}_{5}) = O(1)$$

$$4 = 4 \cdot \underline{\underline{n}}^0$$

Determining the Asymptotic Upper Bound (2)

```
int findMax (int[] a, int n) {
  currentMax = a[0];              O(1)
  for (int i = 1; i < n; ) {      O(n)
    if (a[i] > currentMax) {      O(1)
      currentMax = a[i]; }        O(1)
    i ++ }                        O(1)
  return currentMax; }            O(1)
```

$$O(1 + 1 + n \cdot (1 + 1 + 1))$$

$$= O(2 + 3n)$$

$$= O(n)$$

Determining the Asymptotic Upper Bound (3)

```
boolean containsDuplicate (int[] a, int n) {
  for (int i = 0; i < n; ) { O(n)
    for (int j = 0; j < n; ) { O(n)
      if (i != j && a[i] == a[j]) { O(1)
        return true; } O(1)
      j ++; } O(1)
    i ++; } O(1)
  return false; } O(1)
```

$$O(n \cdot (\underbrace{n \cdot (1 + 1 + 1) + 1}_{3n+1}) + 1)$$

$$= O(3n^2 + n + 1)$$

$$= \boxed{O(n^2)}$$

Determining the Asymptotic Upper Bound (4)

```
int sumMaxAndCrossProducts (int[] a, int n) {
  int max = a[0];  O(1)
  for(int i = 1; i < n; i ++) {  O(n)
    if (a[i] > max) { max = a[i]; }  O(1)
  }
  int sum = max;  O(1)
  for (int j = 0; j < n; j ++) {  O(n)
    for (int k = 0; k < n; k ++) {  O(n)
      sum += a[j] * a[k]; } }  O(1)
  return sum; }  O(1)
```

$$O\left(\underline{1+1} + (n \cdot 1) + 1 + n \cdot n \cdot 1\right)$$

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

$$= O(n^2)$$

How many #'s in [a, b] = b − a + 1

Determining the Asymptotic Upper Bound (5)

```
1  int triangularSum (int[] a, int n) {
2    int sum = 0;                           O(1)
3    for (int i = 0; i < n; i ++) {         O(n)
4      for (int j = i; j < n; j ++) {
5        sum += a[j]; }                     O(1)
6    return sum; }                          O(1)
```

outer-loop

i
0
1
2
⋮
n−1

inner loop when i = 0

j
0 1 … (n−1)
1 … (n−1)
2 … (n−1)
⋮
(n−1)

Sum of Arithmetic Sequence

$$1 + 2 + 3 + \cdots + n = \boxed{\frac{(1+n)\cdot n}{2}}$$

$$i + (i + c) + (i + 2c) + (i + 3c) + \cdots + (i + (n-1)c) = \boxed{\frac{(i + (i + (n-1)c))\cdot n}{2}}$$

$i + 0\cdot c$, +c, +c, +c

$$O(1 + 1 + \underbrace{n}_{\text{when } i=0}\cdot 1 + \underbrace{(n-1)}_{\text{when } i=1}\cdot 1 + \cdots + \underbrace{1}_{\text{when } i=n-1}\cdot 1)$$

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

$$= O\left(2 + \frac{(n+1)\cdot n}{2}\right)$$

$$= O(n^2)$$

Lecture 2

Part A

Asymptotic Upper Bounds of Array Operations

Inserting into an Array

$[0, i-1] = (i-1)-0+1 = i$

assume: $0 \le i \le a.length - 1$ ($a.length = n$)

```
String[] insertAt(String[] a, int n, String e, int i)
  String[] result = new String[n + 1];                          O(1)
  for(int j = 0; j <= i - 1; j ++){ result[j] = a[j]; }         O(n-1) = O(n)
  result[i] = e;                                                O(1)
  for(int j = i + 1; j <= n; j ++){ result[j] = a[j-1]; }       O(n-1) = O(n)
  return result;                                                O(1)
```

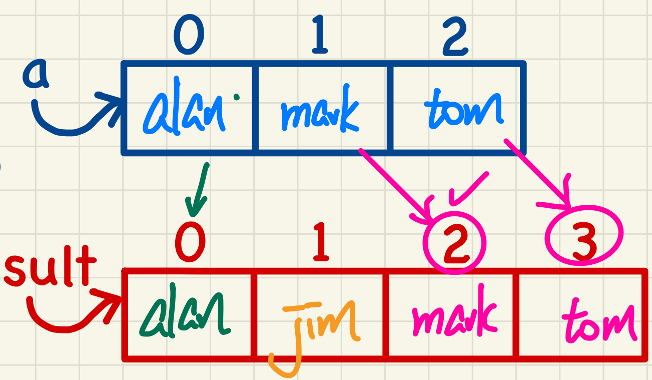
result[0] = a[0]

$[i+1, n] = n-(i+1)+1 = \boxed{n-i}$ → n when $i=0$ → max # of iterations

Example:

insertAt({alan, mark, tom}, 3, jim, 1)

