Introduce the method of designing a DIVIDE AND CONQUER algorithm

Subject: Design and Analysis of Algorithms – CS112.L23.KHCL

Lecturer : Nguyen Thanh Son

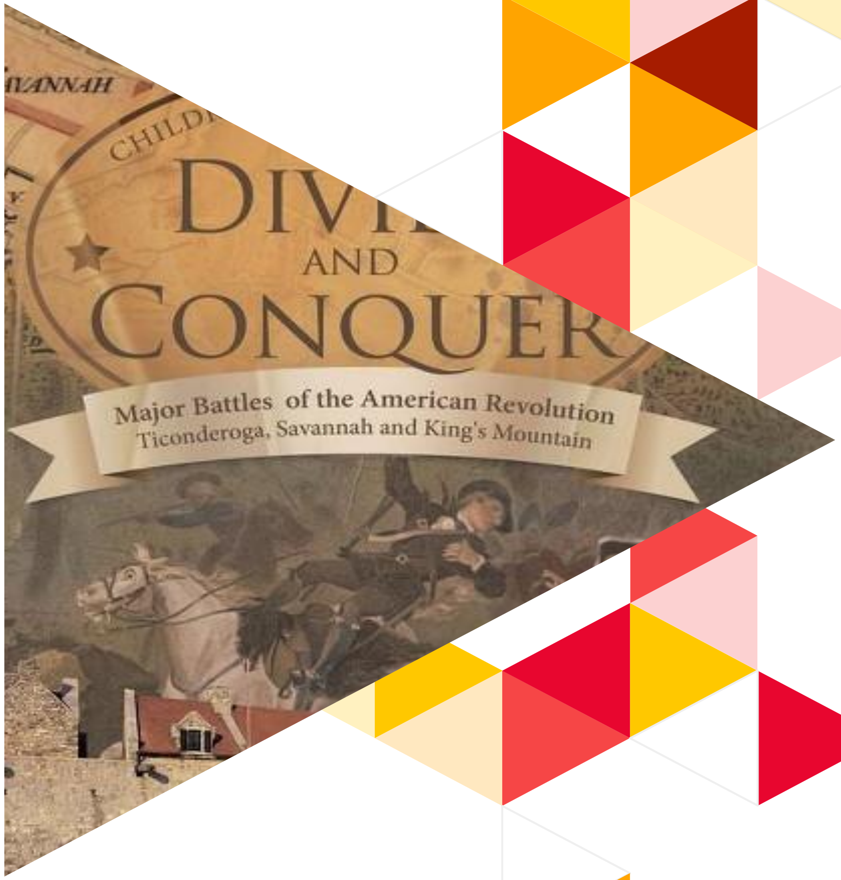
Group 11

Member:

Truong Quoc Truong

Nguyen Quang Tuan

Nguyen Ngoc Tan


VANNAH
CHILD
DIVI
AND
CONQUER
Major Battles of the American Revolution
Ticonderoga, Savannah and King's Mountain

