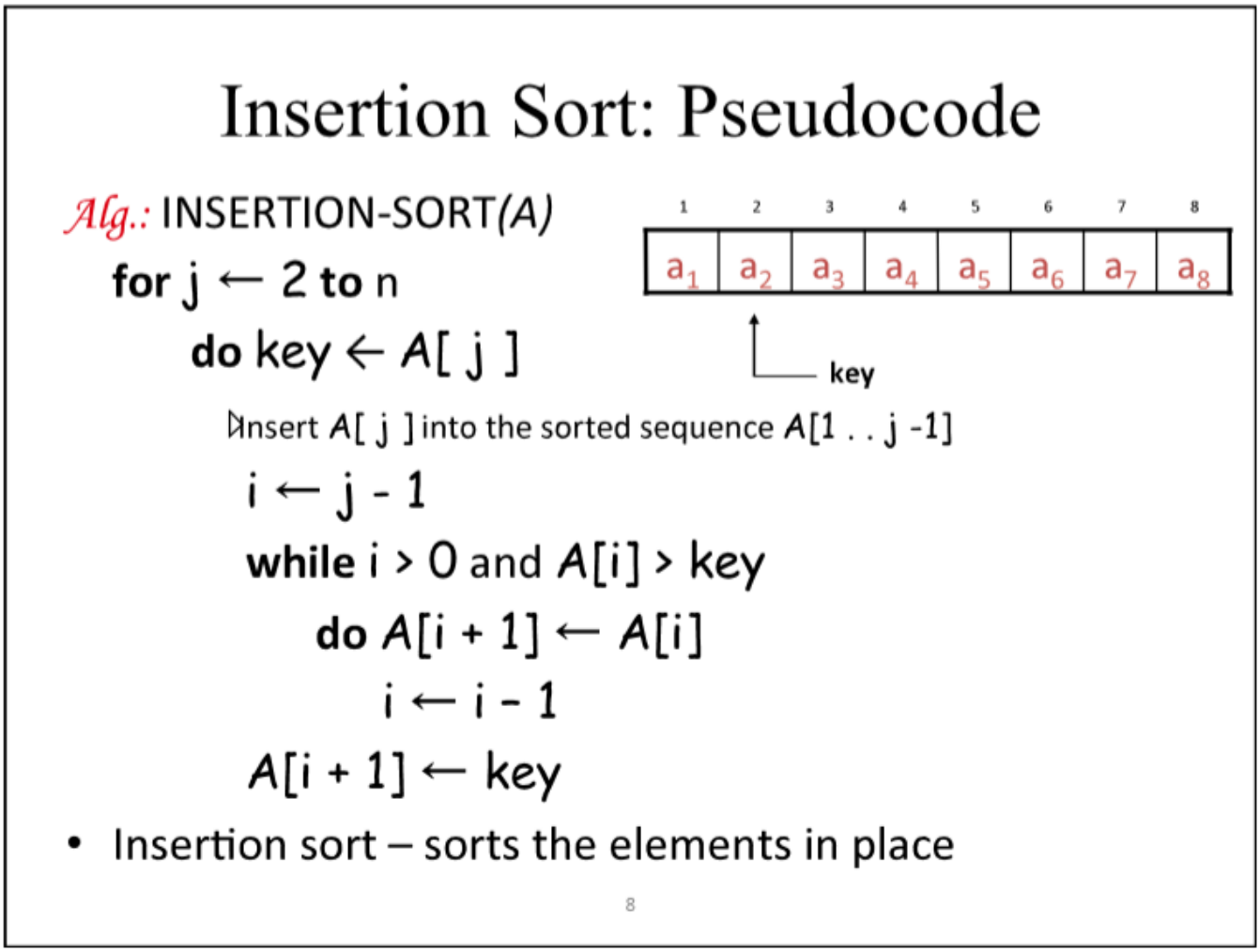
Insertion and Merge Sorts

Sorting is the best studied problem in computer science, because computers spend more time sorting than anything else

Insertion Sort

Iterative
Insertion sort is like sorting cards in your hand. If all the cards are face down on the table, pick up one card. Pick up another card, and put it in the right spot. Continue till you've picked up all the cards



Insertion Sort: Analysis
Best case: O(n)

- Array is already sorted. Inner loop never happens
- Number of moves: 2(n-1)
  Worst Case: $O(^2)$
- Array is in reverse order. Inner loop is executed p-1 times, for p = 2,3,.....n

