1. We are given an array A[1..*n*] with *n* entries, with each entry holding a distinct number. The sequence of values A[1], A[2], …, A[*n*] is unimodal: for some index *p* between 1 and *n*, the values in the array entries increase up to the position *p* in A and then decrease the remainder of the way until position *n*. Find the index *p* in O(log *n*) time.

2. You are consulting for a small investment company, and they have the following problem they want to solve. They are doing a simulation in which they are looking at *n* consecutive days of a given stock, at some point in the past. Let's number the days 1, 2, ..., *n*. For each day *i*, they have a price *p(i)* per share for the stock on that day. (Assume for simplicity that the price was fixed during each day). Suppose that during this time period, they wanted to buy 1000 shares on some day and sell all these shares on some (later) day.

They want to know: when should they have bought and when should they have sold in order to maximize the profit? If there is no way to make money during the *n* days, you should report this instead. Show how to find the correct buy/sell days in O(*n*log*n*).

3. You have *n* coins that are all supposed to be gold coins of the same weight, but you know that one coin is fake and weights less than the others. You have a balance scale: you can put any number of coins on each side of the scale at one time, and it will tell you if the two sides weight the same, or which side is lighter if they don’t weight the same. Outline an algorithm for finding the fake coin. What is the running time of your algorithm?

1. We are given an array A[1..*n*] with *n* entries, with each entry holding a distinct number. The sequence of values A[1], A[2], …, A[*n*] is unimodal: for some index *p* between 1 and *n*, the values in the array entries increase up to the position *p* in A and then decrease the remainder of the way until position *n*. Find the index *p* in O(log *n*) time.

A[1..n]

p=6

	1	2	3	4	5	6	7	8
A	2	5	9	11	14	18	12	6

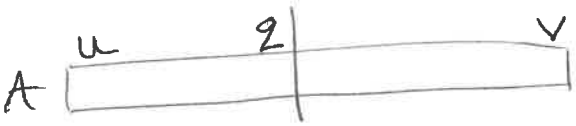
n=8

$T(n) = O(\log n)$

- algorithm recurses on 1 subproblem of size $n/2$
- divide + combine take $\Theta(1)$

$T(n) = T\left(\frac{n}{2}\right) + \Theta(1)$

Divide-and-conquer



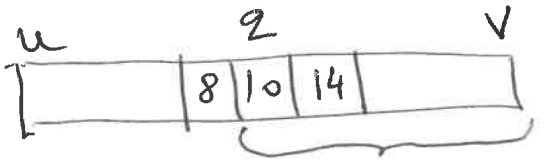
Divide $\Theta(1)$

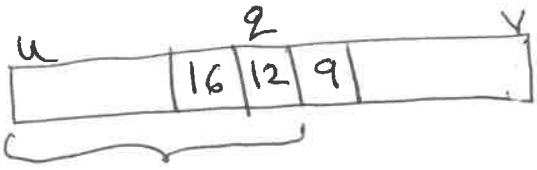
$q = \left\lfloor \frac{u+v}{2} \right\rfloor$

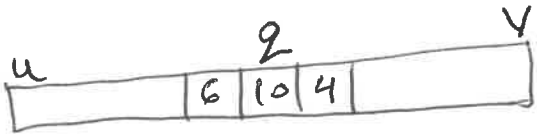
Conquer $1 \cdot T\left(\frac{n}{2}\right)$

There are 3 possible cases:

- if $A[q-1] < A[q] < A[q+1]$ then recurs on $A[q..v]$
- if $A[q-1] > A[q] > A[q+1]$ then recurs on $A[u..q]$
- if $A[q-1] < A[q] > A[q+1]$ then $p = q$. Return q.







Combine: nothing

RT analysis

$T(n) = T\left(\frac{n}{2}\right) + \Theta(1) \Rightarrow \boxed{T(n) = \Theta(\log n)}$

2. You are consulting for a small investment company, and they have the following problem they want to solve. They are doing a simulation in which they are looking at *n* consecutive days of a given stock, at some point in the past. Let's number the days 1, 2, ..., *n*. For each day *i*, they have a price *p(i)* per share for the stock on that day. (Assume for simplicity that the price was fixed during each day). Suppose that during this time period, they wanted to buy 1000 shares on some day and sell all these shares on some (later) day.

