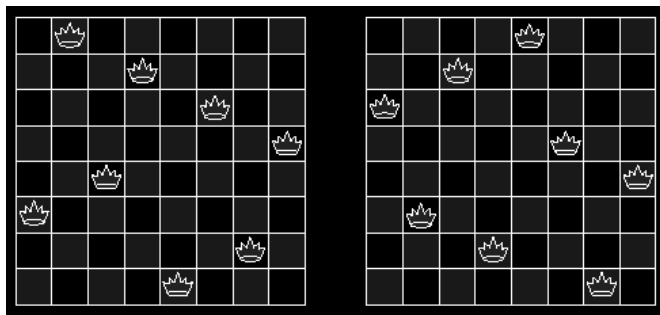


A Useful Case

15

A Fruitful Application of Recursion: n-queen problem



Apparently an analytically unsolvable problem. Even C. F. Gauss, who attempted this in 1850 was perplexed by this problem. But, solution do exists. See the two shown above.



DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-16
ALG00S, javed@kent.edu
Javed I. Khan@1999

Solution Outline

```
void
AddQueen(void)
{
    for (every unguarded position p on the
board) {
        Place a queen in position p;
        n++;
        if (n == 8)
            Print the configuration;
        else
            AddQueen();
        Remove the queen from position p;
        n--;
    }
}
```



DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-17
ALG00S, javed@kent.edu
Javed I. Khan@1999

Example 4-Queen



Dead end

(a)



Dead end

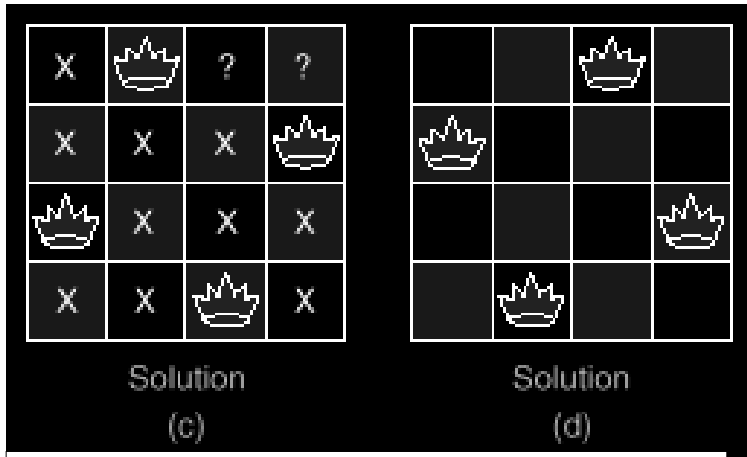
(b)



DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-18
ALG00S, javed@kent.edu
Javed I. Khan@1999

Example 4-Queen



Backtracking: Build correct partial solution and proceed. When an inconsistent state arises, the algorithm backs up to the point of last correct partial solution.

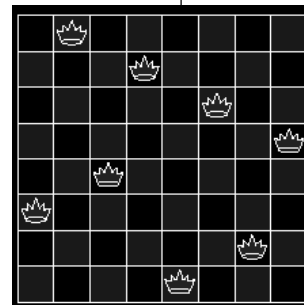


DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-19
ALG00S, javed@kent.edu
Javed I. Khan@1999

Choice of Data Structure

- Boolean array or integer array?
 - Keep a count of check, to help backtracking.
- Search later or mark ahead?
- Pigeon hole principle
 - use one row one queen to reduce search.
- Keep track of free columns
 - `int col[8]`
- Keep track of free diagonals
 - number of diagonal $2 * \text{boardsize} - 1$
 - all downdiagonal $(x-y) = \text{constant}$
 - all updiagonal $(x+y) = \text{constant}$



DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-20
ALG00S, javed@kent.edu
Javed I. Khan@1999

```

#include "common.h"
#define BOARDSIZE 8
#define DIAGONAL (2*BOARDSIZE-1)
#define DOWNOFFSET 7

void WriteBoard(void);
void AddQueen(void);

int queencol[BOARDSIZE]; /* column with the queen */
Boolean colfree[BOARDSIZE]; /* Is the column free? */
Boolean upfree[DIAGONAL]; /* Is the upward diagonal free? */
Boolean downfree[DIAGONAL]; /* Is the downward diagonal free? */

int queencount = -1, /* row whose queen is currently placed */
*/
numsol = 0; /* number of solutions found so far */
*/

int main(void)
{
    int i;

    for (i = 0; i < BOARDSIZE; i++)
        colfree[i] = TRUE;

    for (i = 0; i < DIAGONAL; i++) {
        upfree[i] = TRUE;
        downfree[i] = TRUE;
    }
    AddQueen();

    return 0;
}

