Recursion

Summary

- Definition of recursion and *divide et impera*
- Simple recursive algorithms
- Merge Sort
- Quicksort
- More complex recursive algorithms

2

Definition

Recursion

See "Recursion".

Recursion

If you still don't get it, see: "Recursion".

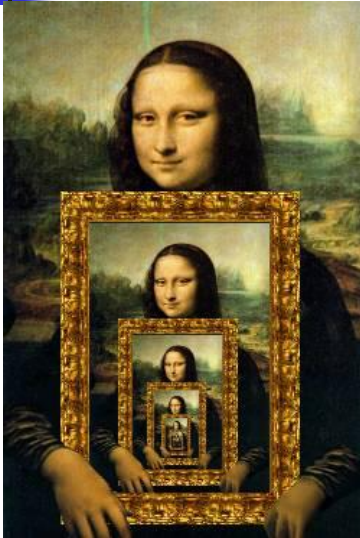
3

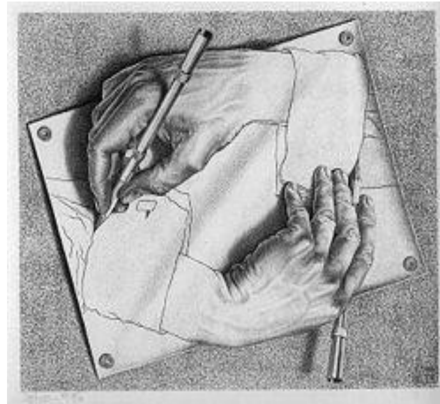
Definition

- A function F is called *recursive* when:
  1 Within the F function definition, there is a call to the same F function
  2 or there is a call to a function G which calls, directly or indirectly, the function F.
- An algorithm is called *recursive* when it is based on recursive functions.

4

Examples





5

Example: factorial

- $0! \equiv 1$
- $\forall N \geq 1$:
  $N! \equiv N \cdot (N-1)!$

```
double fact( double N )
{
  if(N == 0)
    return 1.0 ;
  return N * fact( N-1 );
}
```

6

Motivation

- Many problems have a recursive nature:
  - Define a method for solving sub-problems similar to the initial one (but smaller)
  - Define a method to combine partial solutions in the whole solution of the initial problem.

7

Divide and Conquer

"a" sub-problems, each one "b" times smaller than the problem

- Solution = Solve (problem) ;
- Solve (problem):
  - $\text{Subproblem}_{1,2,3,\ldots,a}$ = Divide(problem) ;
  - For each subproblem_i:
    - Sub-solution_i = Solve(subproblem_i) ;
  - Return solution = Combine($\text{sub-solution}_{1,2,3,\ldots,a}$) ;

8

Divide and Conquer

- Solution = Solve(problem) ;
- Solve (problem):
  - $Subproblem_{1,2,3,\ldots,a}$ = Divide(problem) ;
  - For each $subproblem_i$:
    - $Sub\text{-}solution_i$ = **Solve** ($subproblem_i$) ;
  - Return solution = Combine($sub\text{-}solution_{1,2,3,\ldots,a}$) ;

Recursive call

9

When stopping?

Recursion must not be infinite, as an algorithm must stop eventually…

At some point, sub-problems become so simple to be solvable because:

- Obvious solution, in case of sets of only one element
- Methods alternative to recursion.

10

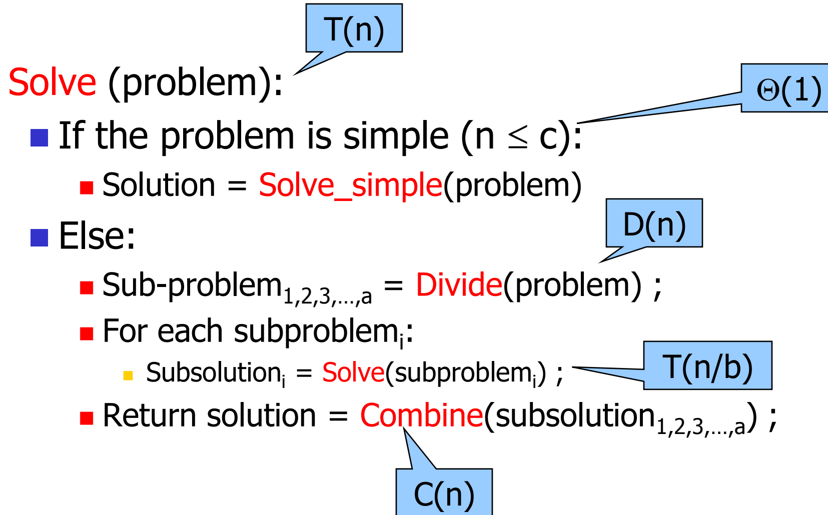
Note Well

- Remember to always set a stop condition
- Try to make sub-problems dimensions less than the initial problem

Divide and Conquer (with stop condition)

Solve(problem):

- If the problem is simple:
  - Solution = Solve_simple(problem)
- Else:
  - Sub-problem$_{1,2,3,\ldots,a}$ = Divide(problem) ;
  - Fro each sub-problem$_i$:
    - Sub-solution$_i$ = Solve(subproblem$_i$) ;
  - Return solution = Combine (subsolution$_{1,2,3,\ldots,a}$);



Complexity (II)

$T(n) =$

- $\Theta(1)$ for $n \leq c$
- $D(n) + a\,T(n/b) + C(n)$ for $n > c$

Recursive Equation not easy to solve.

If $D(n)+C(n)=\Theta(n)$, then $T(n)=\Theta(n \log n)$.

14

Summary

- Definition of recursion and *divide et impera*
- Simple recursive algorithms
- Merge Sort
- Quicksort
- More complex recursive algorithms

15

Example: Fibonacci numbers

Problem:

- Calculate the N-th number of Fibonacci sequence

Definition:

- $FIB_{N+1} = FIB_N + FIB_{N-1}$ for $n>0$
- $FIB_1 = 1$
- $FIB_0 = 0$

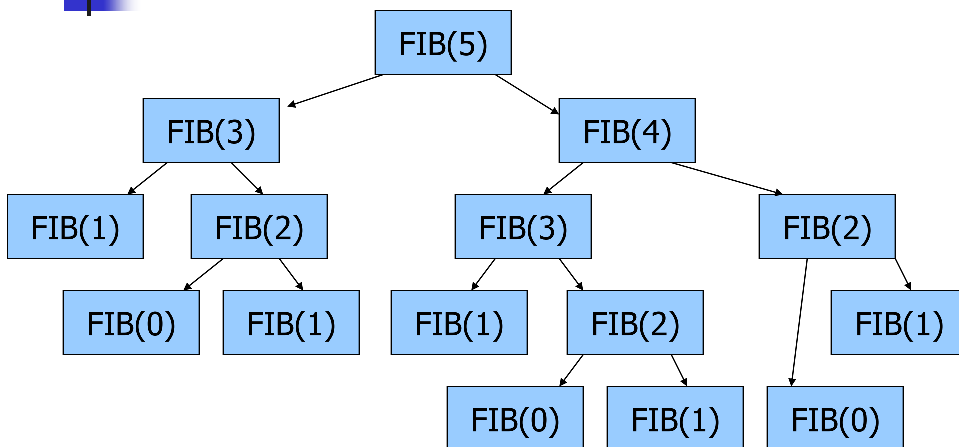
16

Solution

```
long fib(int N)
{
        long f1, f2 ;

        if(N == 0) return 0 ;
        if(N == 1) return 1 ;

        f1 = fib(N-1) ;
        f2 = fib(N-2) ;
        return f1+f2 ;
}
```

17

Analysis



18

Example: binary search

Problem

- Define if an element x is included in a sorted array v[N]

Approach

- Divide the array in 2 half parts and re-apply the problem on one half (the other half can be excluded as the array is sorted)

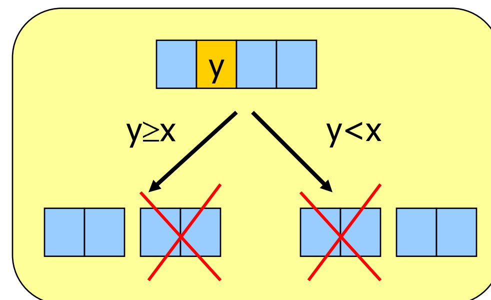
Example

v 1 3 4 6 8 9 11 12 x 4

1 3 4 6 8 9 11 12

1 3 4 6

4 6



Solution

```
int search(int v[], int a, int b, int x)
{
     int c ;
     if(b-a == 0)
          if (v[a]==x) return a ;
          else return -1 ;

     c = (a+b) / 2 ;
     if(v[c] >= x)
          return search(v, a, c, x) ;
     else return search(v, c+1, b, x) ;
}
```

21

Exercise

Calculate binomial coefficient (n m), taking into account information derived from Tartaglia triangle:

$$\begin{cases} \binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m} \\ \binom{n}{n} = \binom{n}{0} = 1 \\ 0 \le n, \quad 0 \le m \le n \end{cases}$$

22

Exercise

Calculate the determinant of a square matrix.

Remember that:

- $Det(M_{1x1}) = m_{11}$
- $Det(M_{NxN})$ = sum of products of elements of a row (or column) multiplied the determinants of sub-matrixes (N-1)x(N-1) obtained by deleting the row and column containing the element, taken with sign $(-1)^{i+j}$.

Recursion and iteration

AN Information Theory theorem proves that each recursive program can be implemented with an iterative program.

Best solution may depend on efficiency and clarity of code and it depends by the kind of problem.

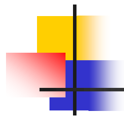
Example: factorial (iterative)

- $0! \equiv 1$
- $\forall N \geq 1$:
  $N! \equiv N \cdot (N-1)!$

```
double fact( double N )
{
    double tot = 1.0 ;
    int i;
    for(i=2; i<=N; ++i)
        tot = tot * i ;
    return tot ;
}
```


Fibonacci (iterative)

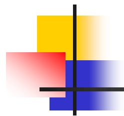
```
long fib(int N)
{    long f1=1, f2=0, f ;
     int i;
     if(N == 0) return 0 ;
     if(N == 1) return 1 ;

     f = f1 + f2 ; /* N==2 */
     for(i=3; i<= N; ++i)
     {    f2 = f1 ;
          f1 = f ;
          f = f1+f2 ;
     }
     return f ;
}
```



Binary Search (iterative)

```
int search(int v[], int a, int b, int x)
{
     int c ;
     while(b-a != 0)
     {
          c = (a+b) / 2 ;
          if(v[c] >= x)
               b = c ;
          else a = c+1 ;
     }
     if(v[a]==x) return a ;
     else return -1 ;
}
```

27



Proposed Excercises

1. Provide the iterative version of binomial coefficient (n m).

2. Analyze the difficulties in realizing the iterative version of determinant calculation.

28

Sorting Algorithms
(Recursive Algorithms)

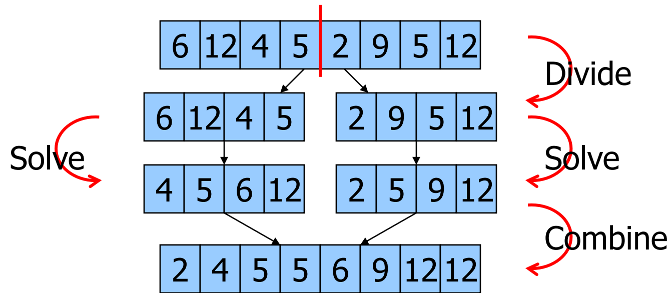
Summary

- Definition of recursion and *divide et impera*
- Simple recursive algorithms
- Merge Sort
- Quicksort
- More complex recursive algorithms

30

Merge Sort

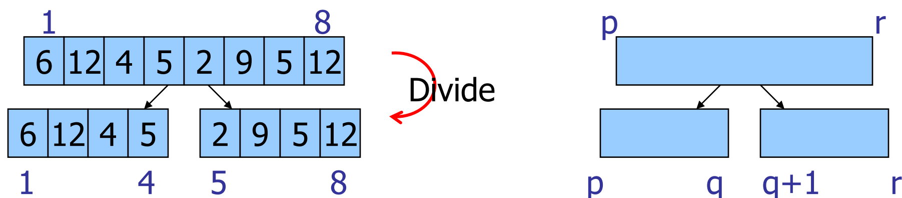
Merge Sort algorithm is the direct application of "Divide and Conquer" to sorting problems.



31

Merge Sort: Divide

"Divide" step simply consists in partitioning the initial array in 2 sub-arrays, before and after the division point, which is usually chosen in the middle of the array.



32

Merge Sort: stop condition

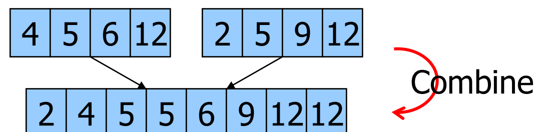
Stop condition holds whenever the sub-array has only one element (p=r) or no elements (p>r).