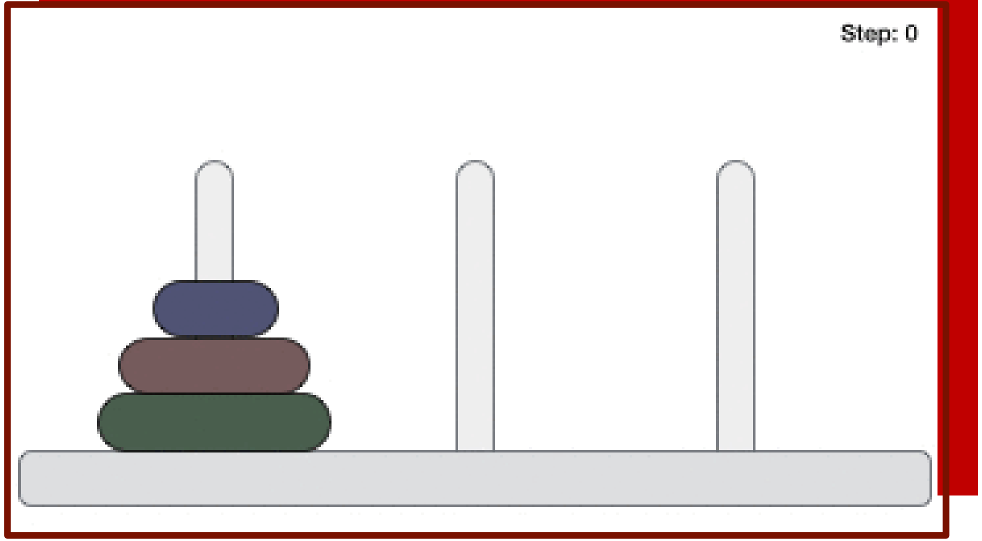
Overview

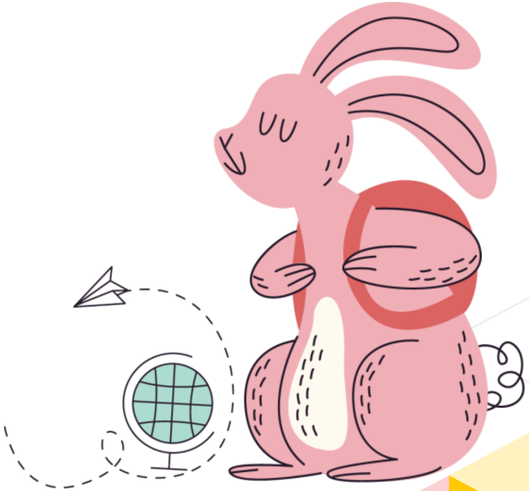
- I: Introduce algorithm
- II: Generality algorithm
- III: Problems
- IV: Conclusion

www.slidescarnival.com/help-use-presentation-template

Warm up

The classic problem in the practice part of Data Structure and Algorithms is the **Hanoi Tower** lesson.





1.

Introduce Algorithm

Divide and conquer algorithm

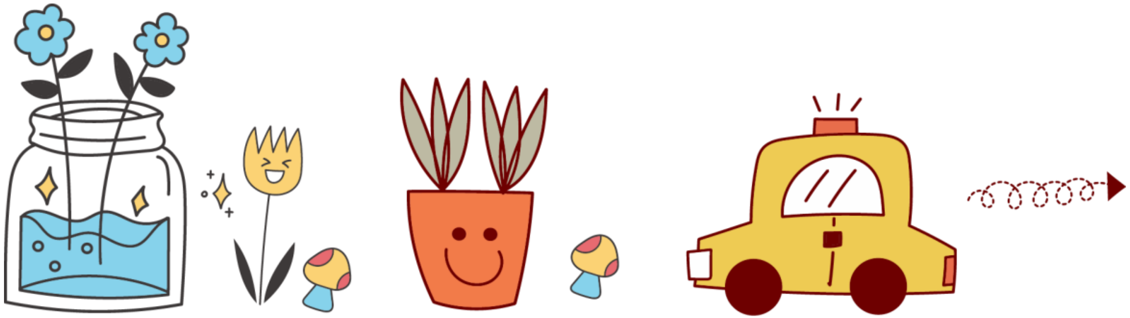
Introduce Algorithm

Introduce Algorithm

What is divide and conquer?

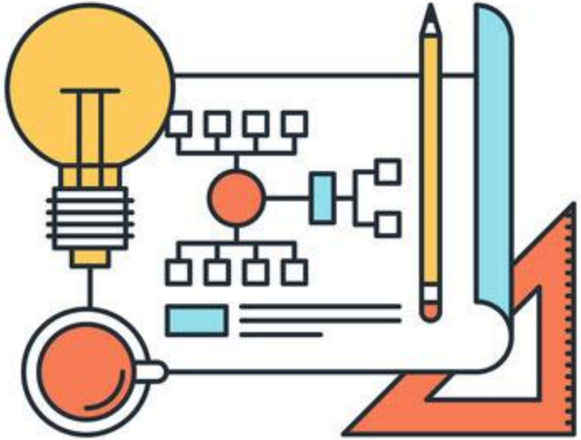
- It is an algorithm design paradigm, isn’t a programming technique.

- The basis of efficient algorithms for many problems.



Introduce Algorithm

Have you
seen or used the
divide-and-conquer
method?



2.

