Decrease-and-Conquer

Decrease-by-Constant-Factor Algorithms

- In this variation of decrease-and-conquer, instance size is reduced by the same factor (typically, 2)

- Examples:
  - Binary search and the method of bisection
  - Exponentiation by squaring
  - Multiplication - Russian peasant method
  - Fake-coin puzzle
  - Josephus problem

Exponentiation by Squaring

Compute a^n where n is a nonnegative integer

Exponentiation by Squaring

Compute a^n where n is a nonnegative integer

The problem can be solved by applying recursively the formulas:

$$a^n = \begin{cases} (a^{\frac{n}{2}})^2 & n \text{ is even} \\ (a^{\frac{n-1}{2}})^2 . a & n \text{ is odd} \\ 1 & n == 0 \end{cases}$$

Exponentiation by Squaring

Compute a^n where n is a nonnegative integer

The recurrence will be

- $T(0) = 0$
- $T(n) = T\left(\frac{n}{2}\right) + 1$
- $T(n) = T(1) + \sum_{i=1}^{k} f(b^i)$, $where\ n = b^k\ and\ k = \log\frac{n}{b}$
- $T(n) = 1 + \sum_{i=1}^{log\ n} 1 = 1 + log\ n$
- $T(n) \in \theta(log\ n)$

Binary Search

- Searching in sorted array for a key
- Compare *key* with array's middle element $A[m]$

$$\begin{cases} match & \rightarrow stop, returm\ m \\ key < A[m] & \rightarrow search\ recursively\ in\ first\ half \\ key > A[m] & \rightarrow search\ recursively\ in\ second\ half \end{cases}$$

$$\begin{array}{c} K \\ \updownarrow \\ \underbrace{A[0] \ldots A[m-1]}_{\substack{\text{search here if} \\ K < A[m]}} \quad A[m] \quad \underbrace{A[m+1] \ldots A[n-1]}_{\substack{\text{search here if} \\ K > A[m]}}. \end{array}$$

Binary Search

- Searching in sorted array for a key
- Compare *key* with array's middle element $A[m]$

$$\begin{cases} \quad match \qquad \rightarrow stop, returm\ m \\ key < A[m] \rightarrow search\ recursively\ in\ first\ half \\ key > A[m] \rightarrow search\ recursively\ in\ second\ half \end{cases}$$

- Search for 70

index	0	1	2	3	4	5	6	7	8	9	10	11	12
value	3	14	27	31	39	42	55	70	74	81	85	93	98

Binary Search

- Searching in sorted array for a key
- Compare *key* with array's middle element $A[m]$

$$\begin{cases} \quad match & \rightarrow stop, returm\ m \\ key < A[m] & \rightarrow search\ recursively\ in\ first\ half \\ key > A[m] & \rightarrow search\ recursively\ in\ second\ half \end{cases}$$

- Search for 70

index	0	1	2	3	4	5	6	7	8	9	10	11	12
value	3	14	27	31	39	42	55	70	74	81	85	93	98
iteration 1	l						m						r
iteration 2								l		m			r
iteration 3								l,m	r				

Binary Search

ALGORITHM $BinarySearch(A[0..n-1], K)$

//Implements nonrecursive binary search
//Input: An array $A[0..n-1]$ sorted in ascending order and
// a search key K
//Output: An index of the array's element that is equal to K
// or -1 if there is no such element
$l \leftarrow 0$; $r \leftarrow n-1$
while $l \leq r$ **do**
  $m \leftarrow \lfloor (l+r)/2 \rfloor$
  if $K = A[m]$ **return** m
  else if $K < A[m]$ $r \leftarrow m-1$
  else $l \leftarrow m+1$
return -1

Binary Search

ALGORITHM *BinarySearch*$(A[0..n-1], K)$

```
//Implements nonrecursive binary search
//Input: An array A[0..n − 1] sorted in ascending order and
//          a search key K
//Output: An index of the array's element that is equal to K
//          or −1 if there is no such element
l ← 0;    r ← n − 1
while l ≤ r do
     m ← ⌊(l + r)/2⌋
     if K = A[m] return m
     else if K < A[m] r ← m − 1
     else l ← m + 1
return −1
```

Basic operation: 3-way compare

- $C_{best}(n) = 1$
- $C_{worst}(1) = 1$
- $C_{worst}(n) = C_{worst}\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 1$
- $C_{worst}(n) = \lfloor \log_2 n \rfloor + 1 = \lceil \log_2(n+1) \rceil$
- $C_{worst}(n) \in \theta(\log n)$

Fake-Coin Puzzle

There are n identically looking coins.

