6.046
Lecture 1

Insertion Sort,
Mergesort

Analysis of Algorithms:

theoretical study of computer program performance and resource usage.

What's more important than performance?

- correctness, simplicity, maintainability, modularity, security, ... user friendliness

Why study algs [redacted] performance

- feasible vs. infeasible
- currency to pay for other important [redacted] properties
- bottom of the heap

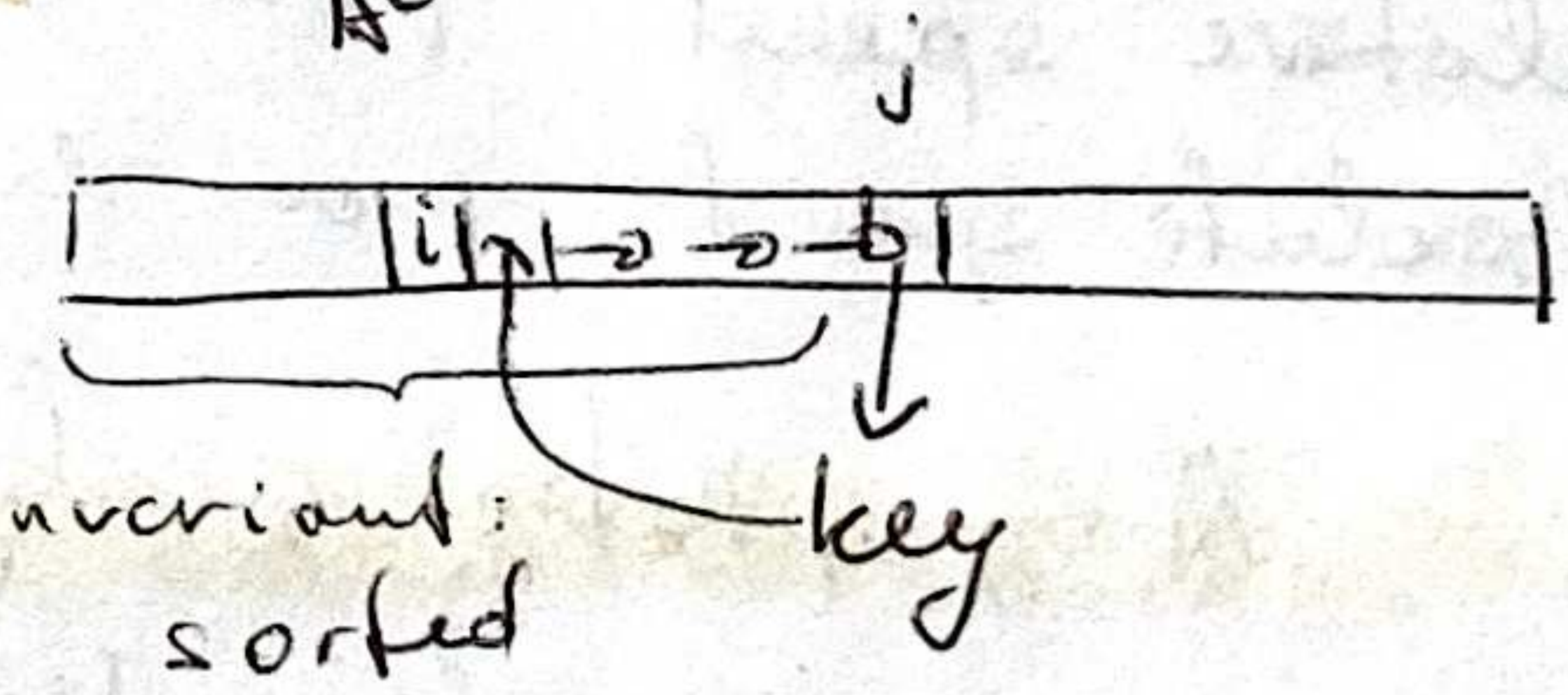
Sorting Problem

Input: sequence $\langle a_1, a_2 \ldots a_n \rangle$ of numbers
Output: permutation $\langle a_1', a_2' \ldots a_n' \rangle$
s.t. ($) $a_1' \le a_2' \le \ldots \le a_n'$