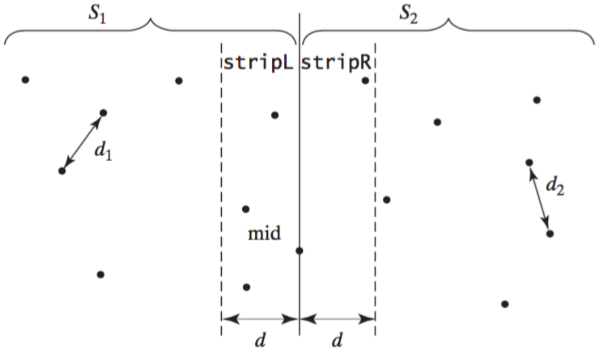
Generality Algorithm

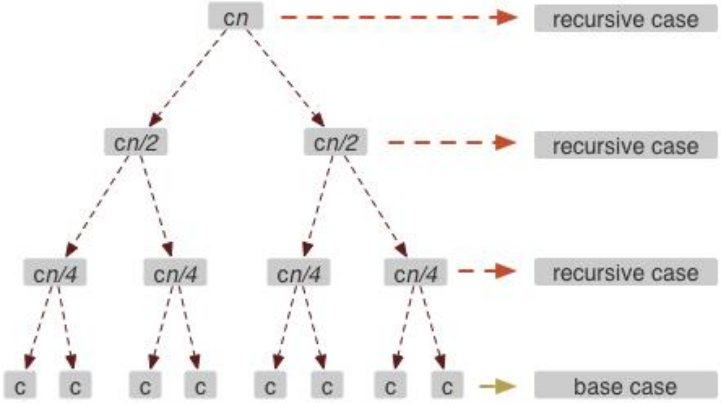
Divide and conquer algorithm

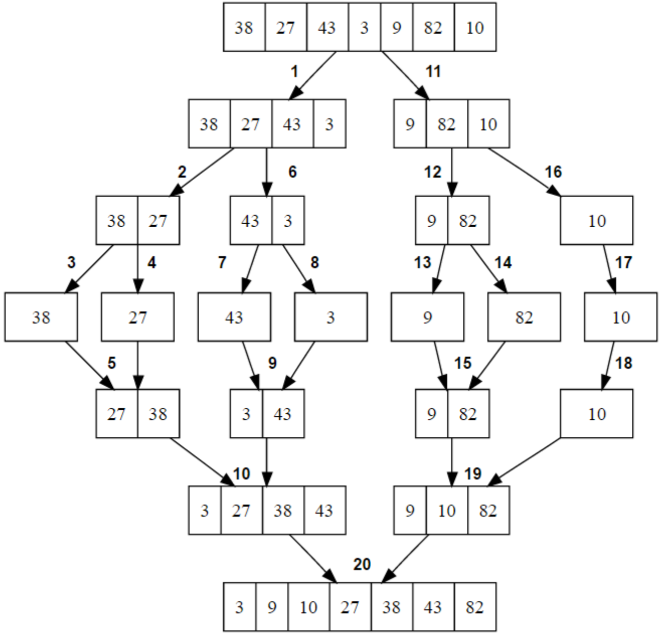
Feature of the problem

How do we know it's a divide-and-conquer problem?

What do these problems have in common?