One of which is fake (lighter).
There is a balance scale that can tell whether two sides weigh the same and, if not, which of the two sides is heavier (but not by how much).

Design an efficient algorithm for detecting the fake coin.

Fake-Coin Puzzle

Decrease by factor 2 algorithm

Divide coins to two sets of $\left\lfloor \frac{n}{2} \right\rfloor$, leave one aside if n is odd

Compare $\begin{cases} equal \ \rightarrow the\ left\ aside\ one\ is\ fake \\ O.W. \rightarrow repeate\ with\ lighter\ set \end{cases}$

Fake-Coin Puzzle

Decrease by factor 2 algorithm

Divide coins to two sets of $\left\lfloor \frac{n}{2} \right\rfloor$, leave one aside if n is odd

Compare $\begin{cases} equal \ \rightarrow the\ left\ aside\ one\ is\ fake \\ \ O.W. \rightarrow repeate\ with\ lighter\ set \end{cases}$

$T(1) = 0$

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

$T(n) = \lfloor \log_2 n \rfloor$

$T(n) \in \theta(\log n)$

Fake-Coin Puzzle

Decrease by factor 3 algorithm

Divide coins to three sets of $\left\lfloor \frac{n}{3} \right\rfloor$, leave one or two aside if needed

Compare two sets $\begin{cases} equal \ \rightarrow repeate\ with\ third\ set + left\ aside\ ones \\ O.W. \rightarrow repeate\ with\ lighter\ set \end{cases}$

$T(1) = 0$

$T(n) = T\left(\left\lfloor \frac{n}{3} \right\rfloor\right) + 1$

$T(n) = \lfloor \log_3 n \rfloor$

$T(n) \in \theta(\log n)$

Russian Peasant Multiplication

Compute the product of two positive integers.

Russian Peasant Multiplication

Compute the product of two positive integers.

Can be solved by a decrease-by-half algorithm based on the following formulas.

$$
\begin{cases}
n.m = \frac{n}{2}.2m & n \text{ is even} \\
n.m = \frac{n-1}{2}.2m + m & n \text{ is odd} \\
1.m = m & n == 1
\end{cases}
$$

Russian Peasant Multiplication

Compute the product of two positive integers.

Can be solved by a decrease-by-half algorithm

n	m	addition
50	65	
Answer		

Russian Peasant Multiplication

Compute the product of two positive integers.

Can be solved by a decrease-by-half algorithm

n	m	addition
50	65	
25	130	
12	260	+130
6	520	
3	1040	
1	2080	+1040
Answer	2080+130+1040=3250	

Russian Peasant Multiplication

Compute the product of two positive integers.

