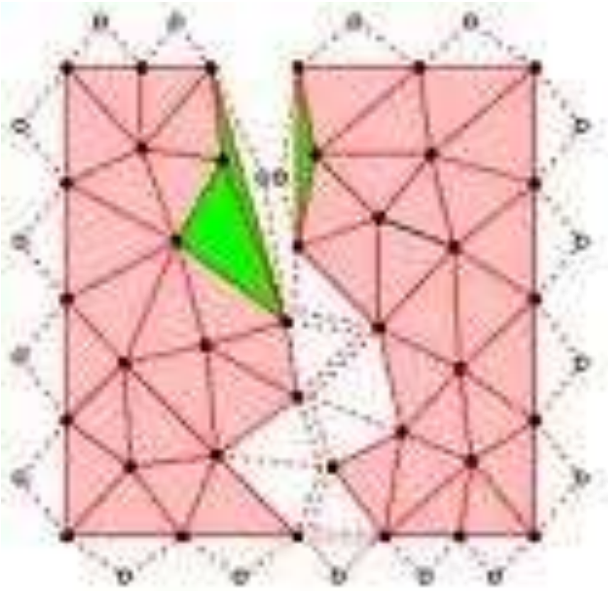
Divide & Conquer Strategy

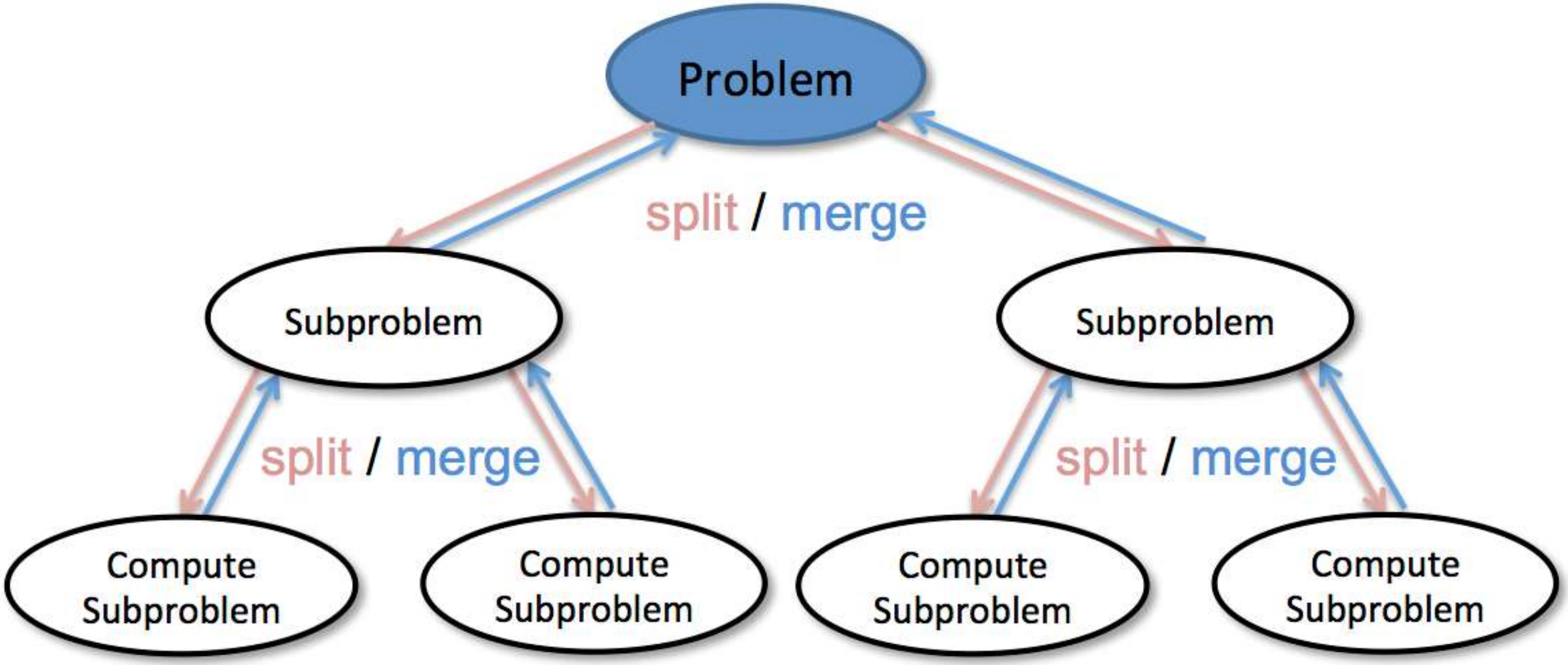
General Method, Finding the maximum and minimum,
Merge Sort, Quick Sort