Merge Sort: Combine

Combine step is based on *merge* of two ordered vectors:

- Given 2 ordered arrays, merge them together in a third sorted array

This algorithm complexity is $\Theta(n)$.



Pseudo-code

```
MERGE-SORT(A, p, r)
1   if p < r                                   } Stop condition
2       then q ← ⌊(p+r)/2⌋                     } Divide
3            MERGE-SORT(A, p, q)               } Solve
4            MERGE-SORT(A, q+1, r)             }
5            MERGE(A, p, q, r)                 } Combine
```

35

Notation

These symbols are used in math and computer science:

- $\lfloor x \rfloor$ = the biggest integer <= x (read: "floor of x")
  - i.e. integer part of x
- $\lceil x \rceil$ = the smallest integer >= x (read: "ceiling of x")

Examples:

- $\lfloor 3 \rfloor = \lceil 3 \rceil = 3$
- $\lfloor 3{,}1 \rfloor = 3$; $\lceil 3{,}1 \rceil = 4$

36

Merge pseudo-code

```
MERGE(A, p, q, r)
1   i ← p ; j ← q+1 ; k ← 1
2   while( i ≤ q and j ≤ r )
3       if( A[i] < A[j])  B[k] ← A[i] ; i ← i+1
4       else              B[k] ← A[j] ; j ← j+1
5       k ← k+1
6   while( i≤q )    B[k]←A[i] ; i←i+1; k←k+1
7   while( j≤r )    B[k]←A[j] ; j←j+1; k←k+1
8   A[p..r] ← B[1..k-1]
```

37

Merge Procedure

```
MERGE(A, p, q, r)
1   i ← p ; j ← q+1 ; k ← 1
2   while( i ≤ q and j ≤ r )
3       if( A[i] < A[j])  B[k] ← A[i] ; i ← i+1
4       else              B[k] ← A[j] ; j ← j+1
5       k ← k+1
6   while( i≤q )    B[k]←A[i] ; i←i+1; k←k+1
7   while( j≤r )    B[k]←A[j] ; j←j+1; k←k+1
8   A[p..r] ← B[1..k-1]
```

Take each time the smallest, between the first two elements not considered yet

Process the “tail” of the remaining vector

Complexity (I)

Merge Sort complexity analysis leads to the following formulas:

- Stop condition: simple test, $\Theta(1)$
- Divide (2): split array in 2 parts, $D(n)=\Theta(1)$
- Solve (3-4): solve 2 sub-problems, one of dimension $\lfloor n/2 \rfloor$, the other $\lceil n/2 \rceil$, so the sum is $2T(n/2)$
- Combine (5): based on Merge, $C(n) = \Theta(n)$.

39

Complexity (II)

$T(n) =$

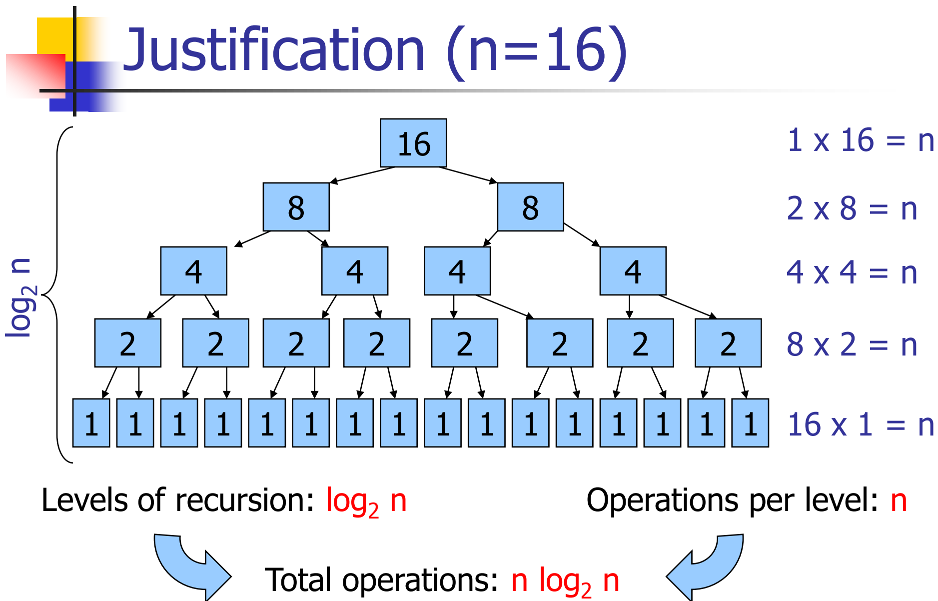
- $\Theta(1)$ for $n \le 1$
- $2T(n/2) + \Theta(n)$ for $n > 1$

There is a proof showing that the solution is:

- $T(n) = \Theta(n \log n)$

40

Justification (n=16)



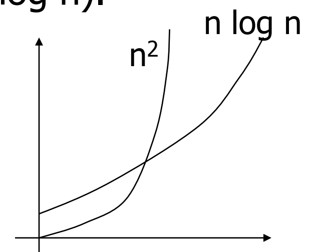
Note Well

Not all recursive procedures have complexity $\Theta(n \log n)$.

For example, merge sort with an asymmetric partition (q=p+1), can become an insertion sort, with *n* recursive calls, and in each one an element is added to the already ordered set, so the complexity in this case is $\Theta(n^2)$.

Exercise

Implement Merge Sort algorithm in C.
Verify that its behavior is Θ(n log n).



43

Summary

- Definition of recursion and *divide et impera*
- Simple recursive algorithms
- Merge Sort
- Quicksort
- More complex recursive algorithms

44

Description

Quicksort is a Divide et Impera algorithm where partitioning is based on *value* (and not on position) of array elements.

At each step, a "pivot" value is chosen, and elements are assigned to one of the two partitions depending on the fact they are less than or more than the pivot value.

45

Solution Steps

- Divide: partitioning A[p..r], reordering elements, in two sub-vectors A[p..q] and A[q+1..r], so that the elements of A[p..q] are all ≤ the elements of A[q+1..r]. The value of q is variable.
- Solve: recursively order A[p..q] and A[q+1..r].
- Combine: as A[p..q] and A[q+1..r] are ordered, and first ones are ≤ second ones, do nothing: A[p..r] is sorted.

46

Quicksort

```
QUICKSORT(A, p, r)
1   if p < r
2       then    q ← PARTITION(A, p, r)
3               QUICKSORT(A, p, q)
4               QUICKSORT(A, q+1, r)
```

Partition

```
PARTITION(A, p, r)
1   x ← A[p]        ▷ Pivot element
2   i ← p − 1
3   j ← r + 1
4   while true
5       do  repeat  j ← j-1
6               until A[j] ≤ x
7           repeat i ← i+1
8               until A[i] ≥ x
9           if i < j
10              then swap  A[i] ↔ A[j]
11              else return j
```

Example

p r

A	6	12	4	5	2	9	5	12

i j

6 x

A	6	12	4	5	2	9	5	12

i j

9 10 11

A	5	12	4	5	2	9	6	12

i j

A	5	12	4	5	2	9	6	12

i j

A	5	2	4	5	12	9	6	12

i j

```
do  repeat j ← j-1
        until A[j] ≤ x
    repeat i ← i+1
        until A[i] ≥ x
```

```
if i < j
    then swap    A[i] ↔ A[j]
    else return j
```

```
do  repeat j ← j-1
        until A[j] ≤ x
    repeat i ← i+1
        until A[i] ≥ x
```

p q q+1 r

5	2	4	5		12	9	6	12

49

Analysis of Partition

This procedure makes indexes i and j to converge into the center of the array.

Once i finds an element more than the pivot and j finds an element less than the pivot, the elements are swapped (exchanged) to allow indexes to continue.

When i "meets" j, partition is complete.

The number of operations is Θ(r-p+1), even if there are nested loops.

50

Exercise 1

Show how Partition works on this sequence of numbers:

A = { 8, 13, 11, 19, 12, 9, 5, 7, 4, 2, 6, 1 }

51

Solution 1

A = { 8, 13, 11, 19, 12, 9, 5, 7, 4, 2, 6, 1 }
x = 8
A = { 8, 13, 11, 19, 12, 9, 5, 7, 4, 2, 6, 1 }
A = { 1, 13, 11, 19, 12, 9, 5, 7, 4, 2, 6, 8 }
A = { 1, 6, 11, 19, 12, 9, 5, 7, 4, 2, 13, 8 }
A = { 1, 6, 2, 19, 12, 9, 5, 7, 4, 11, 13, 8 }
A = { 1, 6, 2, 4, 12, 9, 5, 7, 19, 11, 13, 8 }
A = { 1, 6, 2, 4, 7, 9, 5, 12, 19, 11, 13, 8 }
A = { 1, 6, 2, 4, 7, 5, 9, 12, 19, 11, 13, 8 }
Partition:
A = { 1, 6, 2, 4, 7, 5, 9, 12, 19, 11, 13, 8 }

52

Exercise

Implement in C the Partition procedure, verifying its behavior.

Which value is returned as partition point q on random arrays? An what happens on sorted arrays? And on arrays sorted in inverse order?

Exercise

In the same array:

A = { 13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 1 }

Which would be a better choice for pivot x?

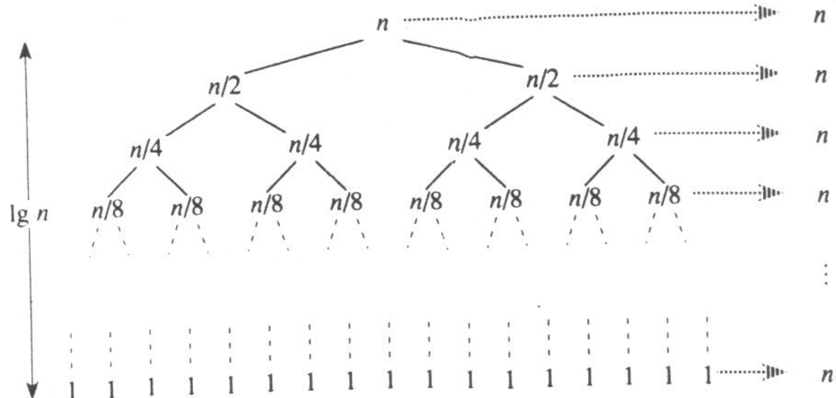
Performance

The efficiency of Quicksort totally depends on the quality of partitioning.

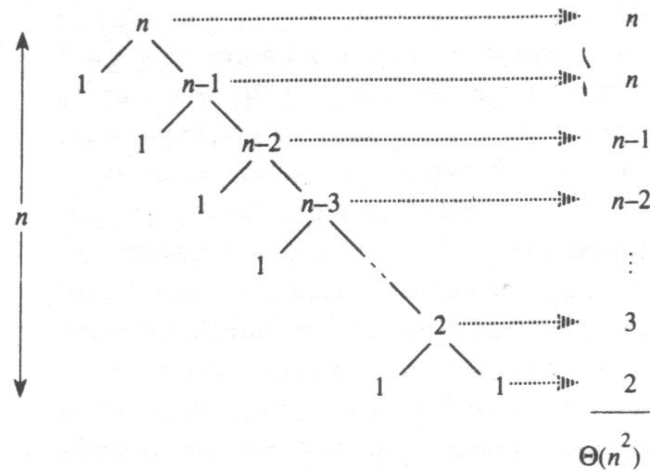
- If partitioning is optimal (q ~ (p+r)/2), the algorithm is O(n log n)
- If partitioning is the worst case (q~p or q~r), lthe algorithm becomes $O(n^2)$

Partitioning quality depends on the good choice of the pivot value.

Best case



Worst Case



57

Pivot choice

Worst case happens when one partition has n-1 elements and the other just one, i.e. when pivot is the maximum or minimum.

An already sorted array falls into the worst case! Same case for an array with inverse ordering.

Best case is when the array is as most random as possible.

58

Quicksort randomized

PARTITION-RANDOM(A, p, r)
  ▷ generate a random number between p and r
1 $i \leftarrow$ RANDOM(p, r)
  ▷ avoid worst case!
2 swaps $A[i] \leftrightarrow A[1]$
  ▷ apply partitioning
3 **return** PARTITION (A, p, r)

59

Choosing the pivot

- Choose a random value (PARTITION-RANDOM)
- Choose the element in the middle:
  - x ← A[(p+r)/2]
- Choose the mean value between min and max
- Choose the mean value among 3 random elements in the array
- ...