Can be solved by a decrease-by-half algorithm

$$n.m = \frac{n}{2}.2m \qquad n \text{ is even}$$

$$n.m = \frac{n-1}{2}.2m + m \qquad n \text{ is odd}$$

$$1.m = m \qquad n == 1$$

$$T(1) = 0$$

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

$$T(n) = \lfloor\log_2 n\rfloor$$

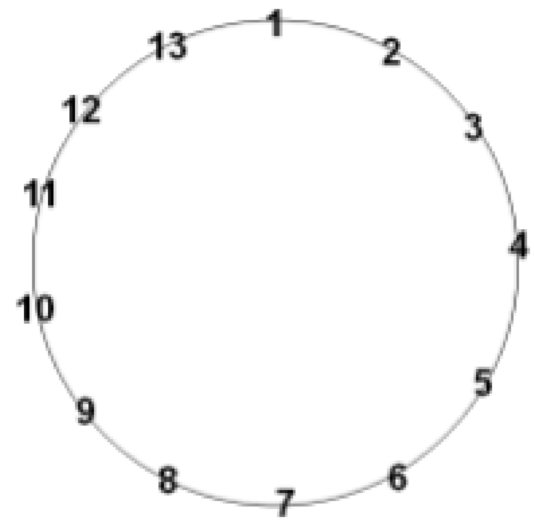
$$T(n) \in \theta(\log n)$$

Josephus Problem

- Jewish revolt against Roman
- 41 rebels took refuge in a cove (containing Josephus)
- Voted to perish rather than surrender
- Each man in turn dispatch his neighbor
- Order is determined by lot, Josephus got the last lot
- The last one surviving will surrender to Roman

Josephus Problem

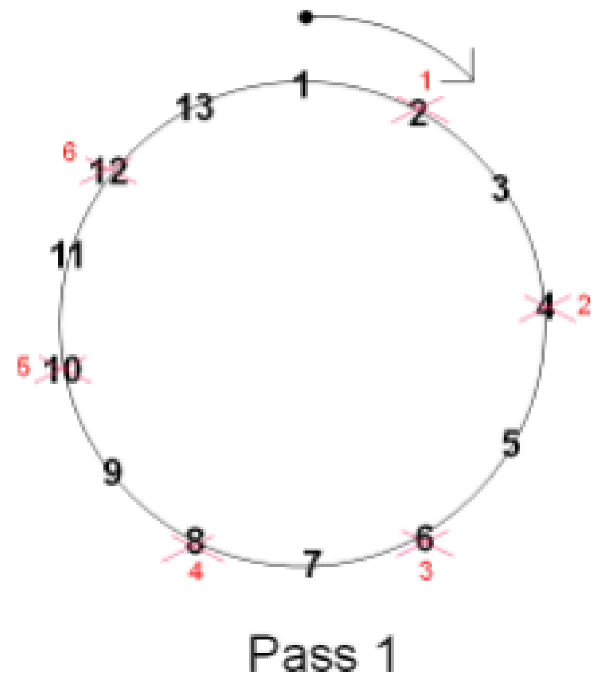
- n people numbered 1 to n
- Stand in circle
- Starting with person 1
- Eliminate every second person
- Until only one is left
- Determine the survivor's number $J(n)$



Initial

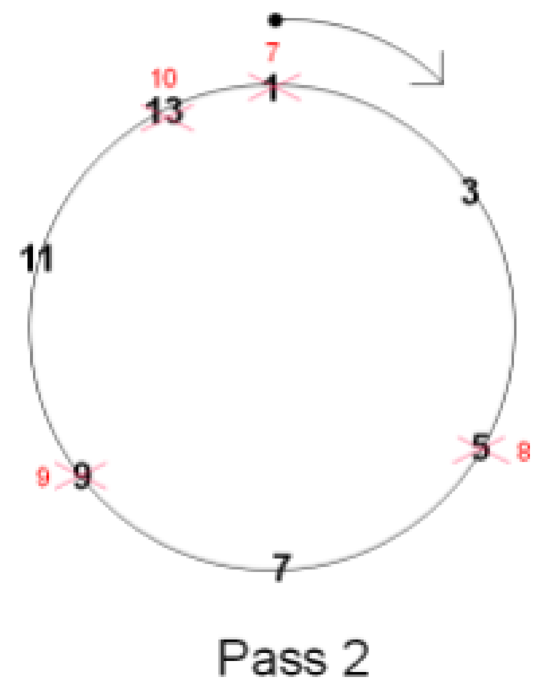
Josephus Problem

- n people numbered 1 to n
- Stand in circle
- Starting with person 1
- Eliminate every second person
- Until only one is left
- Determine the survivor's number $J(n)$