(a = {alan, mark, tom}, n = 3, e = jim, i = 1)



result[2] = a[2-1]

result[3] = a[3-1]

RT:

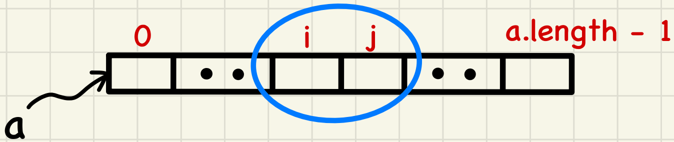
$O(1+n+1+n+1) = O(n)$

<u>**Lecture 2**</u>

<u>**Part B**</u>

Asymptotic Upper Bounds
Selection Sort vs. Insertion Sort

Sorting Orders of Arrays



decreasing/descending : $a[i] > a[j]$

increasing/ascending : $a[i] < a[j]$

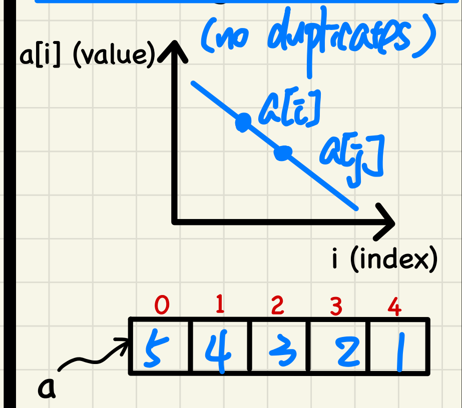
non-descending : $\neg(a[i] > a[j])$

$\equiv a[i] \le a[j]$

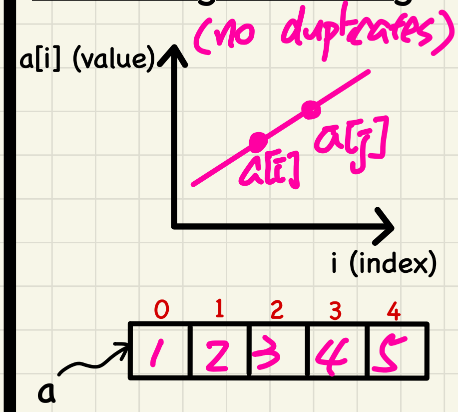
non-ascending : $\neg(a[i] < a[j])$

$\equiv a[i] \geq a[j]$

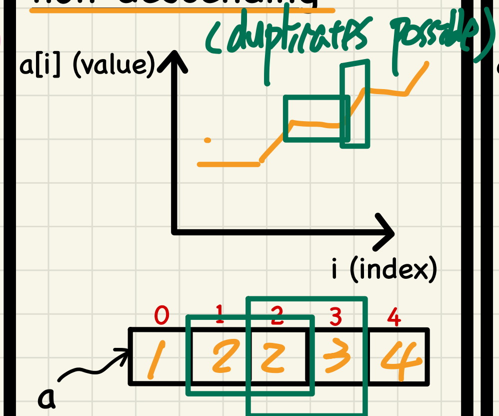
decreasing/descending



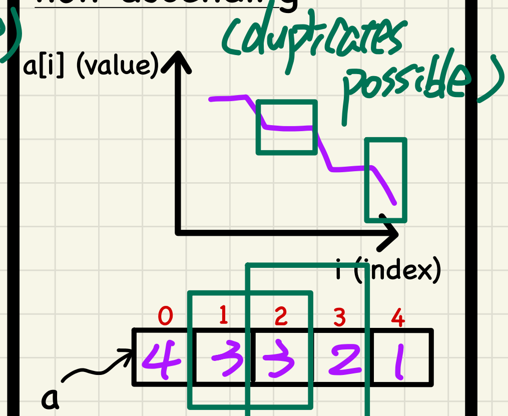
increasing/ascending



non-descending



non-ascending



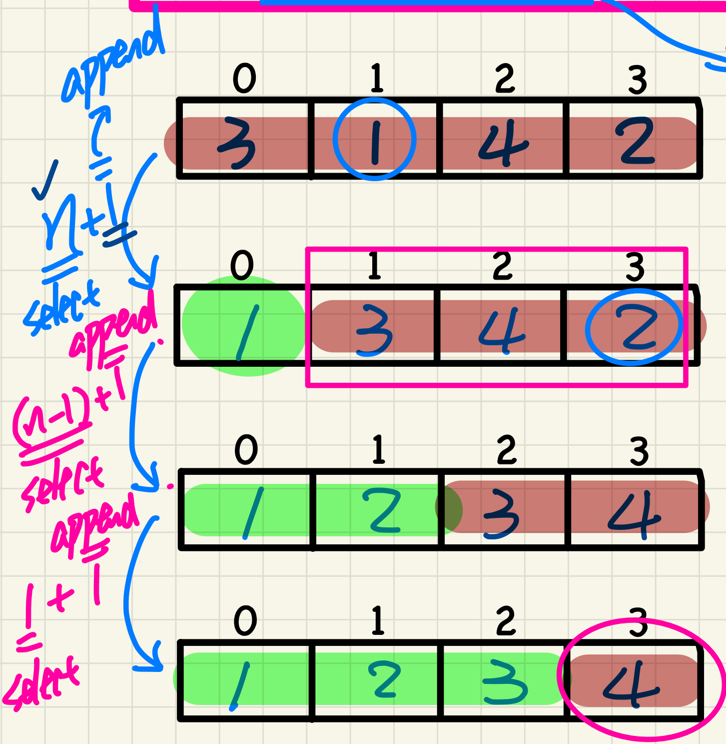
Selection Sort

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

more expensive

from L to R

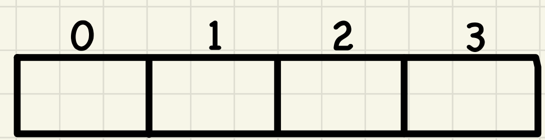
cheaper



unsorted

sorted

$$O(\underbrace{n}_{\text{append to end of sorted portion}} + \underbrace{(n+1)}_{\text{1st selection}} + \underbrace{n}_{\text{2nd selection}} + \cdots + 1) = O(n^2)$$

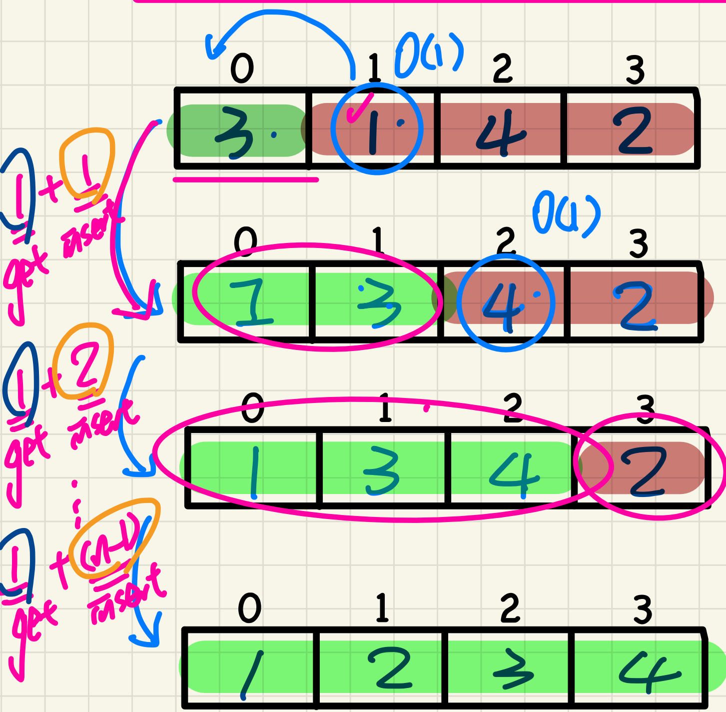


Insertion Sort

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

cheaper

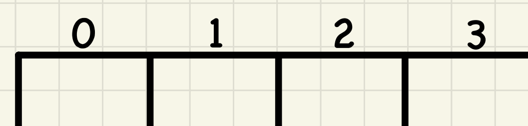
more expensive



unsorted