```

Main()



DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-21
ALG00S, javed@kent.edu
Javed I. Khan@1999

```

void AddQueen(void)
{
    int col; /* column being tried for the queen */

    queencount++;
    for (col = 0; col < BOARDSIZE; col++)
        if (colfree[col] && upfree[queencount + col] &&
            downfree[queencount - col + DOWNOFFSET]) {
            /* Put a queen in position (queencount, col). */
            queencol[queencount] = col;
            colfree[col] = FALSE;
            upfree[queencount + col] = FALSE;
            downfree[queencount - col + DOWNOFFSET] = FALSE;
            if (queencount == BOARDSIZE-1) /* termination condition */
                WriteBoard();
            else
                AddQueen(); /* Proceed recursively. */
            colfree[col] = TRUE; /* Now backtrack by removing the
queen. */
            upfree[queencount + col] = TRUE;
            downfree[queencount - col + DOWNOFFSET] = TRUE;
        }
    queencount--;
}

```

AddQueen()



DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-22
ALG00S, javed@kent.edu
Javed I. Khan@1999

Analysis

- Naïve approach:
 - generate a random configuration and test it.

$$\binom{64}{8} = 4,426,165,368$$

- One queen per row

$$8^8 = 16,777,216$$

- One queen per column

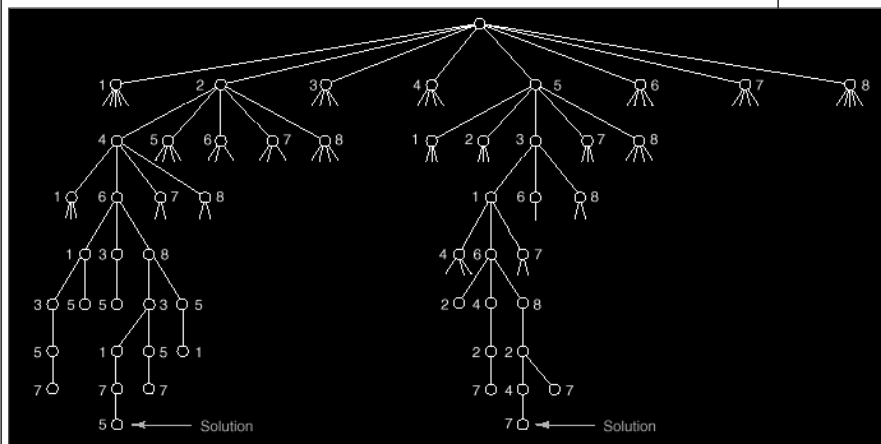
$$8! = 40,320$$



DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-23
ALG00S, javed@kent.edu
Javed I. Khan@1999

Part of Recursion Tree for 8-queen



Height & Number of Nodes



DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-24
ALG00S, javed@kent.edu
Javed I. Khan@1999

Principles of Recursive Program Design

25

Designing Recursive Algorithms

- Find the key step.
 - How can this problem be divided into parts?
 - How will the key step in the middle be done?
 - Avoid ending with multitude of special cases.
- Find a stopping rule.
 - This stopping rule is usually the small case that is easy to handle without recursion.
- Outline your algorithm.
 - Combine the stopping rule and the key step, using an if statement to select between them.



DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-26
ALG00S, javed@kent.edu
Javed I. Khan@1999

Designing Recursive Algorithms

(Continued..)

- **Check termination.**
 - Verify that the recursion will always terminate.
 - Be sure that your algorithm correctly handles extreme cases.
- **Draw a recursion tree.**
 - The height of the tree is closely related to the amount of memory that the program will require,
 - the total size of the tree relates to the number of times the key step will be done.