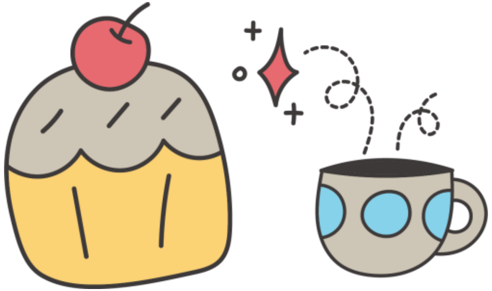






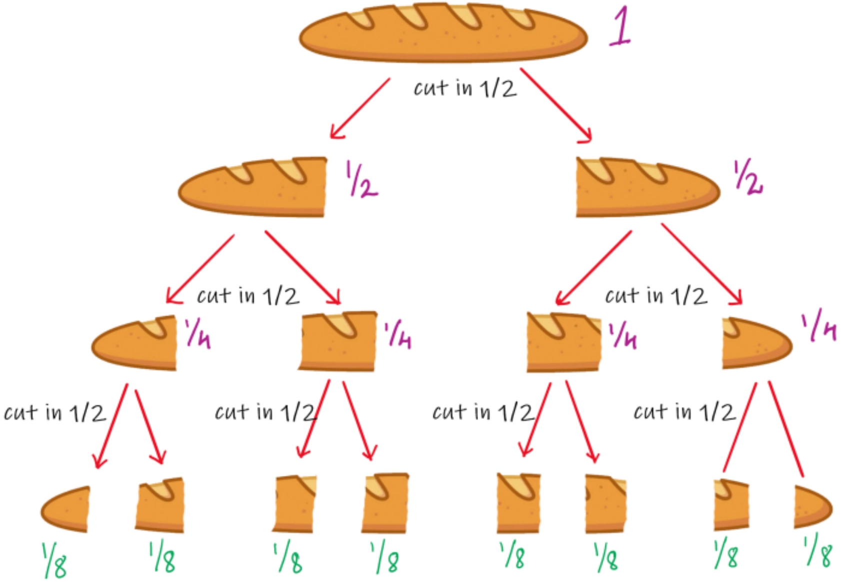
- The original problem is replaced with a more general problem in order to create a recursive relationship.

- Each difficult or large problem and it can be divided into two or more subproblems.



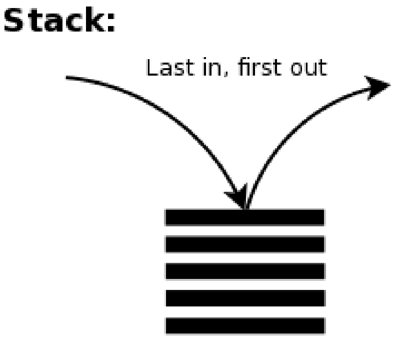
Question

Why do we have to use recursive call? If we don't use recursive calls, what other ways can we use them?



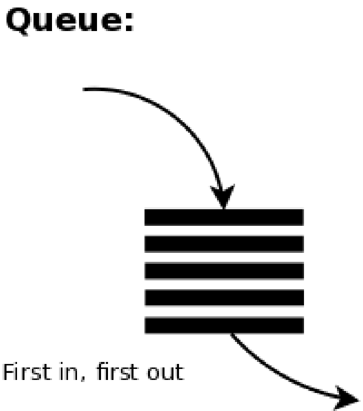
- Algorithm creates at least two subproblems, so it makes multiple recursive calls. It’s easier and more brief.

- In addition to the recursive call, we can also use stack data structures, queue, etc.



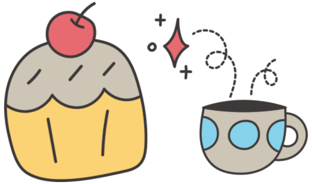
Stack:

Last in, first out

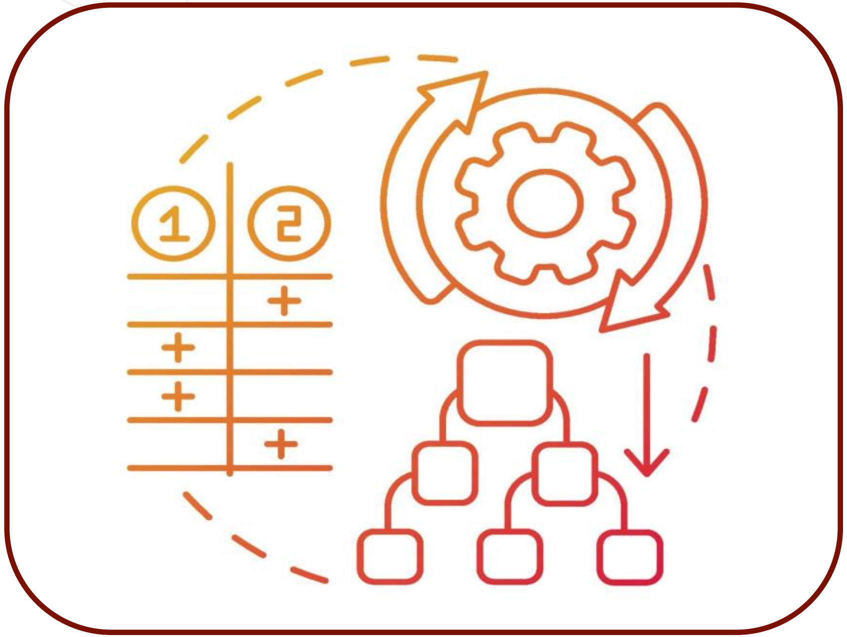


Queue:

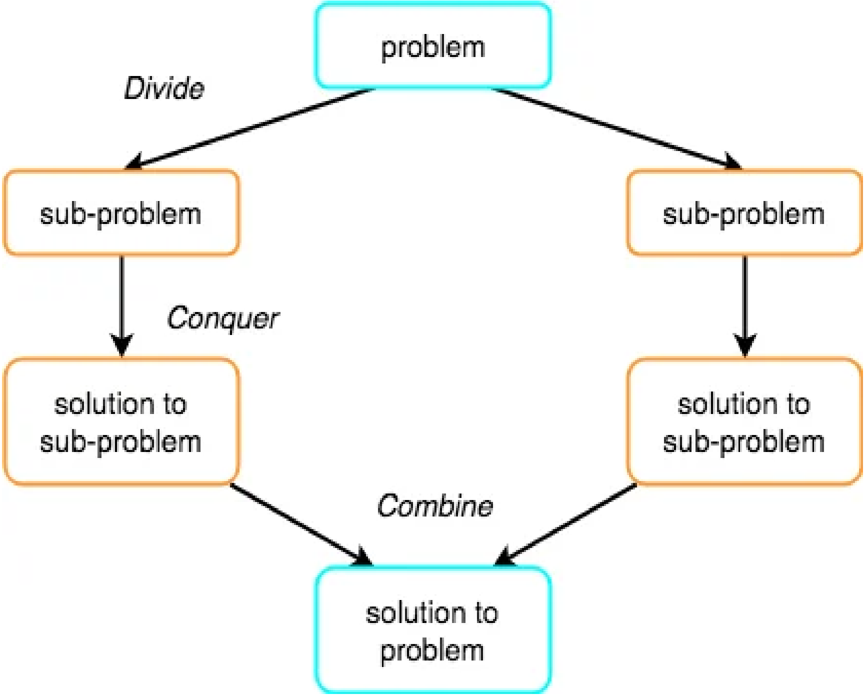
First in, first out

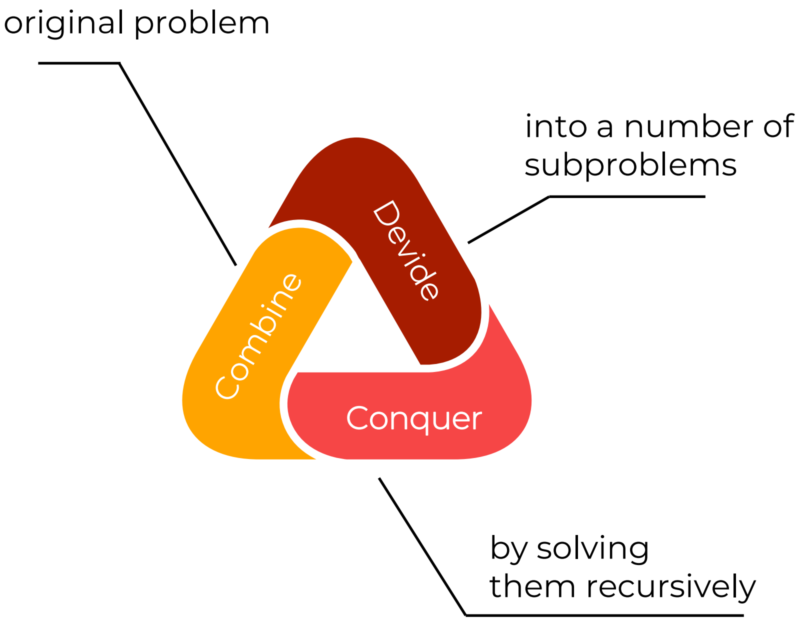


Generality Algorithm



How to solve problems by the divide and conquer algorithm?


problem
Divide
sub-problem
sub-problem
Conquer
solution to sub-problem
solution to sub-problem
Combine
solution to problem


original problem
into a number of subproblems
Devide
Combine
Conquer
by solving them recursively

$T(n) = aT(n/b) + f(n)$

- **n** = size of input
- **a** = number of subproblems in the recursion
- **n/b** = size of each subproblem.
- **f(n)** = cost of the work done outside the recursive call, which includes the cost of dividing the problem and cost of merging the solutions.

3.