60

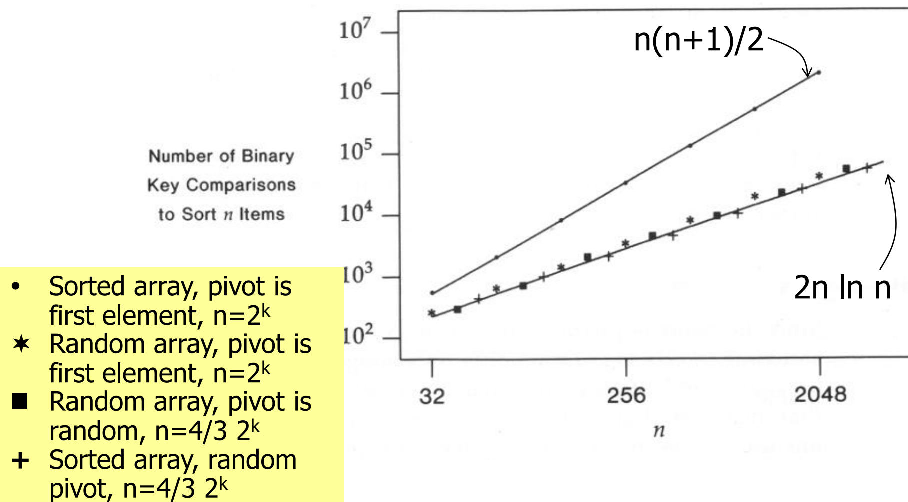
Complexity

So far, Quicksort behavior is:

- $O(n^2)$ in general, $O(n \log n)$ in average
- $\Theta(n^2)$ in worst case (which must be avoided)
- $\Theta(n \log n)$ in average and best cases

61

Examples



62

Exercise

Provide an implementation in C language of Quicksort algorithm, and try different pivot choices.

63

Summary

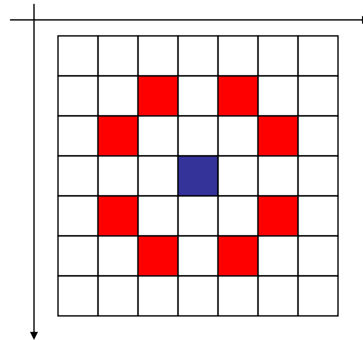
- Definition of recursion and *divide et impera*
- Simple recursive algorithms
- Merge Sort
- Quicksort
- More complex recursive algorithms

64

The Knight Tour

Find a sequence of moves for the knight on a chessboard (N x N dimensions), in order to make the knight passing on each square ONLY ONCE.

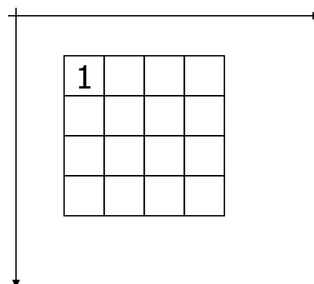
Remember that a knight can move in 8 ways at most.



65

Analysis

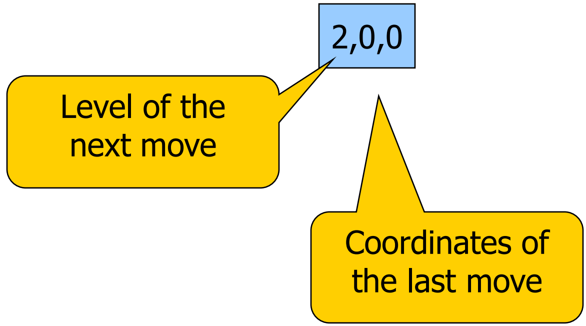
Suppose N=4.