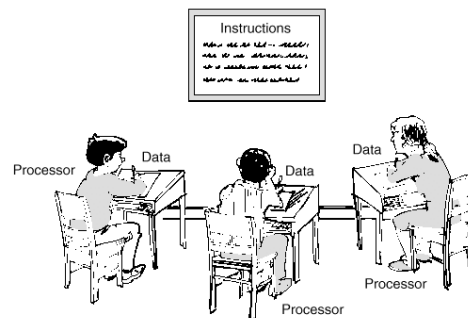


DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-27
ALG00S, javed@kent.edu
Javed I. Khan@1999

Implementing Recursion

- **Implementation is separate from design.**
 - Implementation can be in any language.
- **Multiple Processors:**
 - Processes that take place simultaneously are called **concurrent**.
- **Single Processor:**
 - can use multiple storage areas with a single processor.
- **Re-entrant Programs**




DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-28
ALG00S, javed@kent.edu
Javed I. Khan@1999

Improving Recursive Programs: The case of Tail Recursion

29



**DESIGN &
ANALYSIS OF
ALGORITHM**

Who creates/manages these stacks?

LECT-03, S-30
ALG00S, javed@kent.edu
Javed I. Khan@1999

Tail Recursion



DESIGN &
ANALYSIS OF
ALGORITHM

DEFINITION **Tail recursion** occurs when the last-executed statement of a function is a recursive call to itself.

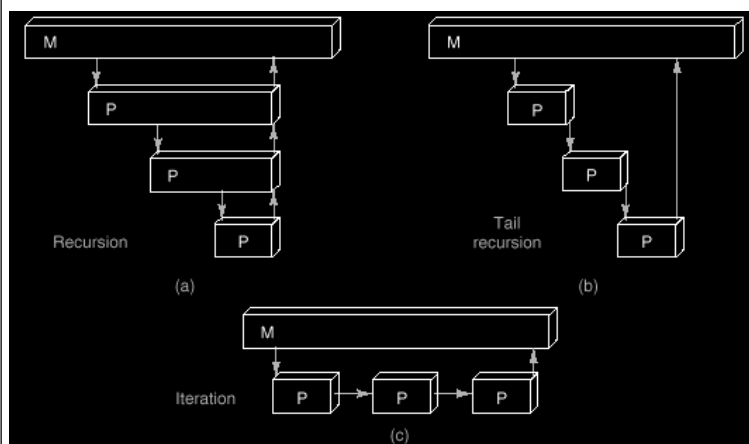
If the last-executed statement of a function is a recursive call to the function itself, then this call can be eliminated by reassigning the calling parameters to the values specified in the recursive call, and then repeating the whole function.

LECT-03, S-31
ALG00S, javed@kent.edu
Javed I. Khan@1999

Removal of Tail Recursion



DESIGN &
ANALYSIS OF
ALGORITHM



LECT-03, S-32
ALG00S, javed@kent.edu
Javed I. Khan@1999

Can Tower of Hanoi be simplified?

```
/* Move: moves count disks from start to finish using temp
for temporary storage. */

void Move(int count, int start, int finish, int temp)
{
    if (count > 0) {
        Move(count-1, start, temp, finish);
        printf("Move a disk from %d to %d.\n", start,
finish);
        Move(count-1, temp, finish, start);
    }
}
```



DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-33
ALG00S, javed@kent.edu
Javed I. Khan@1999

Iterative Tower of Hanoi

```
void Move(int count, int start, int finish, int temp)
{
    if (count > 0) {
        Move(count-1, start, temp, finish);
        printf("Move a disk from %d to %d.\n", start, finish);
        Move(count-1, temp, finish, start);
    }
}
```

```
void Move(int count, int start, int finish, int temp)
{
    int swap; /* temporary storage to swap towers */
    while (count > 0) {
        Move(count - 1, start, temp, finish);
        printf("Move %d from %d to %d.\n", count, start, finish);
        count--;
        swap = start;
        start = temp;
        temp = swap;
    }
}
```



DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-34
ALG00S, javed@kent.edu
Javed I. Khan@1999



Can n-queen be simplified?



```
void AddQueen(void)
{
    int col;    /* column being tried for the queen */

    queencount++;
    for (col = 0; col < BOARDSIZE; col++)
        if (colfree[col] && upfree[queencount + col] &&
            downfree[queencount - col + DOWNOFFSET]) {
            /* Put a queen in position (queencount, col). */
            queencol[queencount] = col;
            colfree[col] = FALSE;
            upfree[queencount + col] = FALSE;
            downfree[queencount - col + DOWNOFFSET] = FALSE;
            if (queencount == BOARDSIZE-1) /* termination condition
            */
                WriteBoard();
            else
                AddQueen();           /* Proceed recursively.
            */
            colfree[col] = TRUE;      /* Now backtrack by removing the
            queen. */
            upfree[queencount + col] = TRUE;
            downfree[queencount - col + DOWNOFFSET] = TRUE;
        }
    queencount--;
}
```

Flashback: AddQueen()

Should we Always use Recursion? Case: Factorials

- Recursive Version:

```
/* Factorial: recursive version.*/  
int Factorial(int n)  
{  
    if (n == 0)  
        return 1;  
    else  
        return n * Factorial(n-1);  
}
```

Which one will work harder?

