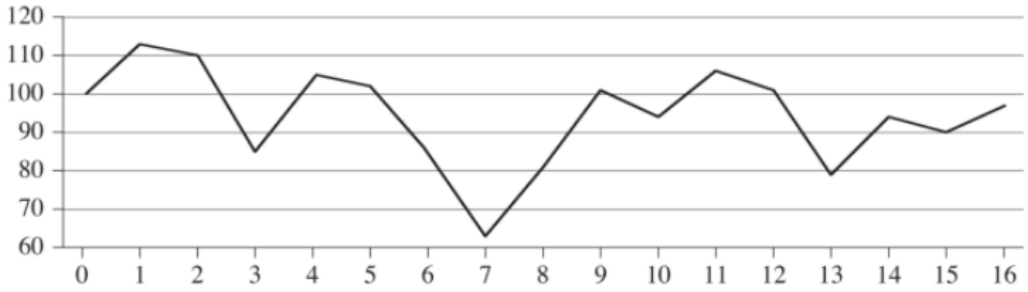
Given the price of a stock over an n-day period, determine the best time to have bought and sold 1000 shares of that stock. (Buy and sell once, on different days.)



Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97

Establish the problem.

- specifications
- examples

Identify avenues of attack.

- targets

brute force: for every pair of buy-sell days, find the max – O(n^2)

- approach
- paradigms and patterns

easy split – divide range of days in half, find best pair in the first half and the second half, then combine…

Define the algorithm.

- generalize / define subproblems

task: find the best buy-sell pair over P[start..end], inclusive

input: array P of prices, start and end

output: buy-sell pair

- base case(s)

2 days (end = start+1) – buy on start, sell on end

- main case

mid ← (start+end)/2

(buy1,sell1) ← stocks(P,start,mid)

(buy2,sell2) ← stocks(P,mid+1,end)

find the best pair (buy3,sell3) where buy3 is in the range start..mid and sell3 is in the range mid+1..end

return higher profit of (buy1,sell1), (buy2,sell2), (buy3,sell3) (or either one if the profit is the same)

- top level
  - initial subproblem
  - setup
  - wrapup
- special cases
- algorithm

Show termination and correctness.

- termination
  - size
  - making progress
  - the end is reached
- correctness
  - establish the base case(s)
  - show the main case
  - final answer

Determine efficiency.

- implementation
- time and space

brute force for finding (buy3,sell3) means trying every pair of a day in the first half and a day in the second half – $O(n^2)$

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

using the recurrence relations tables yields $T(n) = \Theta(n^2)$ – not an improvement over brute force

- room for improvement

can we reduce the work of the combine step, in particular, of finding (buy3,sell3)?