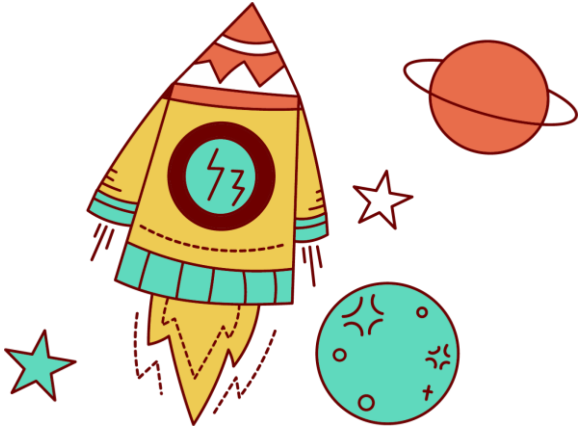
Problems

Divide and conquer algorithm

Pseudocode

```
# P(proplem | size: n)

DAC (P)
{
    if( small(P) ){
        Solve( (P) );
    }
    else{
        Divide P into  P1,P2,P3,...Pk;
        Apply DAC( P1 ), DAC( P2 ),...;
        Combine ( DAC( P1 ), DAC( P2 ),...)
    }
}
```



Problems

Suitable case:

Given a sorted array Arr[] of n elements, write a function to search a given element x in Arr[].

=> within a Sorted array.
=> Binary search runs in logatithmic time in the worst case.

Pseudocode:

```
function binary_search(A, n, T) is
    L := 0
    R := n − 1
    while L ≤ R do
        m := floor((L + R) / 2)
        if A[m] < T then
            L := m + 1
        else if A[m] > T then
            R := m − 1
        else:
            return m
    return unsuccessful
```

Problems

Unsuitable case:

Given a sorted array Arr[] of n elements, write a function to calculate sum n elements.

Time complexity:

D-A-C:

```
total run-time: 601.881504 ms
Sum = 4998100215846
```

Others:

```
total run-time: 185.421944 ms
Sum = 4998100215846
```

Pseudocode:

```
def sum(a, L,  R):
        if (L == R): return a[L]
        m = (L + R) // 2
        sumL = sum(a, L, m)
        sumR = sum(a, m + 1, R)

        return sumL + sumR
```

4.

Conclusion

Divide and conquer algorithm

O(n.log(n))

Pos:

- Solving difficult problems: A powerful tool for solving conceptually dificult problems.
- Algorithm efficiency: Offen helps in the discovery of efficient algorithms.
- Memory access: Naturally tend to make efficient use of memory caches.

Neg:

- Divide and conquer cannot save the results through of problems resolved for the next request.

Example: Fibonacci

Divide and Conquer approach:

```
fib(n)
    If n < 2, return 1
    Else , return f(n - 1) + f(n -2)
```

Dynamic approach:

```
mem = []
fib(n)
    If n in mem: return mem[n]
    else,
        If n < 2, f = 1
        else , f = f(n - 1) + f(n -2)
        mem[n] = f
        return f
```

> *The divide and conquer method is also a way to solve when we encounter difficult tasks, dividing them into smaller tasks to do instead of doing a whole big task.*

Reference source:

- Anany Levitin, Introduction to the Design and Analysis of Algorithms, 3rd Edition, 2014
- https://www.javatpoint.com/divide-and-conquer-introduction
- https://www.geeksforgeeks.org/divide-and-conquer-algorithm-introduction/
- http://www.cs.cmu.edu/afs/cs/academic/class/15210-s15/www/lectures/dandc-notes.pdf
- https://www.geeksforgeeks.org/fundamentals-of-algorithms/#AnalysisofAlgorithms
- https://www.codechef.com/wiki/test-generation-plan

Thank you!

Report

TASK ASSIGNMENT SHEET

Name	Task	Percent
Truong Quoc Truong	Design slide, Presented the parts Introduce algorithm, Generality algorithm	100%
Nguyen Quang Tuan	Presented the parts Problems, Conclusion	100%
Nguyen Ngoc Tan	Presented the parts Algorithmic complexity, In charge of the Mini game, Homework	100%

Report

All team members do the right thing, search for full information, complete their tasks.

In the discussion chaired by the group, there were many mistakes and a lack of connection with the audience

Self-assess the level of completion of the group discussion: 70%