Divide And Conquer Method

- General Method
- Finding the maximum and minimum
- Binary Search
- Merge Sort
- Quick Sort



General Method



General Concept of Divide & Conquer

- Given a function to compute on *n* inputs, the divide-and-conquer strategy consists of:
  - splitting the inputs into k distinct subsets, $1<k\leq n$, yielding k subproblems.
  - solving these subproblems
  - combining the subsolutions into solution of the whole.
  - if the subproblems are relatively large, then divide_Conquer is applied again.
  - if the subproblems are small, the are solved without splitting.

The Divide and Conquer Algorithm

```
Divide_Conquer(problem P)
{
  if Small(P) return S(P);
  else {

    divide P into smaller instances P1, P2, …, Pk, k≥1;

    Apply Divide_Conquer to each of these subproblems;

    return Combine(Divide_Conque(P1), Divide_Conque(P2),…,
  Divide_Conque(Pk));
  }
}
```

Divde_Conquer recurrence relation

- The computing time of Divide_Conquer is

$$T(n) = \begin{cases} g(n) & n \text{ small} \\ T(n_1) + T(n_2) + \ldots + T(n_k) + f(n) & \text{otherwise} \end{cases}$$

  - T(n) is the time for Divide_Conquer on any input size n.
  - g(n) is the time to compute the answer directly (for small inputs)
  - f(n) is the time for dividing P and combining the solutions.

Three Steps of The Divide and Conquer Approach

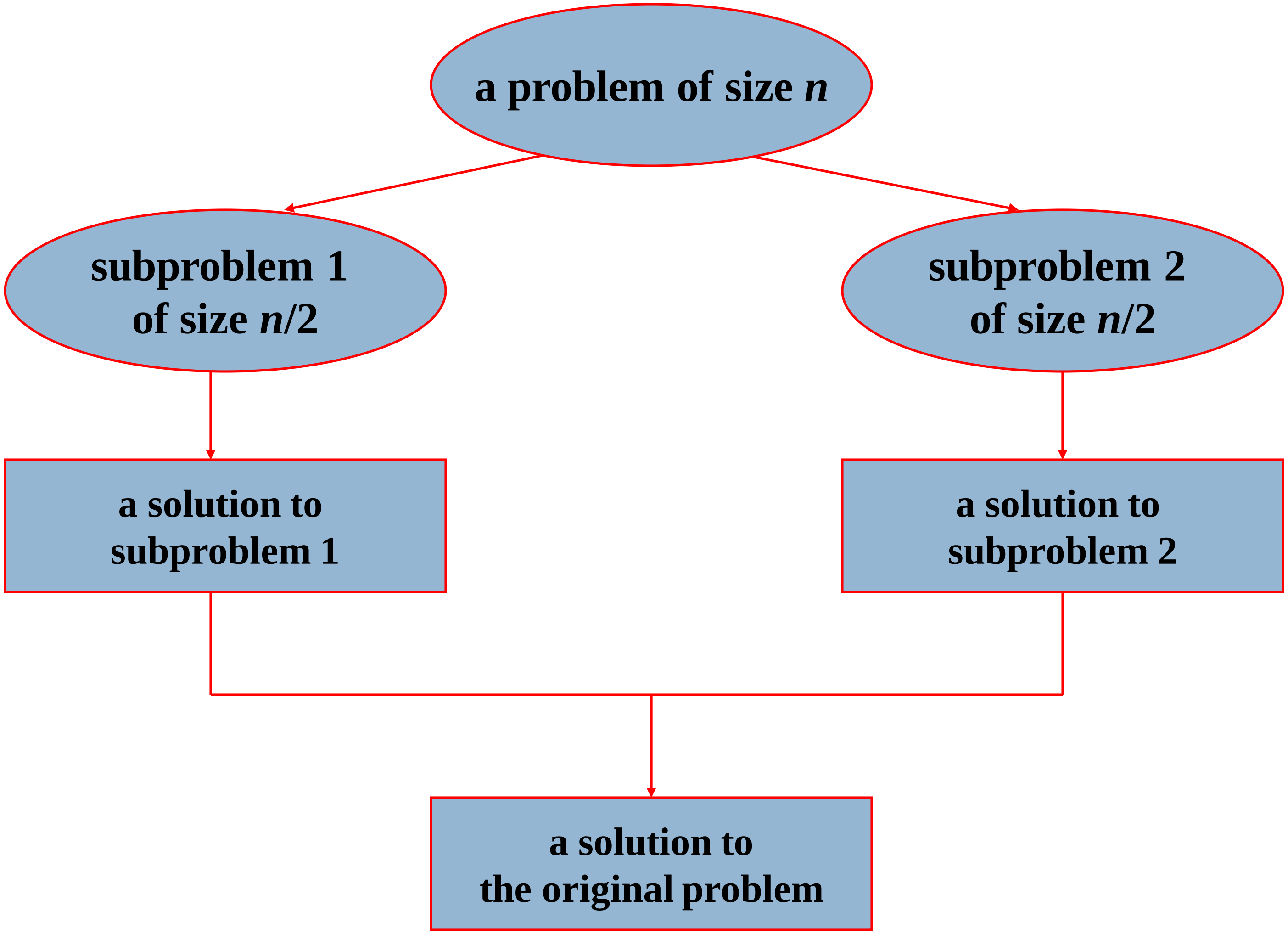
The most well known algorithm design strategy:

1. **Divide** the problem into two or more smaller subproblems.

2. **Conquer** the subproblems by solving them recursively.

3. **Combine** the solutions to the subproblems into the solutions for the original problem.

A Typical Divide and Conquer Case



An Example: Calculating $a_0 + a_1 + \ldots + a_{n-1}$

ALGORITHM RecursiveSum(A[0..n-1])

//Input: An array A[0..n-1] of orderable elements

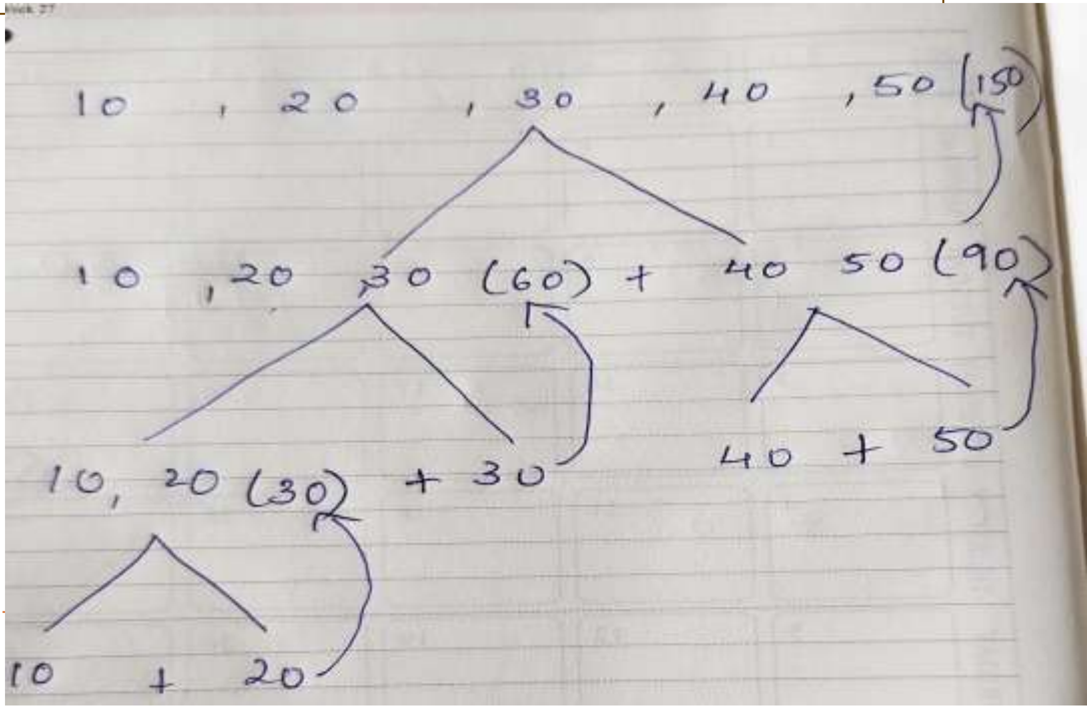
//Output: the summation of the array elements

if n > 1

    return (RecursiveSum($A[0..\lfloor n/2 \rfloor - 1]$) + RecursiveSum($A[\lfloor n/2 \rfloor .. n-1]$))

Efficiency: (for $n = 2^k$)