Insertion Sort (A, n) // sorts A[1 ... n]

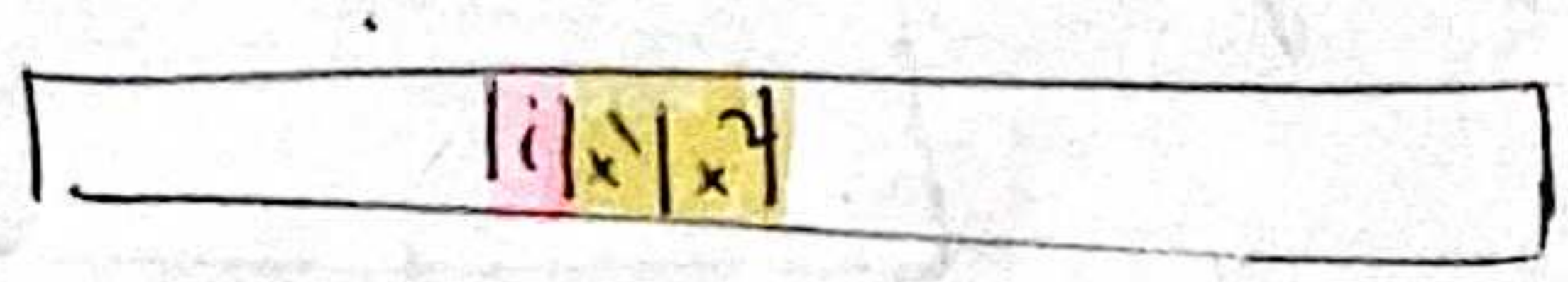
```
for j ← 2 to n
    do key ← A[j]
       i ← j-1
       while i > 0 and A[i] > key
          do A[i+1] ← A[i]
             i ← i-1
       A[i+1] ← key
```

A[i] ≤ key



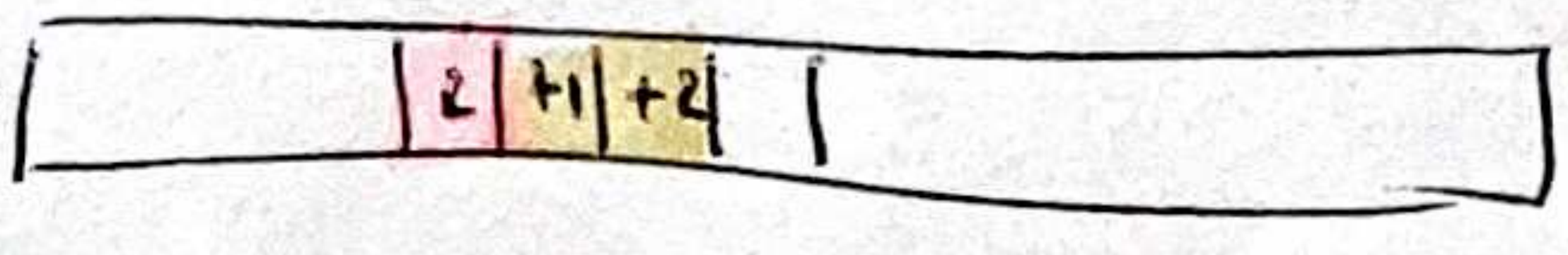
Ex:
8 (2) 4 9 3 6
2 8 (4) 9 3 6
2 4 8 (9) 3 6
2 4 8 9 (3) 6
2 3 4 8 9 (6)
2 3 4 6 8 9

after completing [redacted] a while loop iteration:

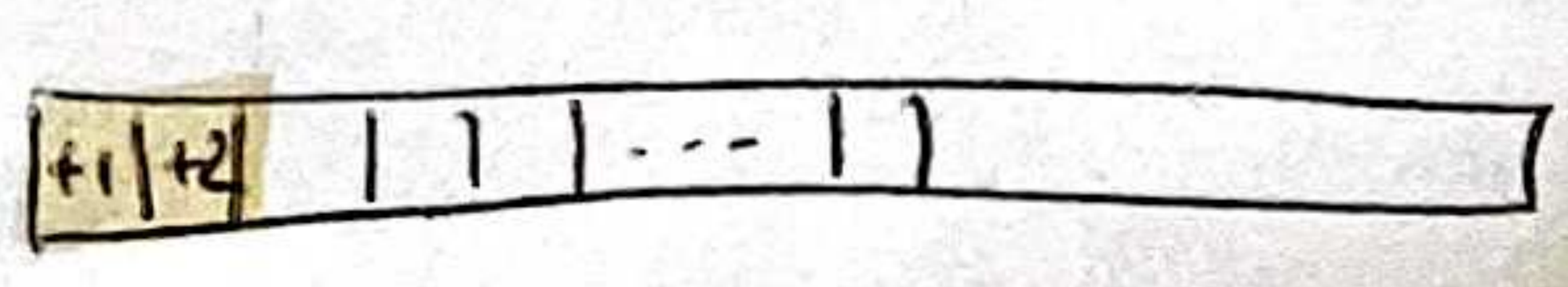


$A[i+1] = A[i+2]$

- so key can be inserted into A[i+1] if A[i] ≤ k
- or A[i] can be copied into A[i+1] to continue



.... border case?!



Running time

- depends on input (already sorted, or reverse-sorted)
- depends on input size (6 elem. vs. 6×10^9)
  - parameterize in input size
- want upper bounds
  - represents a guarantee to user

Kinds of analysis

Worst-case (usually):

$T(n)$ = max time on any input of size n. max turns relation to function

Average case (sometimes):

$T(n)$ = expected time over all inputs of size n

(Need an assumption of statistical distribution of inputs)

Best case: (bogus)

What is insertion sort's worst-case time?

Depends on computer

- relative speed (on same machine)
- absolute speed (on diff. machines)

Big Idea: Asymptotic analysis

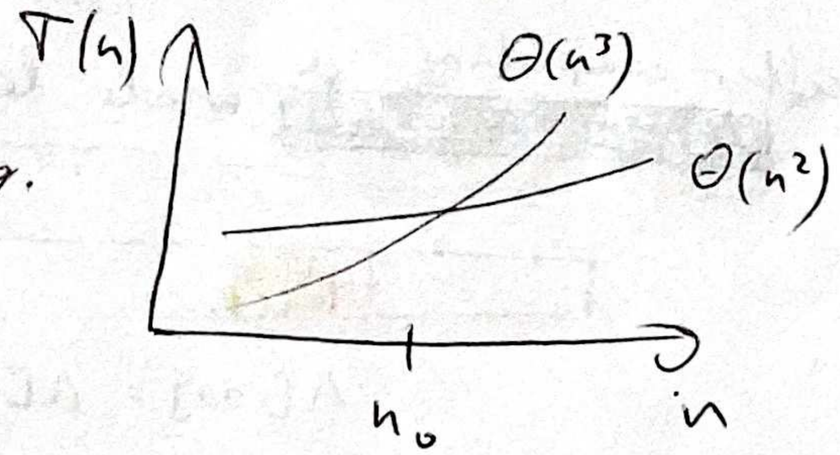
- ignore machine-dependent constants
- look at <u>growth</u> of $T(n)$ as $n \to \infty$

Asymptotic notation

Θ-notation: Drop low-order terms, ignore leading constants

Ex: $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$

as $n \to \infty$, $\Theta(n^2)$ alg always beats a $\Theta(n^3)$ alg.