- Number of moves: 2*(n-1) + (1 + 2 +... +n-1) = 2(n-1) + n*(n-1)/2 or $O(n^2)$

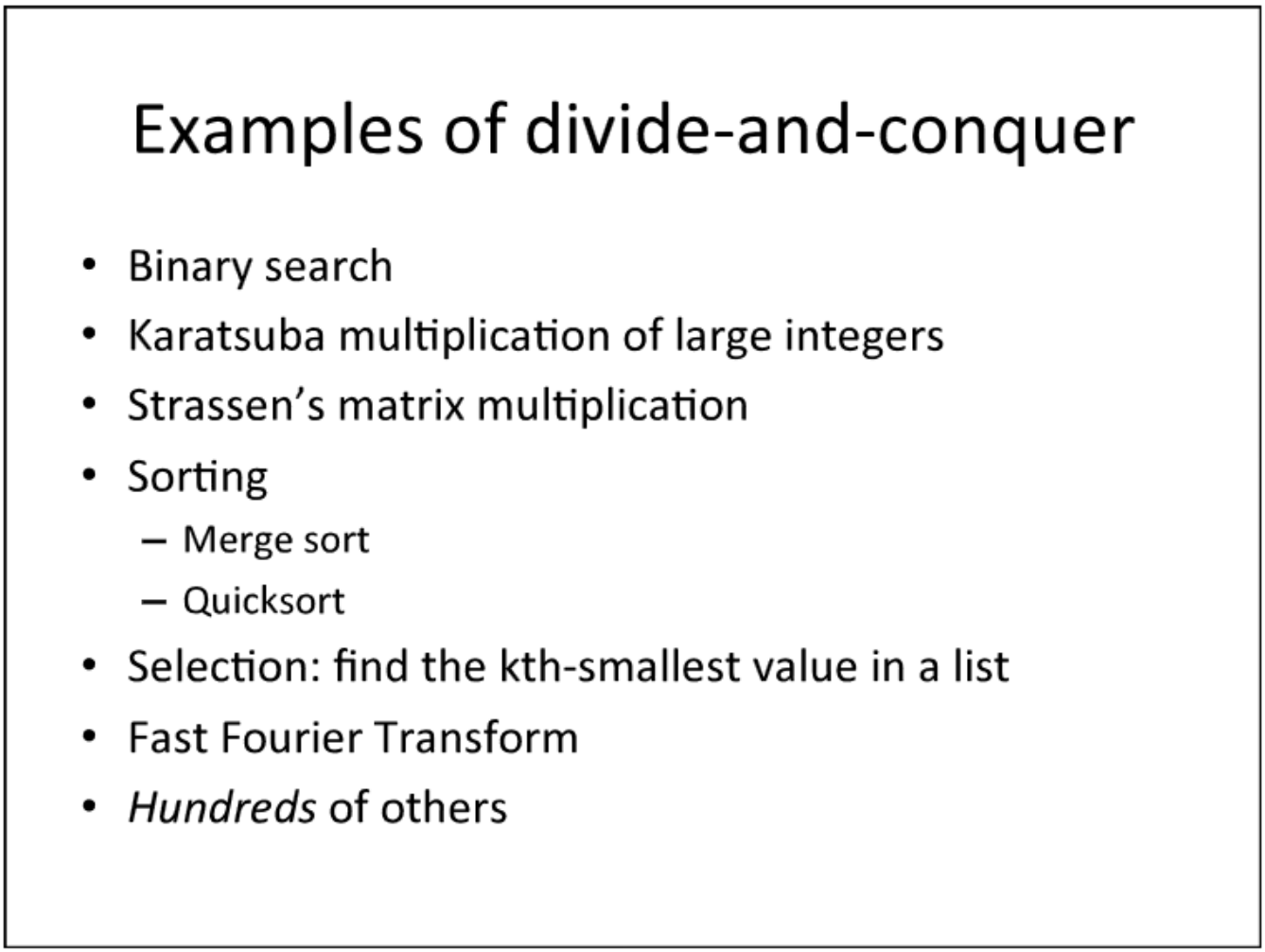
  Average-case: $O(n^2)$
- We have to look at all possible initial data organizations.

