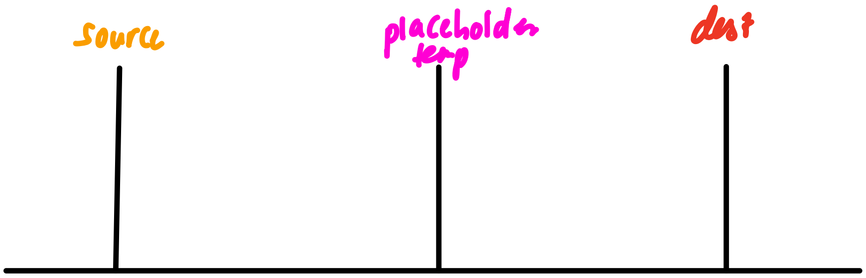
Recursions

Tower of Hanoi



Objective: move pegs from Source to destination with least number of moves

Constraints: A large peg may not be placed on top of a smaller peg

Recursive logic:

Base case: Assume # discs == 1, we can just move it from source to destination

Wishful thinking: We can move k-1 discs to another peg (from src → tmp & from tmp → dest)

1. moving k-1 discs from A to B (A - src, B - dest, C - temp)
2. move largest disc from A to B (trivial case)
3. Moving k-1 discs from B to C (B - src, C - dest, A - temp)

Abstraction of the problem:

- Represent discs as integers 1 to k
- Represent pegs as chars A, B, C

```
void move(long k, char src, char dest){
    cs1010_print_string ("Disk ");
    cs1010_print_long (k);
```

```
    cs1010_print_string(". Peg ");
    putchar(source);
    cs1010_print_string(" → Peg ");
    putchar(dest);
    cs1010_println_string("");
}

void solve(long k, char src, char dest, char placeholder){
    if (k==1){
      move(1, src, dest);
      return; //terminates recursion
    }
    solve(k-1, src, tmp, dest);
    solve(k, src, dest, tmp);
    solve(k-1, tmp, dest, src);
}
```

$T(n) = 2T(n-1)+1$

$= 2(2T(n-2)+1)+1 = 2^2 T(n-2) + 2+1$

$= 2^3(T(n-3)) + 4+2+1$

$\vdots$

$2^k(T(n-k)) + 2^{k-1} + \dots + 1$

$\vdots$

$= 2^{n-1}(T(n-(n-1))) + 2^{n-2} + \dots + 1$

$= 2^{n-1}(T(1)) + 2^{n-2} + \dots + 1$

$$= 2^{n-1} + 2^{n-2} + \cdots + 1$$

$$= \frac{1(2^n - 1)}{2-1}$$

$$= 2^n - 1$$

$$= O(2^n)$$

$$\Rightarrow \frac{2^{n-1}(1-2^{-n})}{1-(\frac{1}{2})} = 2(2^{n} - 2^{-1}) = 2^n - 1$$

Generate permutations

Base case: Generate permutation for trivial case (1 char → 1 permutation)

Wishful thinking: Assume we can solve the problem for string of length k-1

3 chars abc

$$\begin{array}{llll} \text{start with} & a & \text{permute} & bc \\ & b & & ac \\ & c & & \underbrace{ab}_{\substack{\text{wishful thinking} \\ k-1}} \end{array}$$

fixed prefix, changing suffix

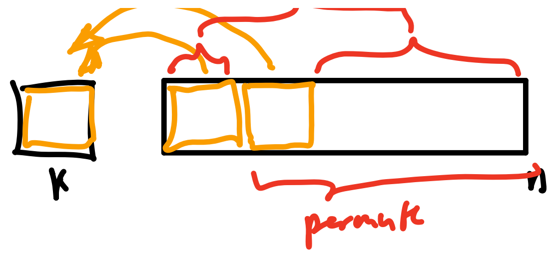
starting index of permutation

```
void permute(char a[], size_t n, size_t k){
    if (k==n-1){   ← only one character to permute as the other are fixed
        cs1010_println_string(a);
        return;      //terminate recursion
    }

    for (size_t i=k; i<n; i+=1){
```

permute

```
        swap (a, k, i);
        permute (a, n, k+1);
        swap (a, i, k);
    }
}
```



Time complexity: $O(n!)$

Run time analysis

$$T(n) \begin{cases} nT(n-1) \\ 1, \; n=1 \end{cases}$$

$$\begin{aligned} T(n) &= nT(n-1) \\ &= n(n-1)(T(n-2)) \\ &\;\vdots \\ &= n(n-1)(n-2)\ldots(n-(n-1))T(1) \\ &= n! \end{aligned}$$

$O(n!)$

Nqueens

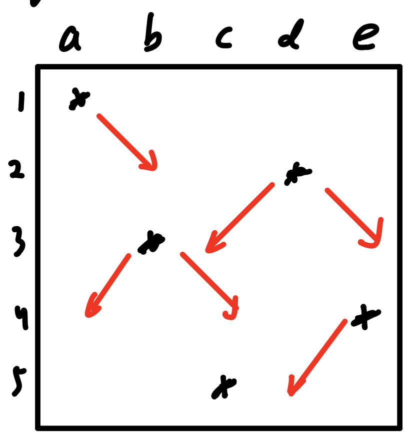
Chess board of size $n \times n$, place n queens such that they do not attack each other.

Queen cannot be in the same row, same column or same diagonal as another.

Solution only exists for n>=4.

One queen per row & one queen per column

We can represent the queen with a column id



"adbec"

Queens are at

a1, d2, b3, e4, c5

How to solve the problem?

① Generate all possible permutations of queens (permutations of "abcde")

② Check if placements are valid (only need to check diagonals as we have ensured there is only 1 queen per row & column)

```
void nqueens(char a[], size_t n, size_t k){
    if (k==n-1){
        if (!threaten_each_other_diagonally(a, n-1)){
            cs1010_println_string(a);
        }
        return;
    }
    for (size_t i=k; i<n; i+=1){
```

```
            swap (a,k, i);
            nqueens(a,n,k+1);   add extra line here to check if a->k
                                has a valid configuration.
            swap(a,i,k);
        }
}


bool threaten_each_other_diagonally(char queens[], size_t end_row){
    for(size_t begin_row = 0; begin_row <= end_row; begin_row += 1){
      if (has_a_queen_in_diagonal(queens, begin_row, end_row){
            return true;
       }
    }
    return false;
}


bool has_a_queen_in_diagonal(const char queens[], size_t curr_row, size_t end_row){
    char curr_col = queens[curr_row]
    char left_col = curr_col - 1
    char right_col = curr_col + 1
    for(size_t row = curr_row + 1; row <= end; row += 1){
        if (queens[row] == left_col || queens[row] == right_col){
            return true;
        }
        left_col -= 1;
        right_col += 1;
```

```
    }
    return false;
}
```

searching - search for all possible combinations

pruning - optimising search based on constraints +

branch & bound problem

↓

don't search if solution is not valid