- $A(n) = 2A(n/2) + 1, n > 1$
- $A(1) = 1;$



$A(n) = 2A(n/2) + 1, n > 1$

$A(1) = 1;$

$$A(n) = 2A(n/2) + 1$$

$$= 2[2A(n/4)+1] + 1$$

$$= 4A(n/4)+2 + 1$$

$$= 4[2A(n/8)+1]+2 + 1$$

$$= 8A(n/8)+4+2 + 1$$

$$= 2^3A(n/2^3)+2^2+2 + 1$$

$$= 2^3A(n/2^3)+2^2+2 + 1 \ldots$$

$$= 2^kA(n/2^k)+2^{k-1}+\ldots+2 + 1$$

for $A(1)$, $n/2^k=1$ => $n= 2^k$

$$= 2^k+2^{k-1}+\ldots+2 + 1 \qquad 2^k+2^{k-1}+\ldots+2 + 1= 2^{k+1}-1$$

$$= 2^{k+1} -1 = 2^k .2(-1)=\mathbf{2n-1=O(n)}$$

Solving Recurrence Equations

- Substitution method
  - Forward
    - Uses initial condition in the initial term and generates next term.
    - Repeated until some formula is guessed
  - Backward
    - Values are substituted recursively to derive some formula

- Master's Method

Recurrence Relation

- Equation that defines a sequence recursively
- Eg1: $T(n)=T(n-1)+n, n>0$ ⟹ Recurrence relation

  $T(0)=0$ ⟹ Initial Condition

- Eg2: $f(n)= 2f(n-1)+1$ for $n>1$

  $f(0)=0$, Solve the recurrence relation

Substitution Method

- $T(n)=T(n-1)+n, n>0$ *Recurrence relation*

  $T(0)=0, n<=0$ *Initial Condition*

Forward Substitution

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

$T(0)=0$

$T(1)=T(0)+1= 0+1$

$T(2)=T(1)+2= 0+1+2$

$T(3)=T(2)+3= 0+1+2+3$ …

$\mathbf{T(n)=T(n-1)+n= 0+1+2+3+\ldots n}$

$T(n)=0+1+2+3+\ldots n=n*(n+1)/2=\mathbf{O(n^2)}$

Backward Substitution

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

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

$=[T(n-3)+n-2]+(n-1)]+n$ …

$=[T(n-k)+n-(k-1)]+\ldots(n-2)+(n-1)+n$

If it terminates at the kth step i.e n-k=0

$=T(n-k)+n-(k-1)+\ldots(n-2)+(n-1)+n$

$=T(0)+1+2+\ldots(n-2)+(n-1)+n$

$=0+1+2+\ldots(n-2)+(n-1)+n$

$=n*(n+1)/2= \mathbf{O(n^2)}$

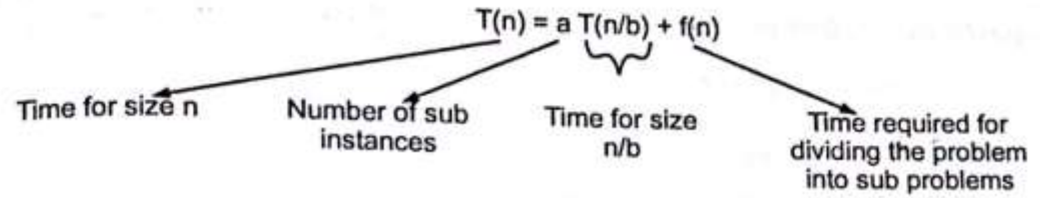
The Master Method

- The master method is used to solve recurrences of the type T(n)=aT(n/b)+f(n) where a>=1 and b>1
- The complexity of the divide and conquer algorithms is given by recurrences of the form
  - T(n)={T(1)   n=1

    {aT(n/b) + f(n)   n>1

    Where a and b are constants

    We assume that T(1) is known and n is a power of b that is n=b^k.



General Divide and Conquer recurrence

The Master Theorem

the time spent on solving a subproblem of size n/b.

$$T(n) = aT(n/b) + f(n), \quad \text{where } f(n) \in \Theta(n^k)$$

Master Theorem can be stated for efficiency analysis as:

1. $a < b^k \qquad T(n) \in \Theta(n^k)$
2. $a = b^k \qquad T(n) \in \Theta(n^k \lg n)$
3. $a > b^k \qquad T(n) \in \Theta(n^{\log_b a})$

the time spent on dividing the problem

into smaller ones and combining their solutions.

Solve the recurrence relation: T(n)=4T(n/2)+n