66

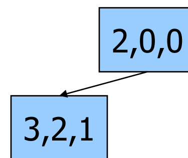
Move 1

1			



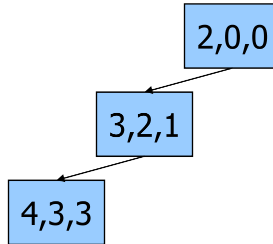
Move 2

1			
	2		



Move 3

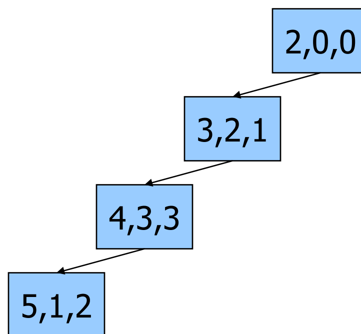
1			
	2		
			3



69

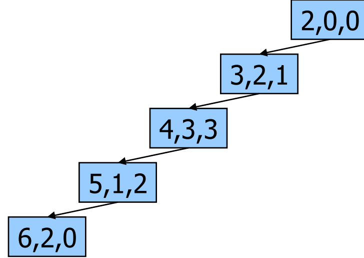
Move 4

1			
		4	
	2		
			3



70

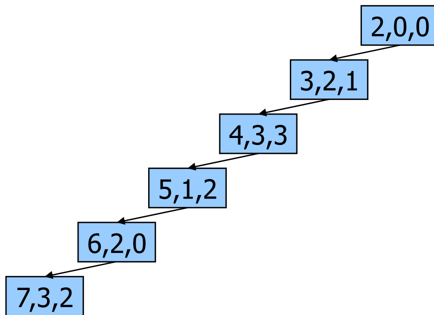
Move 5



1			
		4	
5	2		
			3

71

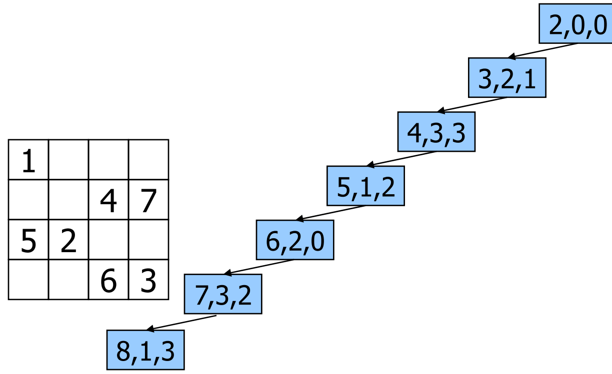
Move 6



1			
		4	
5	2		
		6	3

72

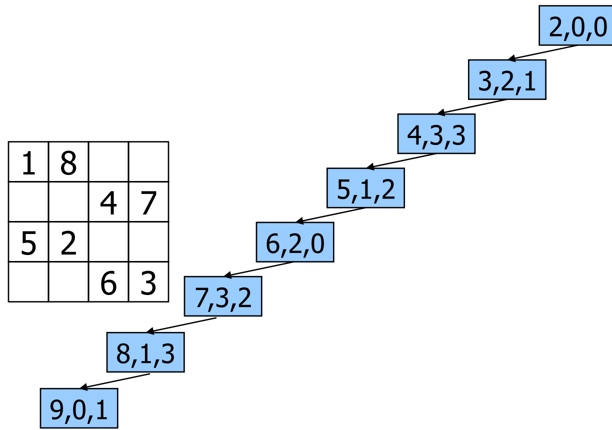
Move 7



1			
		4	7
5	2		
		6	3

73

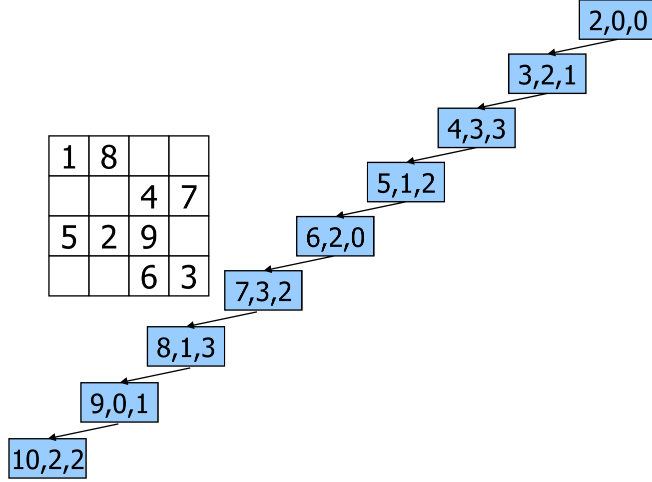
Move 8



1	8		
		4	7
5	2		
		6	3

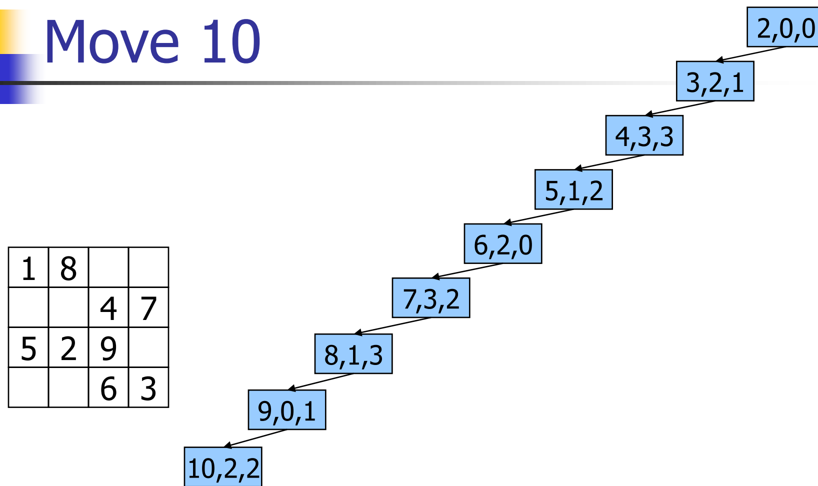
74

Move 9

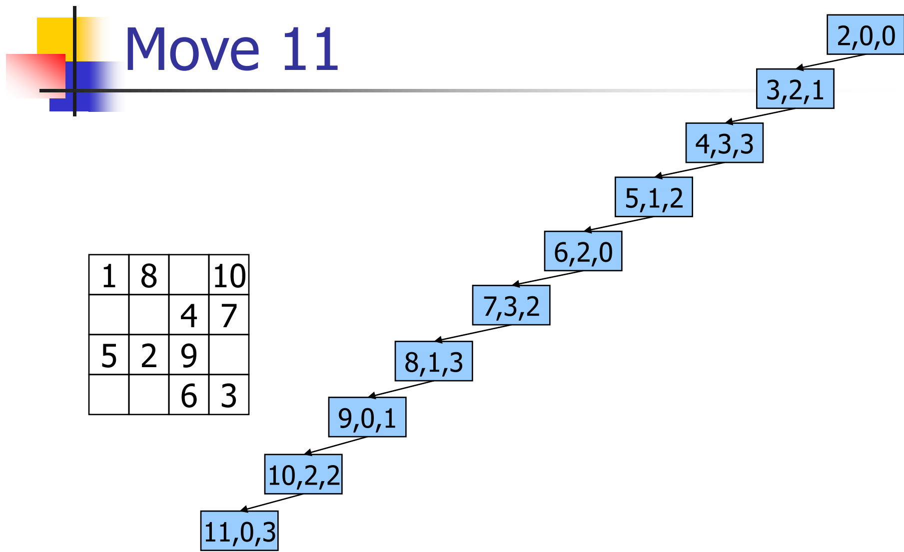


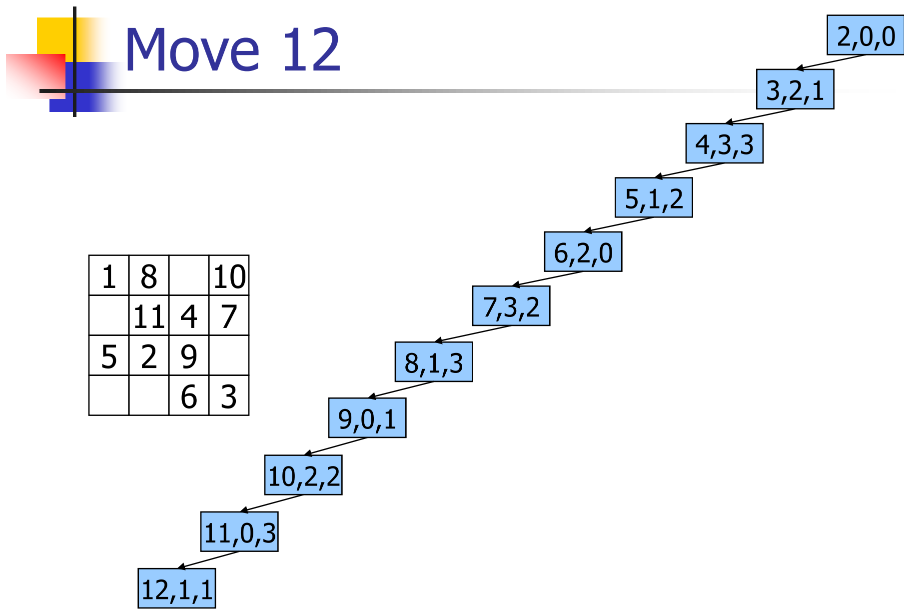
75

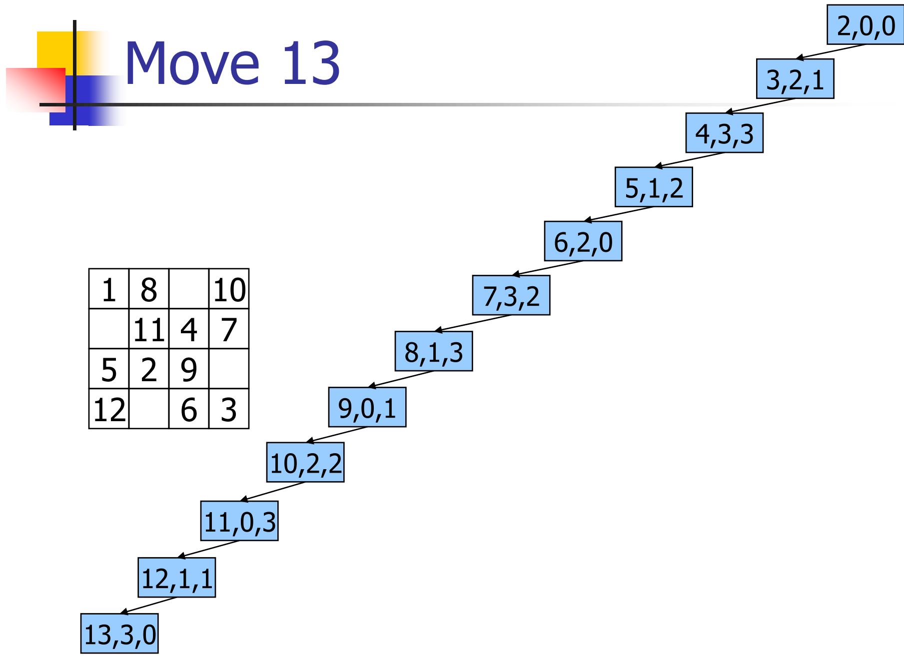
Move 10

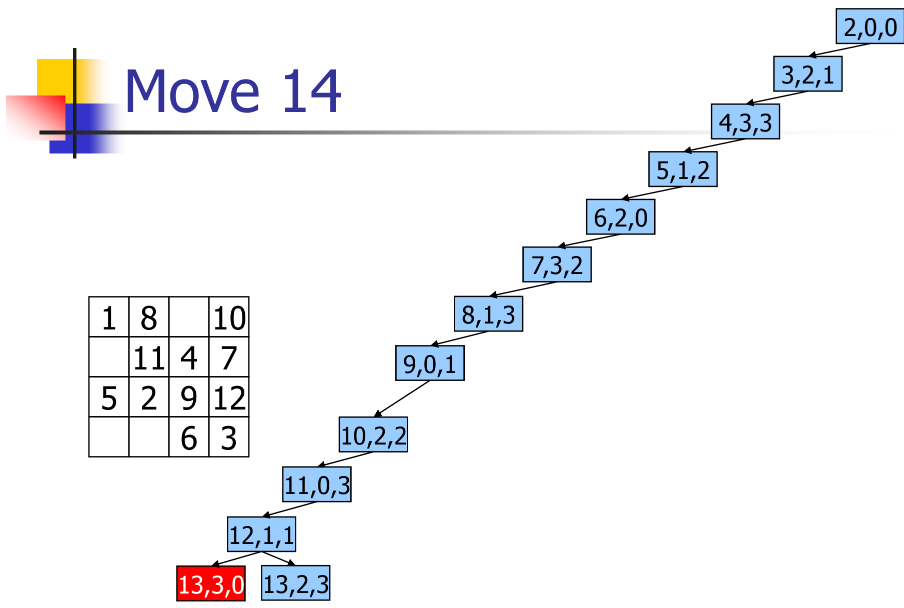


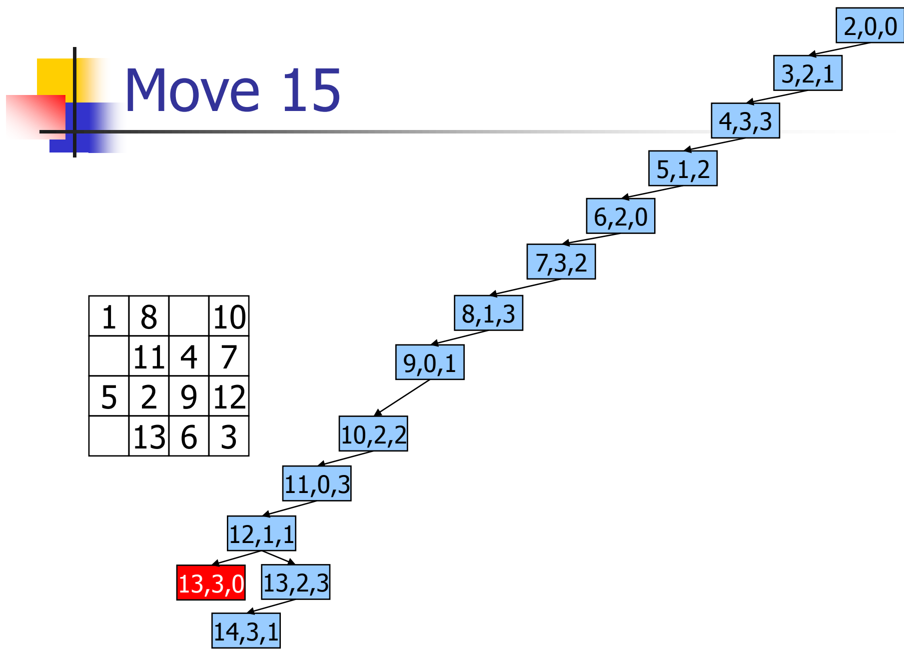
76


Move 11
1 8 10
4 7
5 2 9
6 3
2,0,0
3,2,1
4,3,3
5,1,2
6,2,0
7,3,2
8,1,3
9,0,1
10,2,2
11,0,3
77

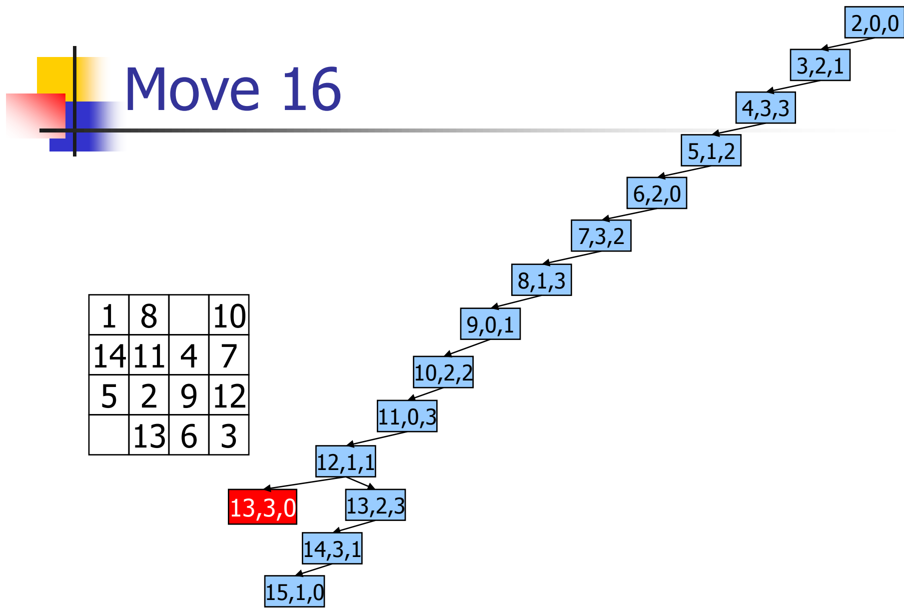

Move 12
1 8 10
11 4 7
5 2 9
6 3
2,0,0
3,2,1
4,3,3
5,1,2
6,2,0
7,3,2
8,1,3
9,0,1
10,2,2
11,0,3
12,1,1
78


Move 13
2,0,0
3,2,1
4,3,3
5,1,2
6,2,0
7,3,2
8,1,3
9,0,1
10,2,2
11,0,3
12,1,1
13,3,0
1 8 10
11 4 7
5 2 9
12 6 3
79

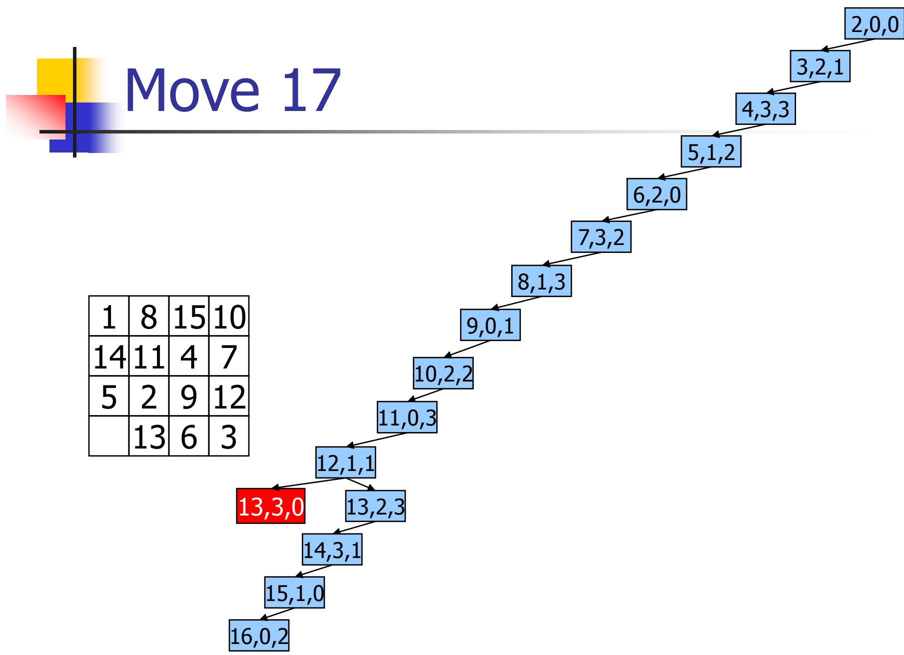

Move 14
2,0,0
3,2,1
4,3,3
5,1,2
6,2,0
7,3,2
8,1,3
9,0,1
10,2,2
11,0,3
12,1,1
13,3,0
13,2,3
1 8 10
11 4 7
5 2 9 12
6 3
80