  Therefore: Insertion Sort is $O(n^2)$

Divide and Conquer

This is an important strategy for algorithm design. The idea is to

- **Divide** the problem into two or more smaller problems, ideally they're same size
- **Conquer** the sub-problems by solving them recursively. If they're small enough, it can be straightforward
- **Combine** the solutions into a solution for the og problem



Merge Sort

Recursive

Best/Worst/Average Case: O(nlogn)

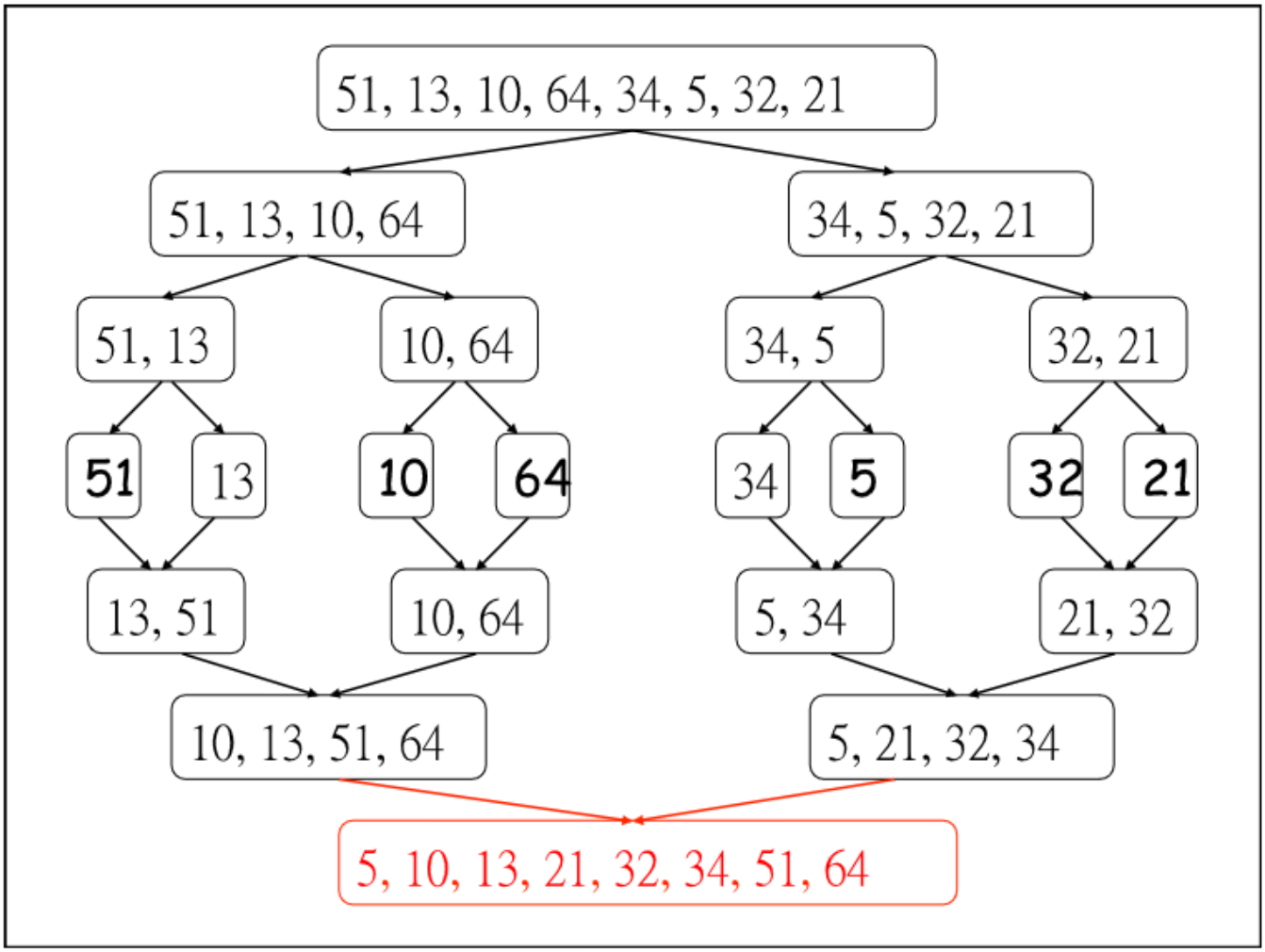
A prime example of divide and conquer, invented by John Von Neumann in 1945.

INPUT: a sequence of *n* numbers stored in array A

OUTPUT: an ordered sequence of *n* numbers

```
MergeSort (A, p, r)  // sort A[p..r] by divide & conquer
1    if p < r
2       then q ← ⌊(p+r)/2⌋
3            MergeSort (A, p, q)
4            MergeSort (A, q+1, r)
5            Merge (A, p, q, r) // merges A[p..q] with A[q+1..r]
```

Initial Call: MergeSort(*A*, 1, *n*) where n = length(A)



Running time T(n) of Merge Sort

Divide: computer the middle takes θ(1)

Conquer: Solving 2 subproblems takes 2T(n/2)

Combine: merging n elements takes θ(n)

Total:

$T(n) = \theta(1)$ if $n=1$

$T(n) = 2T(n/2) + \theta(n)$ if $n>1$