They want to know: when should they have bought and when should they have sold in order to maximize the profit? If there is no way to make money during the *n* days, you should report this instead. Show how to find the correct buy/sell days in O(*n*log*n*).

$p[1..n]$

	1	2	3	4	5	6	7
p	9	1	5	6	4	2	3

$n = 7$

buy in day 2 and sell in day 4
$\Rightarrow$ profit = \$5/share

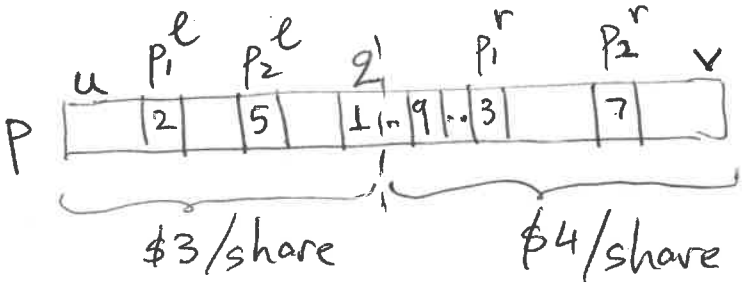
Divide-and-conquer for $O(n \cdot \log n)$

- algorithm which recurses on 2 subproblems of size $n/2$
- divid + combine take $O(n)$

$T(n) = 2 \cdot T(\frac{n}{2}) + O(n)$

- Divide $\Theta(1)$

divide $p[u..v]$ into 2 subarrays $p[u..q]$ and $p[q+1..v]$ where $q = \lfloor \frac{u+v}{2} \rfloor$



- Conquer $2 \cdot T(\frac{n}{2})$

solve 2 subproblems

$(p_1^l, p_2^l) = \text{Buy-Shares}(p, u, q)$

$(p_1^r, p_2^r) = \text{Buy-Shares}(p, q+1..v)$

- Combine $\Theta(n)$

return the best solution between

- (p_1^l, p_2^l) – opt. sol. first subproblem
- (p_1^r, p_2^r) – opt. sol. second subproblem
- optimal solution when we buy in $p[u..q]$ and sell in $p[q+1..v]$
  - find index of element with min value in $p[u..q]$
  - find index of element with max value in $p[q+1..v]$

need one traversal of $p[u..v]$ $\Theta(n)$

RT analysis

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \Theta(n)$$

$$\boxed{T(n) = \Theta(n \cdot \log n)}$$

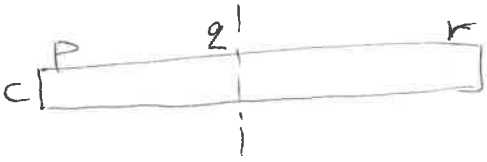
3. You have *n* coins that are all supposed to be gold coins of the same weight, but you know that one coin is fake and weights less than the others. You have a balance scale: you can put any number of coins on each side of the scale at one time, and it will tell you if the two sides weight the same, or which side is lighter if they don't weight the same. Outline an algorithm for finding the fake coin. What is the running time of your algorithm?

Divide-and-conquer algorithm

-general problem: find the fake coin in c[p..r]

• base case: if p=r, then return c_p

If the number of coins is even



Divide the coins into 2 groups c[p..q] and c[q+1..r] where $q=\lfloor \frac{p+r}{2} \rfloor$

Weight the 2 groups

Conquer - recurs on the group of coins of smaller weight

Combine - nothing

If the number of coins is odd

- take away the first coin c[p]

Divide the remaining coins into 2 groups with the same number of coins c[p+1..q] and c[q+1..r], where $q=\lfloor \frac{p+1+r}{2} \rfloor$

Weight the 2 groups

Conquer

- if the 2 groups have the same weight ⇒ return c[p]
- otherwise recurs on the group of coins of smaller weight

Combine - nothing

RT analysis

$$T(n)=T\left(\frac{n}{2}\right)+\Theta(1)$$

$$\boxed{T(n)=\Theta(\lg n)}$$