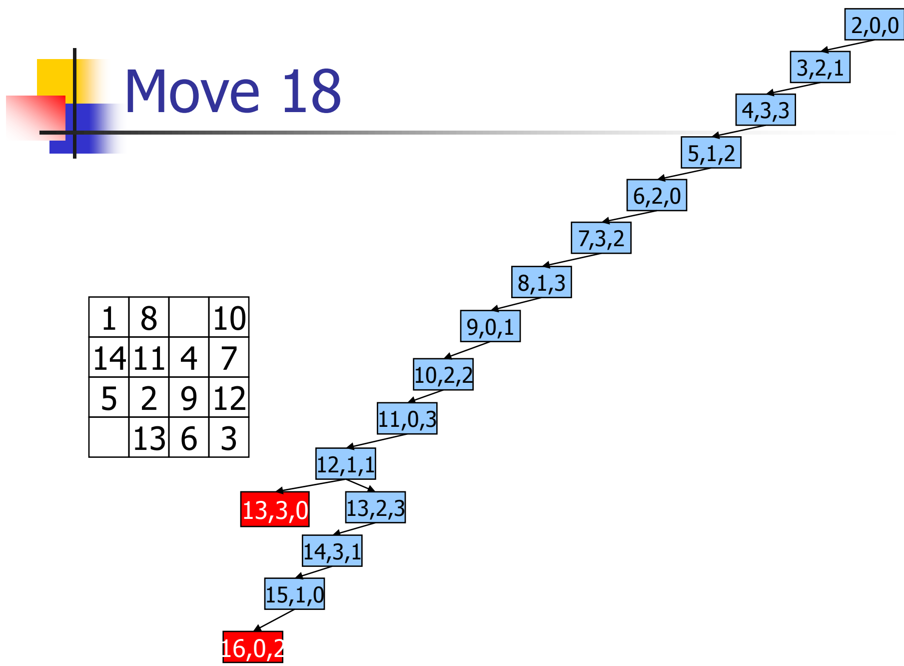


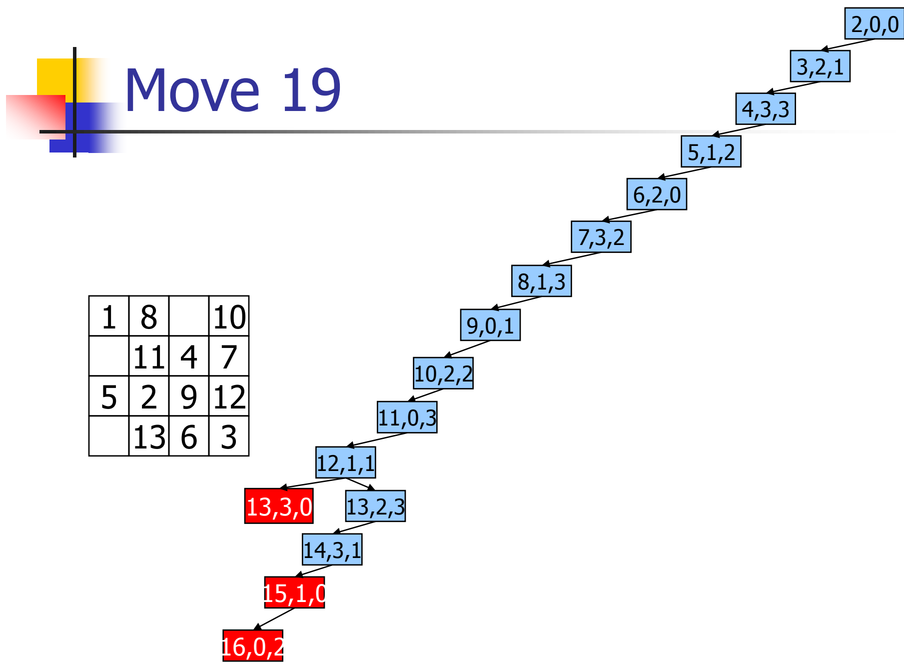
81



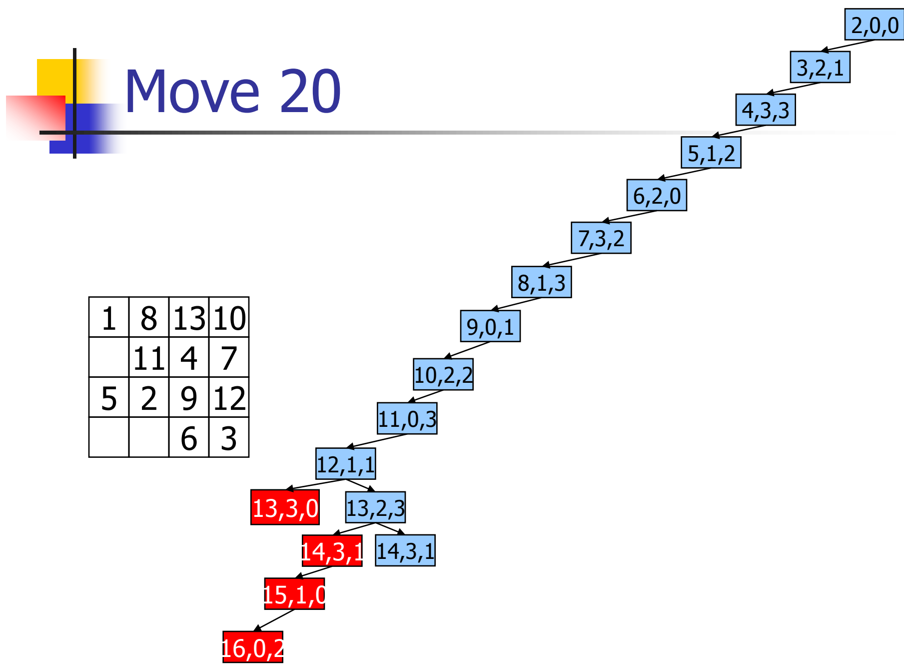
82


Move 17
2,0,0
3,2,1
4,3,3
5,1,2
6,2,0
7,3,2
8,1,3
9,0,1
10,2,2
11,0,3
12,1,1
13,3,0
13,2,3
14,3,1
15,1,0
16,0,2
1 8 15 10
14 11 4 7
5 2 9 12
13 6 3
83

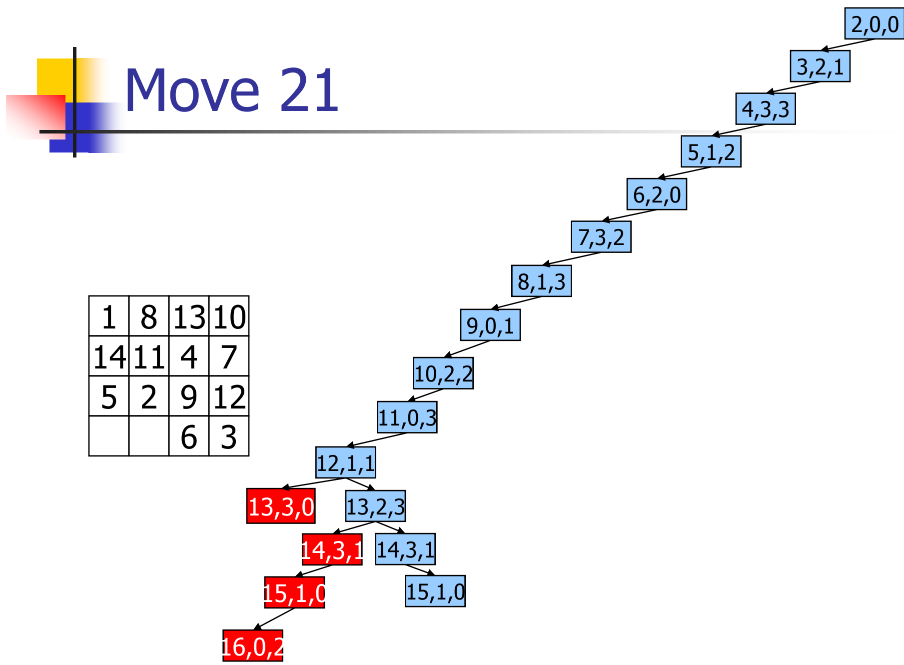

Move 18
2,0,0
3,2,1
4,3,3
5,1,2
6,2,0
7,3,2
8,1,3
9,0,1
10,2,2
11,0,3
12,1,1
13,3,0
13,2,3
14,3,1
15,1,0
16,0,2
1 8 10
14 11 4 7
5 2 9 12
13 6 3
84

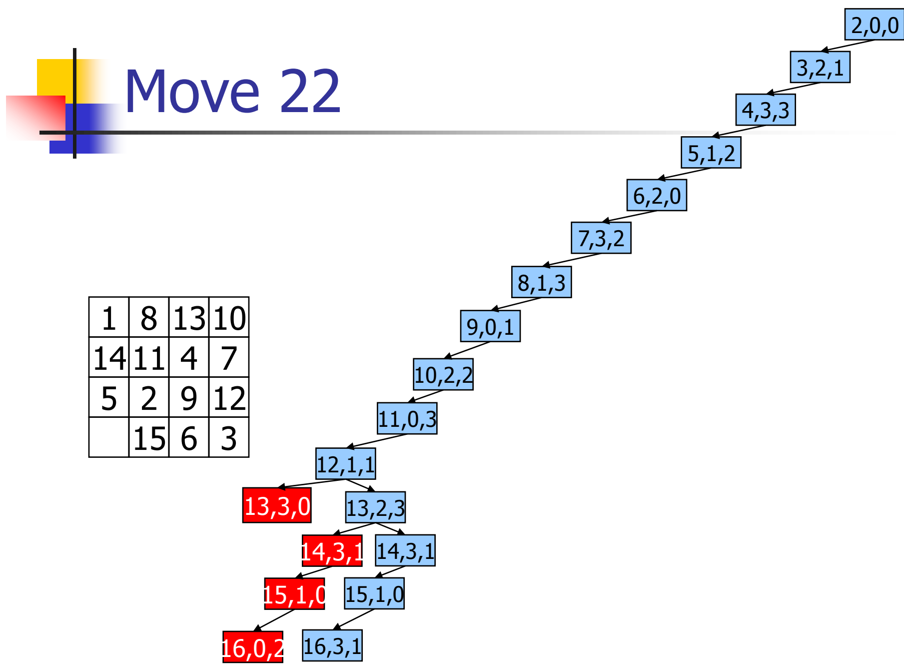


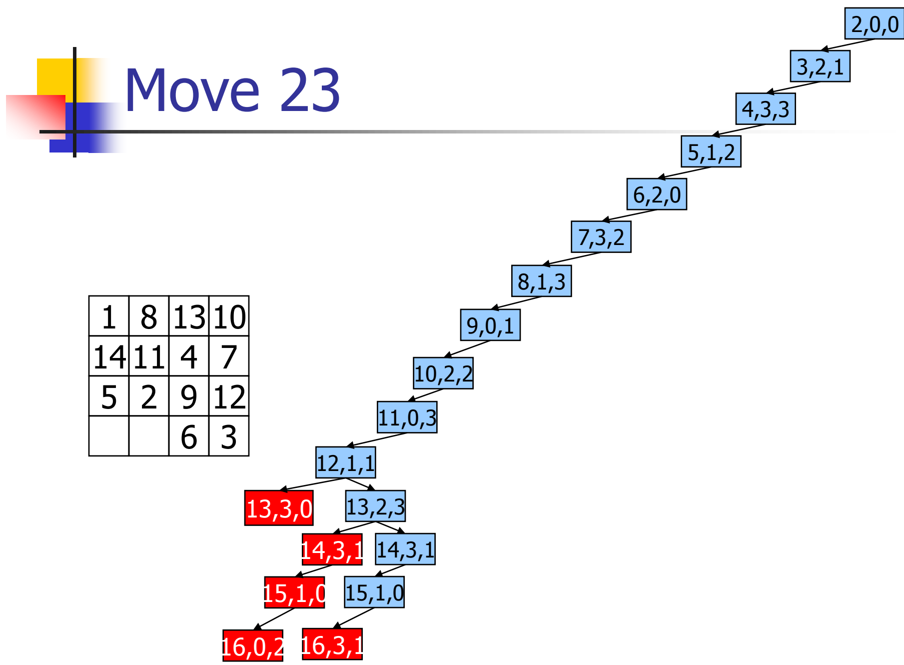
85



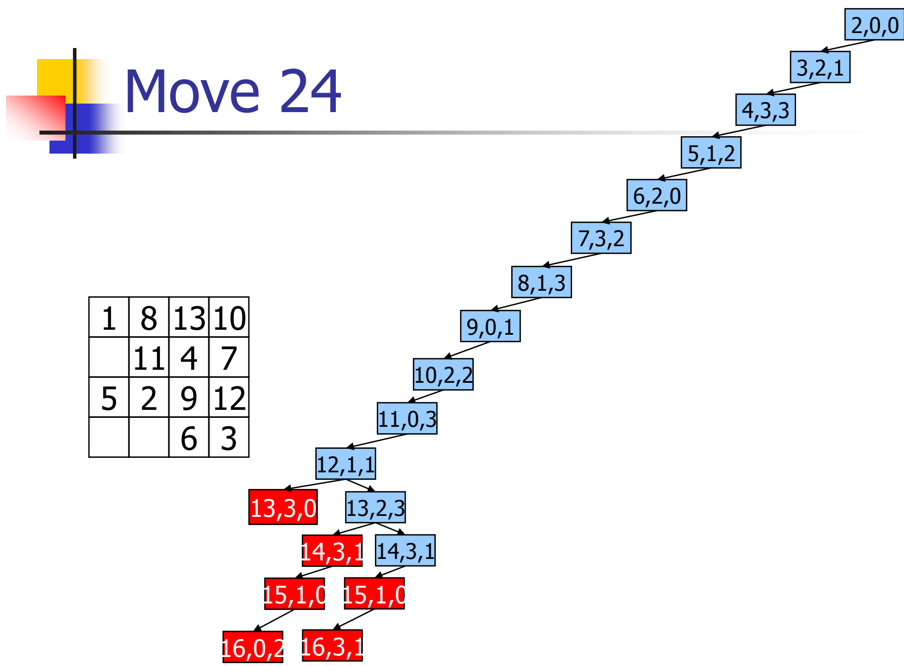
86


Move 21
1 8 13 10
14 11 4 7
5 2 9 12
6 3
2,0,0
3,2,1
4,3,3
5,1,2
6,2,0
7,3,2
8,1,3
9,0,1
10,2,2
11,0,3
12,1,1
13,3,0
13,2,3
14,3,1
14,3,1
15,1,0
15,1,0
16,0,2
87