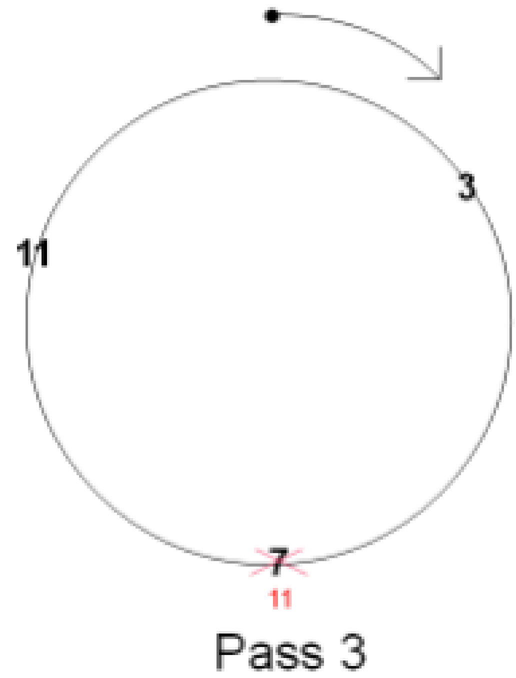
Josephus Problem

- n people numbered 1 to n
- Stand in circle
- Starting with person 1
- Eliminate every second person
- Until only one is left
- Determine the survivor's number $J(n)$



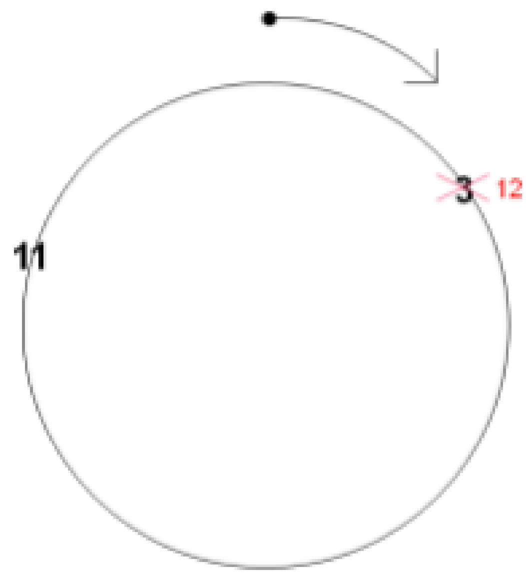
Josephus Problem

- n people numbered 1 to n
- Stand in circle
- Starting with person 1
- Eliminate every second person
- Until only one is left
- Determine the survivor's number $J(n)$



Josephus Problem

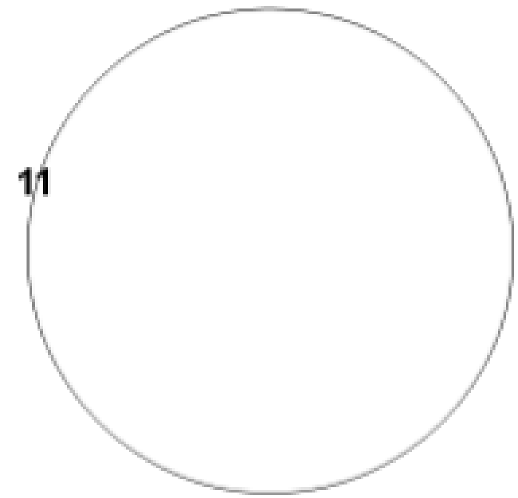
- n people numbered 1 to n
- Stand in circle
- Starting with person 1
- Eliminate every second person
- Until only one is left
- Determine the survivor's number $J(n)$



Pass 4

Josephus Problem

- n people numbered 1 to n
- Stand in circle
- Starting with person 1
- Eliminate every second person
- Until only one is left
- Determine the survivor's number $J(n)$



Final

Josephus Problem

- n people numbered 1 to n
- Stand in circle
- Starting with person 1
- Eliminate every second person
- Until only one is left
- Determine the survivor's number $J(n)$

- $$\begin{cases} n == 1 \rightarrow & J(1) = 1 \\ O.W. \quad \rightarrow & J(n) = 2J\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 1 \end{cases}$$

Josephus Problem

- n people numbered 1 to n
- Stand in circle
- Starting with person 1
- Eliminate every second person
- Until only one is left
- Determine the survivor's number $J(n)$

- Shifting binary representation of n to left by 1-bit cycle.
- $J(13) = J(1101_2) = 1011_2 = 11$
- $J(41) = J(101001_2) = 010011_2 = 19$