sorted

$$O\Big(\underbrace{n}_{\text{get 1st elements of unsorted portion}} + \big(\underbrace{1}_{\text{1st insert}} + \underbrace{2}_{\text{2nd insert}} + \cdots + (n-1)\big)\Big) = O(n^2)$$



Selection Sort in Java

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

```
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;
```

4

3

O(1)

swap a[i] and a[minIndex]

i	j
0	0 … (n-1)
1	1 … (n-1)
2	2 … (n-1)
⋮	
n-2	(n-2) (n-1)

a →

0	1	2	3
3	1	4	2

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]. before i++ ?

i	inner loop: j from ? to ?	midIndex at L6	after L6 – L8, a becomes?
0	0 1 .. (n-1) 3	1	a: [0] ~~3~~ 1, [1] ~~1~~ 3, [2] 4, [3] 2
1	1 .. (n-1) 3	3	a: [0] 1, [1] ~~3~~ 2, [2] 4, [3] ~~2~~ 3

$O(2+3+4+\dots+n) = O(n^2)$

Insertion Sort in Java

$[0, n-1] = n$

exit when: $\neg(j>0 \land a[j-1]>c)$

$= j \le 0 \lor a[j-1] \le c$

worst case for while-loop to exit

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

	0	1	2	3
a	3	1	4	2

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]. (before exit)

i	current after L3	j at L8	after L8, a becomes?
1	a[1] 1	0	a: [1, 3, 4, 2]
2	a[2] 4	2	a: [1, 3, 4, 2]
3	a[3] 2	1	a: [1, 2, 3, 4]

I	J
1	1 0
2	2 1 0
3	3 2 1 0
⋮	
n-1	(n-1) … 0

In-Place Sorting

↳ Sort by modifying directly the input array (without intermediate storage)