Insertion sort analysis

worst case: input reverse-sorted

$$T(n) = \sum_{j=2}^{n} \Theta(j) = \Theta(n^2)$$

arithmetic series

$$\sum_{k=1}^{m} k = \frac{m(m+1)}{2}$$

0, 1 ---- 99, 100 — 50 times

Is insertion sort fast?
- moderately so
- not for large n

Merge sort

$T(n)$

abuse of notation → $\Theta(1)$

sloppy → $2T(n/2)$

$\Theta(n)$

Merge sort A [1 --- n]

1. If $n = 1$ done
2. recursively sort A [1 ---- $\lceil n/2 \rceil$] and A [$\lceil n/2 \rceil + 1$ --- n]
3. merge 2 sorted lists

Recurrence: (usually omit const. base case)

$$T(n) = \begin{cases} \Theta(1), & \text{if } n = 1 \\ 2T(\frac{n}{2}) + \Theta(n), & n > 1 \end{cases}$$

key subroutine: Merge

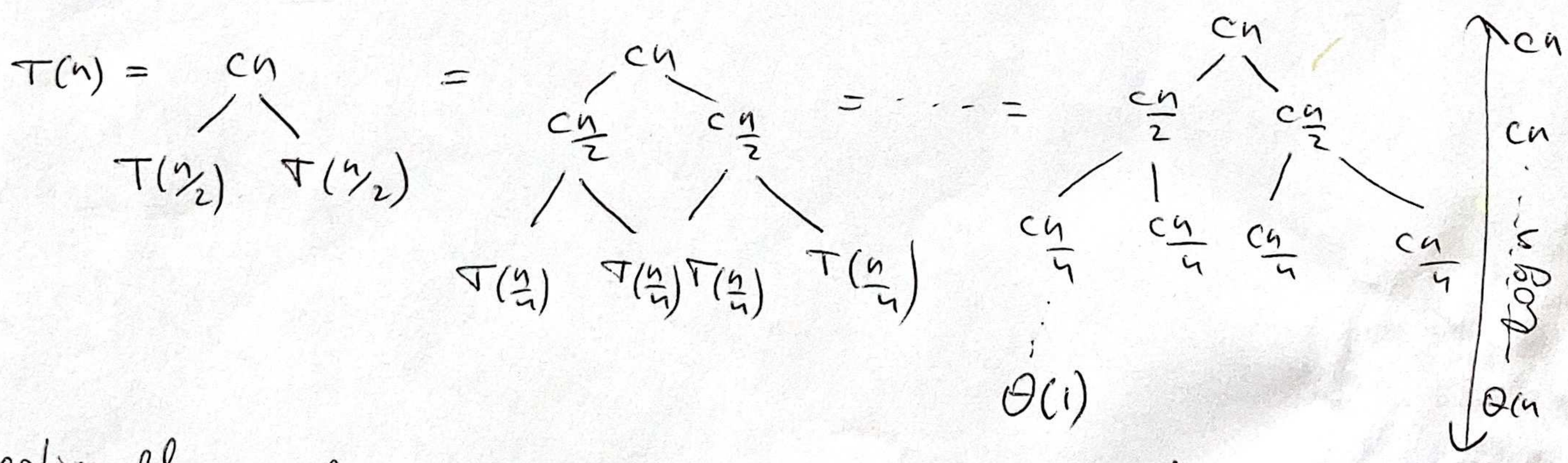
20 12
13 11
7 9
2 1

1 2 7 ----

Time = $\Theta(n)$ on n total elements

Recursion tree:

$T(n) = 2T(\frac{n}{2}) + cn$, $c > 0$ assume n is power of 2



Total: $cn \log n + \Theta(n)$

$= \Theta(n \log n)$

optimally combine insertion sort at lower levels

{ on a large n merge sort is faster than insertion sort ~ n > 30