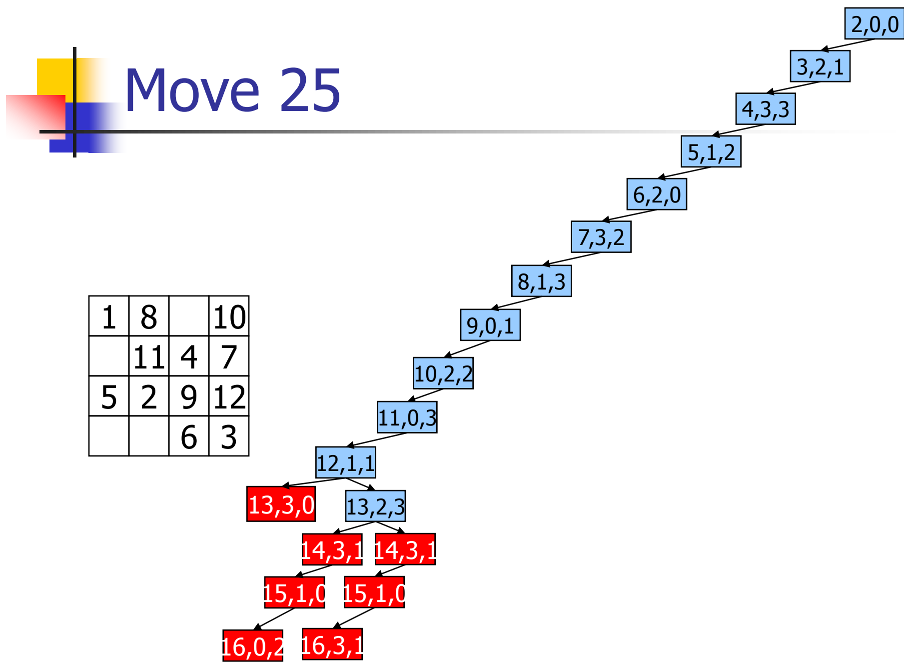

Move 22
1 8 13 10
14 11 4 7
5 2 9 12
15 6 3
2,0,0
3,2,1
4,3,3
5,1,2
6,2,0
7,3,2
8,1,3
9,0,1
10,2,2
11,0,3
12,1,1
13,3,0
13,2,3
14,3,1
14,3,1
15,1,0
15,1,0
16,0,2
16,3,1
88