- Iterative Version?

```
/* Function: iterative version.*/  
int Factorial(int n)  
{  
    int count, product;  
    for (product = 1, count = 2; count <= n;  
        count++)  
        product *= count;  
    return product;  
}
```



DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-37
ALG00S, javed@kent.edu
Javed I. Khan@1999

Fibonacci Numbers

- Fibonacci Number

$$F_0 = 0$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2} \quad \text{when } n \geq 2$$



DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-38
ALG00S, javed@kent.edu
Javed I. Khan@1999

A Cute Solution

```
int Fibonacci(int n)
{
    if (n <= 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return Fibonacci(n-1) + Fibonacci(n-2);
}
```

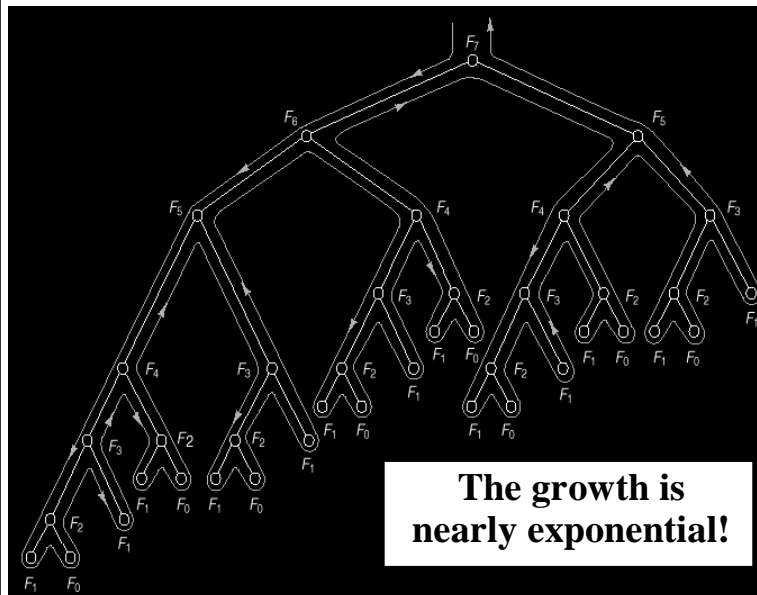
Efficiency?



DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-39
ALG00S, javed@kent.edu
Javed I. Khan@1999

Recursion Tree



DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-40
ALG00S, javed@kent.edu
Javed I. Khan@1999

Iterative Fibonacci

```
/* Fibonacci: iterative version.*/  
  
int Fibonacci(int n)  
{  
    int i;  
    int twoback; /* second previous number, F_i-2 */  
    int oneback; /* previous number, F_i-1 */  
    int current; /* current number, F_i */  
  
    if (n <= 0)  
        return 0;  
    else if (n == 1)  
        return 1;  
    else {  
        twoback = 0;  
        oneback = 1;  
  
        for (i = 2; i <= n; i++) {  
            current = twoback + oneback;  
            twoback = oneback;  
            oneback = current;  
        }  
        return current;  
    }  
}
```



DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-41
ALG00S, javed@kent.edu
Javed I. Khan@1999

Comparison: Iteration vs. Recursion

- Chain
- Duplicate Task
- Change Data Structures
- Recursion Removal



DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-42
ALG00S, javed@kent.edu
Javed I. Khan@1999

Guidelines

- If the recursion tree has a simple form, the iterative version may be better.
- If the recursion tree involves duplicate task, then data structures other than stacks will be appropriate.
- If the recursion tree appears quite bushy, with little duplicate tasks, then recursion is likely the natural solution.



DESIGN &
ANALYSIS OF
ALGORITHM

LECT-03, S-43
ALG00S, javed@kent.edu
Javed I. Khan@1999