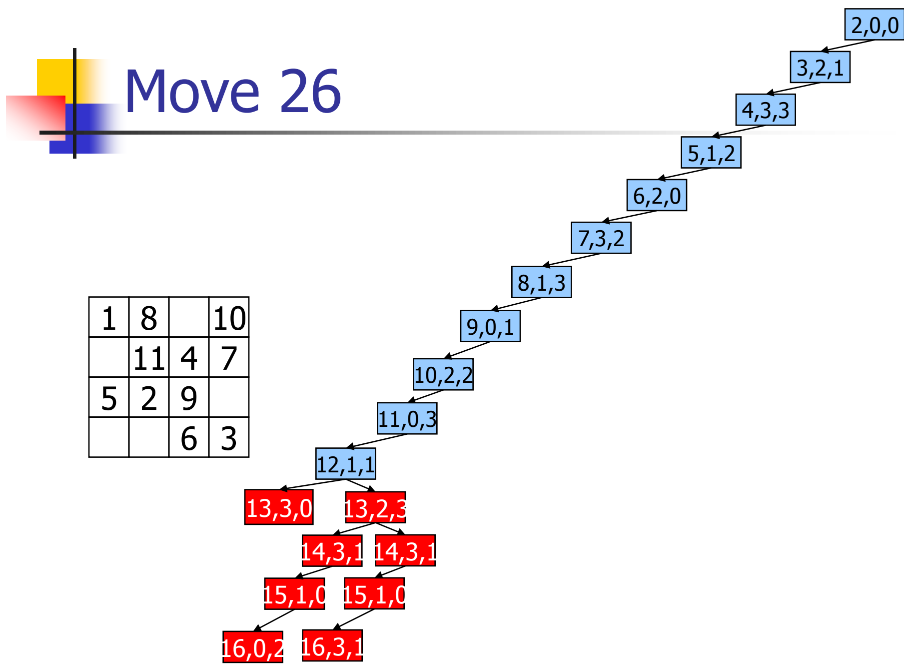
89



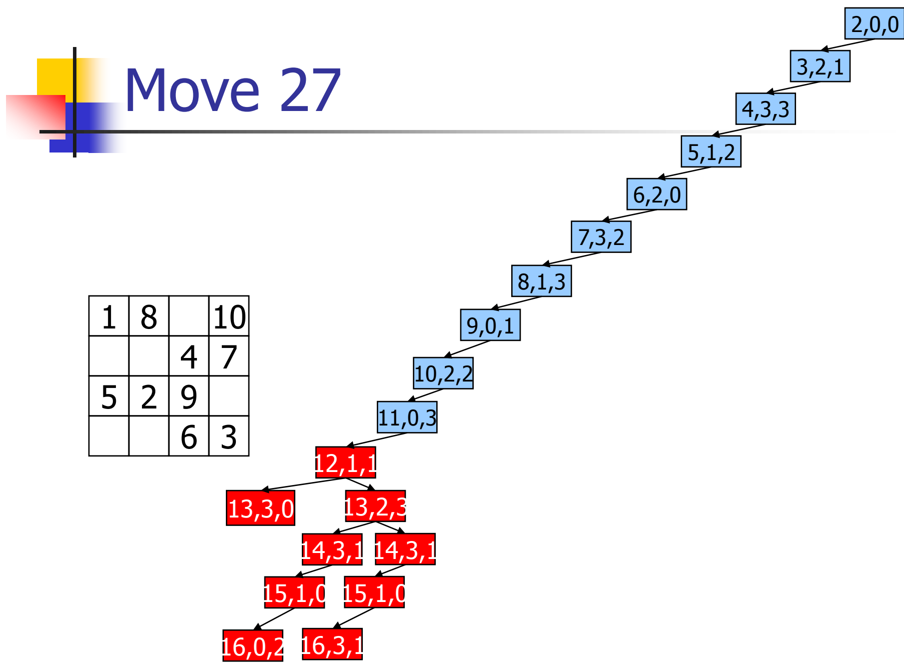
90

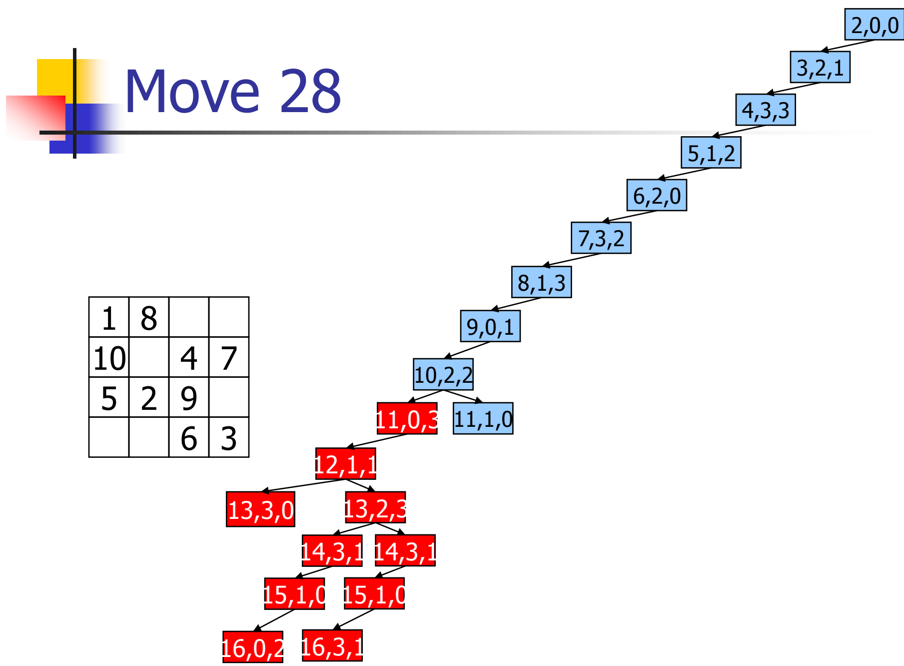


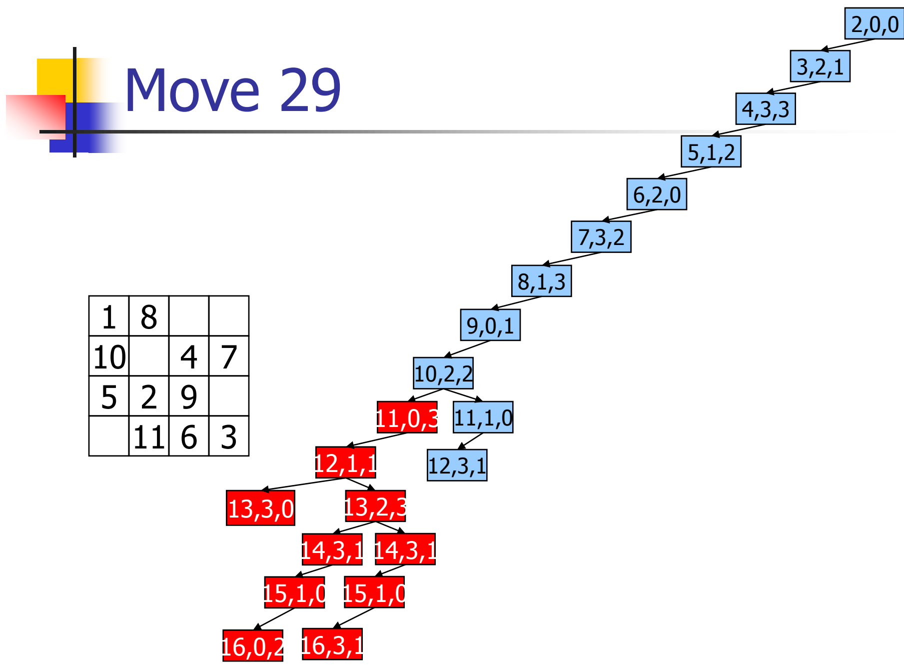
91

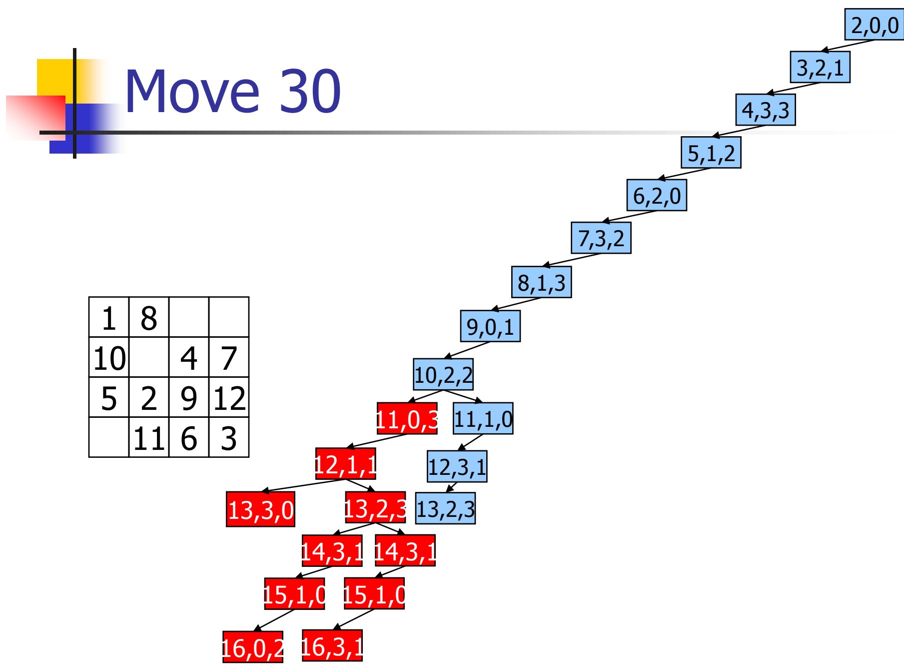


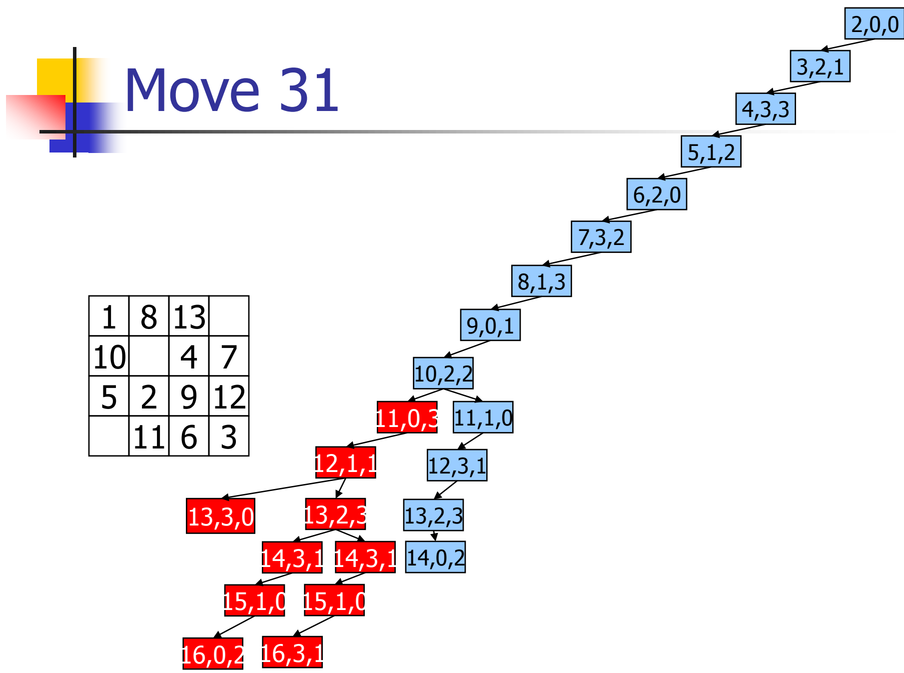
92

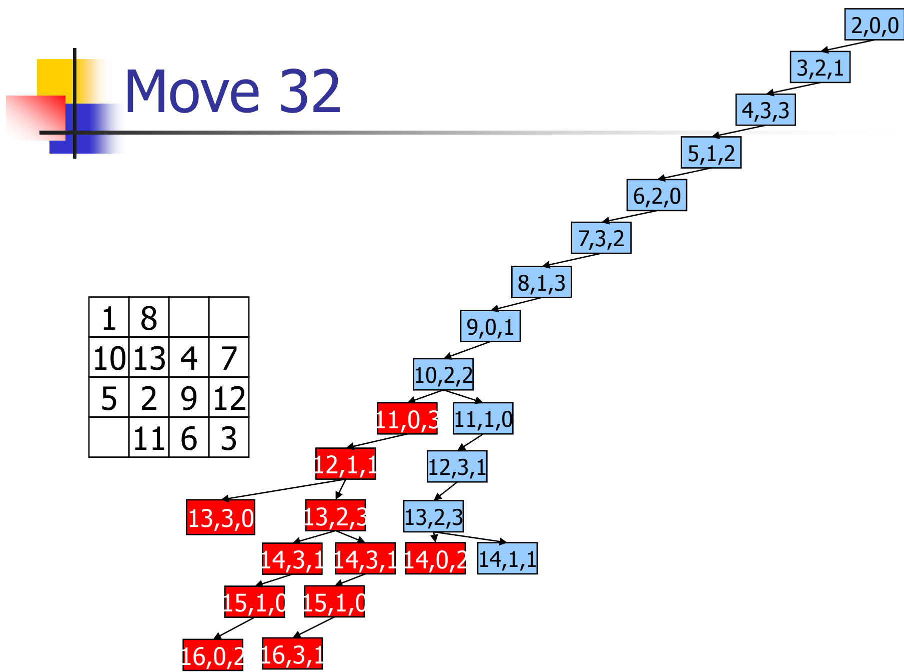

Move 27
2,0,0
3,2,1
4,3,3
5,1,2
6,2,0
7,3,2
8,1,3
9,0,1
10,2,2
11,0,3
12,1,1
13,3,0
13,2,3
14,3,1
14,3,1
15,1,0
15,1,0
16,0,2
16,3,1
1 8 10
4 7
5 2 9
6 3
93

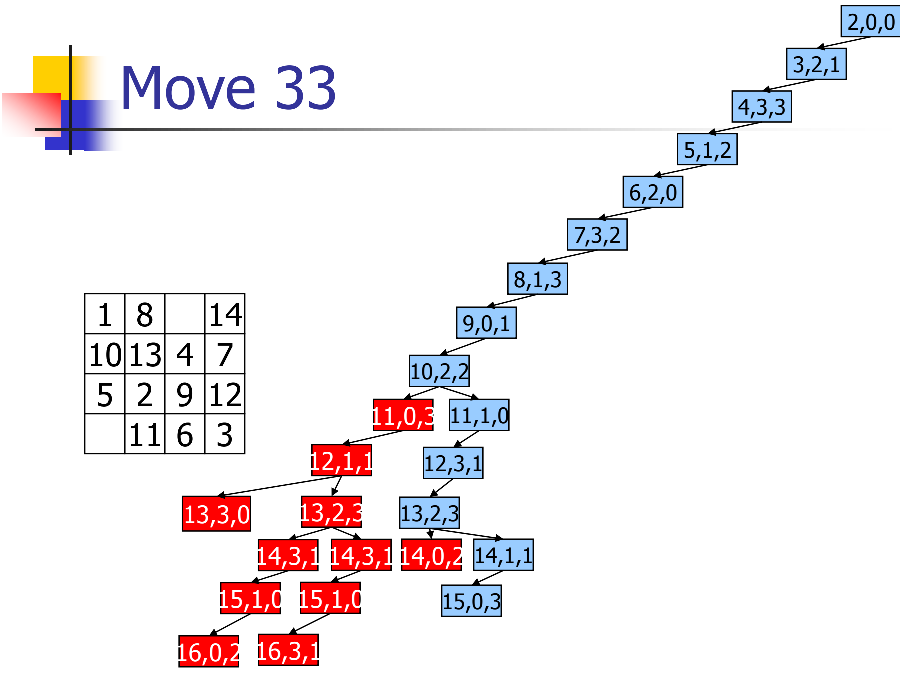

Move 28
2,0,0
3,2,1
4,3,3
5,1,2
6,2,0
7,3,2
8,1,3
9,0,1
10,2,2
11,0,3
11,1,0
12,1,1
13,3,0
13,2,3
14,3,1
14,3,1
15,1,0
15,1,0
16,0,2
16,3,1
1 8
10 4 7
5 2 9
6 3
94

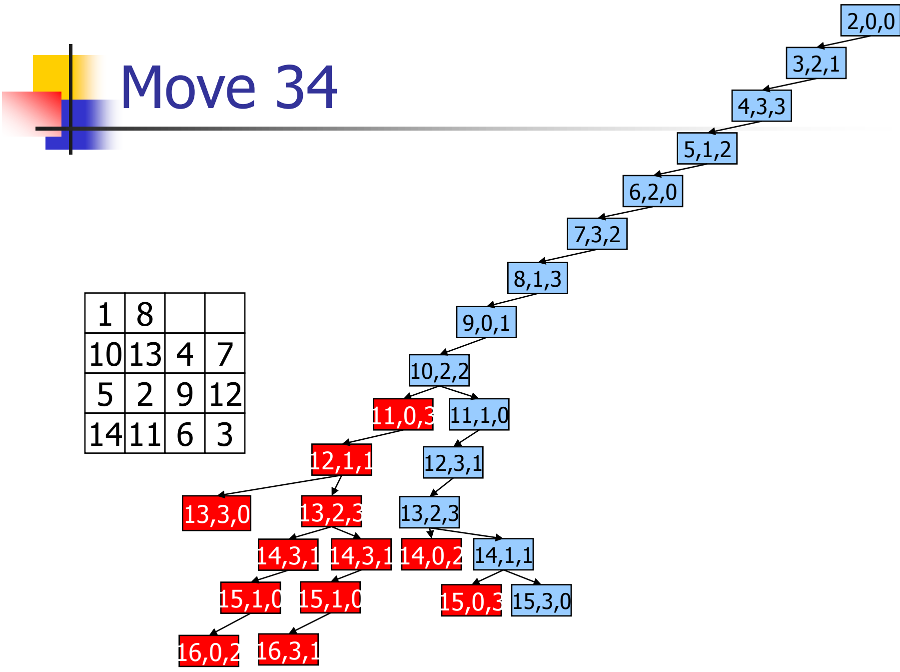

Move 29
2,0,0
3,2,1
4,3,3
5,1,2
6,2,0
7,3,2
8,1,3
9,0,1
10,2,2
11,0,3
11,1,0
12,1,1
12,3,1
13,3,0
13,2,3
14,3,1
14,3,1
15,1,0
15,1,0
16,0,2
16,3,1
1 8
10 4 7
5 2 9
11 6 3
95

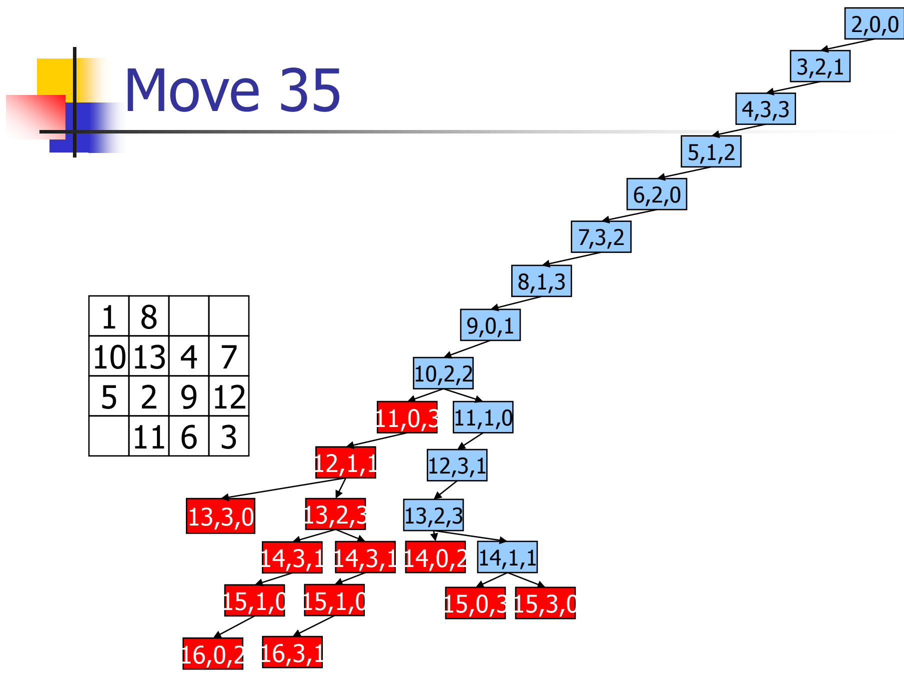

Move 30
2,0,0
3,2,1
4,3,3
5,1,2
6,2,0
7,3,2
8,1,3
9,0,1
10,2,2
11,0,3
11,1,0
12,1,1
12,3,1
13,3,0
13,2,3
13,2,3
14,3,1
14,3,1
15,1,0
15,1,0
16,0,2
16,3,1
1 8
10 4 7
5 2 9 12
11 6 3
96

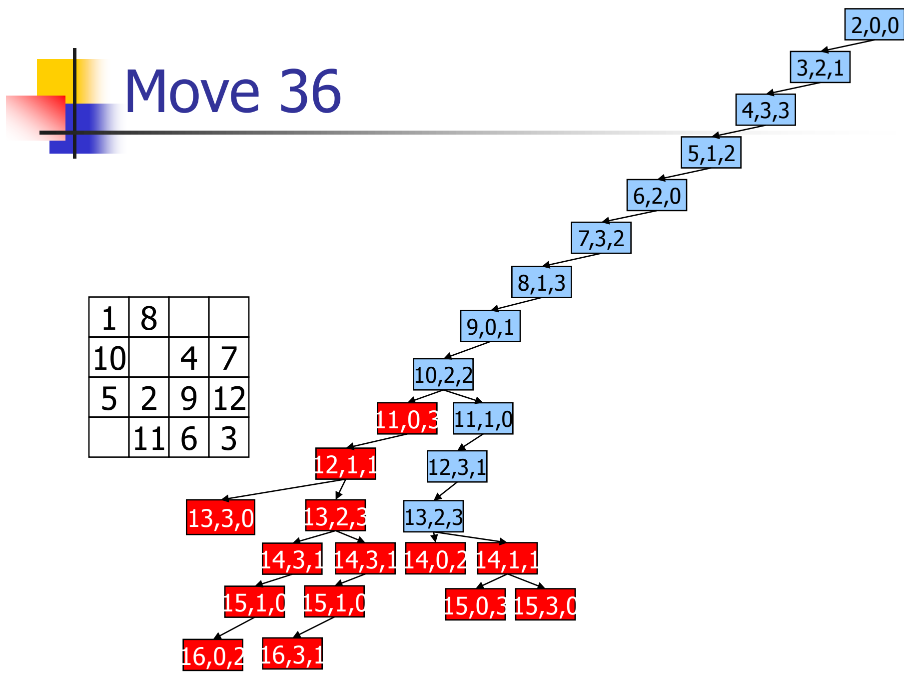

Move 31
2,0,0
3,2,1
4,3,3
5,1,2
6,2,0
7,3,2
8,1,3
9,0,1
10,2,2
11,0,3
11,1,0
12,1,1
12,3,1
13,3,0
13,2,3
13,2,3
14,3,1
14,3,1
14,0,2
15,1,0
15,1,0
16,0,2
16,3,1
1 8 13
10 4 7
5 2 9 12
11 6 3
97

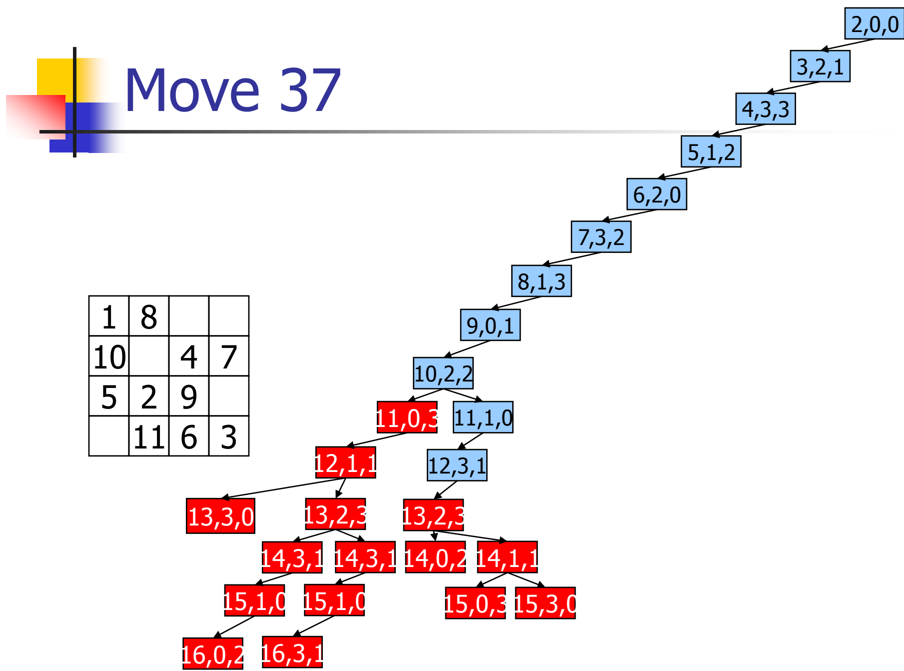

Move 32
2,0,0
3,2,1
4,3,3
5,1,2
6,2,0
7,3,2
8,1,3
9,0,1
10,2,2
11,0,3
11,1,0
12,1,1
12,3,1
13,3,0
13,2,3
13,2,3
14,3,1
14,3,1
14,0,2
14,1,1
15,1,0
15,1,0
16,0,2
16,3,1
1 8
10 13 4 7
5 2 9 12
11 6 3
98

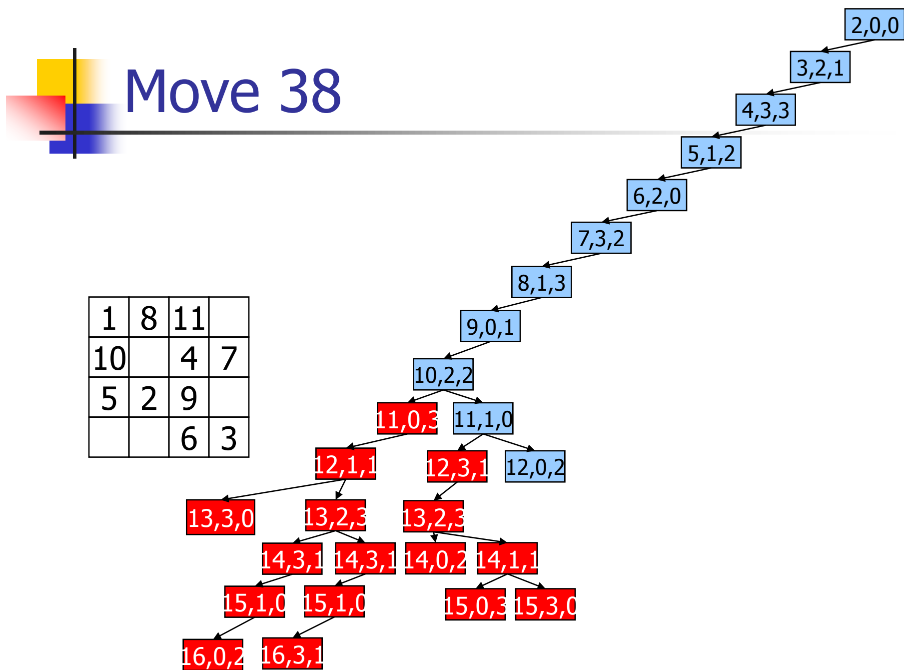

Move 33
1 8 14
10 13 4 7
5 2 9 12
11 6 3
2,0,0
3,2,1
4,3,3
5,1,2
6,2,0
7,3,2
8,1,3
9,0,1
10,2,2
11,0,3
11,1,0
12,1,1
12,3,1
13,3,0
13,2,3
13,2,3
14,3,1
14,3,1
14,0,2
14,1,1
15,1,0
15,1,0
15,0,3
16,0,2
16,3,1
99


Move 34
1 8
10 13 4 7
5 2 9 12
14 11 6 3
2,0,0
3,2,1
4,3,3
5,1,2
6,2,0
7,3,2
8,1,3
9,0,1
10,2,2
11,0,3
11,1,0
12,1,1
12,3,1
13,3,0
13,2,3
13,2,3
14,3,1
14,3,1
14,0,2
14,1,1
15,1,0
15,1,0
15,0,3
15,3,0
16,0,2
16,3,1
100


Move 35
1 8
10 13 4 7
5 2 9 12
11 6 3
2,0,0
3,2,1
4,3,3
5,1,2
6,2,0
7,3,2
8,1,3
9,0,1
10,2,2
11,0,3
11,1,0
12,1,1
12,3,1
13,3,0
13,2,3
13,2,3
14,3,1
14,3,1
14,0,2
14,1,1
15,1,0
15,1,0
15,0,3
15,3,0
16,0,2
16,3,1
101


Move 36
1 8
10 4 7
5 2 9 12
11 6 3
2,0,0
3,2,1
4,3,3
5,1,2
6,2,0
7,3,2
8,1,3
9,0,1
10,2,2
11,0,3
11,1,0
12,1,1
12,3,1
13,3,0
13,2,3
13,2,3
14,3,1
14,3,1
14,0,2
14,1,1
15,1,0
15,1,0
15,0,3
15,3,0
16,0,2
16,3,1
102


Move 37
1 8
10 4 7
5 2 9
11 6 3
2,0,0
3,2,1
4,3,3
5,1,2
6,2,0
7,3,2
8,1,3
9,0,1
10,2,2
11,0,3
11,1,0
12,1,1
12,3,1
13,3,0
13,2,3
13,2,3
14,3,1
14,3,1
14,0,2
14,1,1
15,1,0
15,1,0
15,0,3
15,3,0
16,0,2
16,3,1
103


Move 38
1 8 11
10 4 7
5 2 9
6 3
2,0,0
3,2,1
4,3,3
5,1,2
6,2,0
7,3,2
8,1,3
9,0,1
10,2,2
11,0,3
11,1,0
12,1,1
12,3,1
12,0,2
13,3,0
13,2,3
13,2,3
14,3,1
14,3,1
14,0,2
14,1,1
15,1,0
15,1,0
15,0,3
15,3,0
16,0,2
16,3,1
104

Exercise

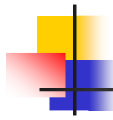
Write a solution in C for the Knight tour problem.

105

Complexity

- Number of moves at each step is at most 8.
- Number of steps is N^2.
- Thus the tree has a number of nodes $\leq 8^{N\wedge 2}$.
- In the worst case
  - The solution is the rightmost leaf
  - The tree is completed, i.e. all the solutions space has been analyzed.
- In this case the number of recursive calls before to find the solution is $\Theta(8^{N\wedge 2})$.

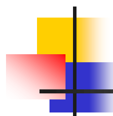
106



Inizialization

```
#define DIM 6
int  a[8],b[8],board[DIM][DIM];
void main(void)
{ int     i,   j,   result;
   a[0]=2; b[0]=1; a[1]=1; b[1]=2;
   a[2]=-1; b[2]=2; a[3]=-2; b[3]=1;
   a[4]=-2; b[4]=-1; a[5]=-1; b[5]=-2;
   a[6]=1; b[6]=-2; a[7]=2; b[7]=-1;

   for( i=0; i<DIM; i++)
      for( j=0; j<DIM; j++)
         board[i][j] = 0;
```

107



Main program

```
   board[0][0] = 1;
   result = moves( 2, 0, 0);
   if( result == 1)
   {  for( i=0; i<DIM; i++)
      {  for( j=0; j<DIM; j++)
            printf( "%2d ",board[i][j]);
         printf( "\n");
      }
   } else
   {  printf( "Solution not found\n");
   }
}
```

108

Moves (1)

```
int moves( int move, int posx, int posy)
{  int    i,ret,newposx,newposy;
   if( move == (DIM*DIM+1))
      return(1);
   for( i=0; i<8; i++)
   {  newposx = posx + a[i];
      newposy = posy + b[i];
```

109

Moves (2)

```
     if( (newposx<DIM) && (newposx>=0) &&
             (newposy<DIM) && (newposy>=0))
     {  if( board[newposx][newposy] == 0)
       {  board[newposx][newposy]=Move;
          ret=moves(move+1, newposx,newposy);
          if( ret == 0)
             board[newposx][newposy]=0;
          else
             return(1);
       }
     }
   }
  return(0);
}
```

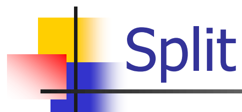
110

X value

When using boolean functions, the symbol X is used to represent whichever boolean value, i.e. 0 or 1.

For example, the OR function returns 1 for input values 01, 10 and 11, and it can be written more concisely X1, 1X.

111



Split

The program receives an inout string made of 0, 1 or X and it must generate all possible combinations matching the input expression.

Example: if input is 01X0X, program generates these combinations

01000

01001

01100

01101

112

Solution

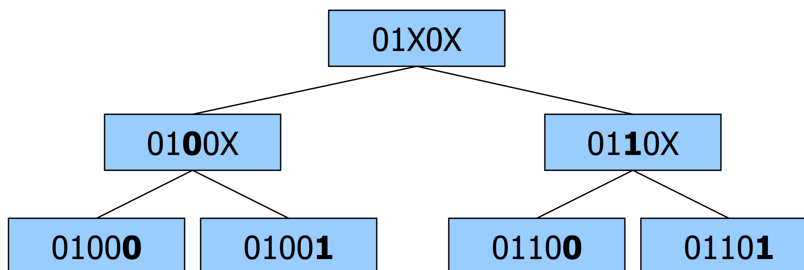
It is based on a recursive algorithm exploring the solution space, i.e. it explores the whole tree of possible combinations matching the input string, by transforming each X symbol in 0 and then in 1.

The number of leaves of such tree (each one corresponding to a possible combination) is equal to 2^N, where N is the number of X symbols in the input string.

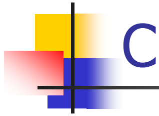
The tree's height is N+1.

113

Combinations Tree



114

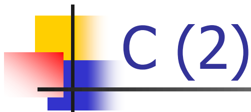


C

```
#include <stdio.h>

#define MAX 100

char    ibuff[MAX],
        obuff[MAX];
int     index,
        len;

void split(int);
```

```
void main(void)
{

  gets( ibuff);

  len = strlen( ibuff);

  split(0);

}
```

115



C (2)

```
void split(int index)
{
  if( index == len)
  {
       obuff[index]='\0';
       printf( "%s\n", obuff);
       return;
  }
```

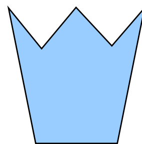
```
switch( ibuff[index])
  {   case '0':
      case '1':
        obuff[index] = ibuff[index];
        split(index+1);
        return;
      case 'X':
        obuff[index] = '0';
        split(index+1);
        obuff[index] = '1';
        split(index+1);
        return;
  }
}
```

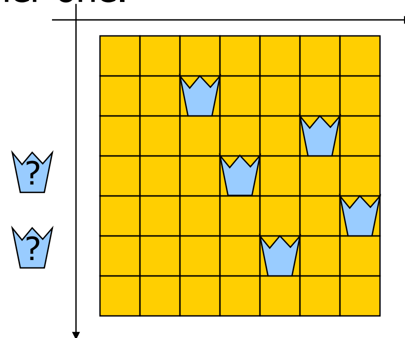
116

The N queens problem

Given a NxN board, and given N queens of chess game.

Find a layout of the N queens so that no one can be taken by another one.



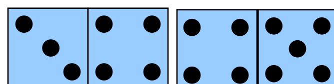


117

Domino Game

Given N tiles of Domino game, each one has two faces, labeled with a number between 1 and 6.

Find the longest correct sequence of tiles, so that adjacent faces of consecutive tiles must have the same value.



118