DESIGN AND ANALYSIS OF ALGORITHMS (DAA)

Radhika Chapaneri

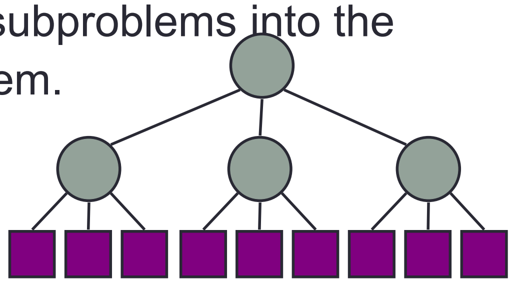
Divide and Conquer Approach

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. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.

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



Control Abstraction of Divide and Conquer

```
Algorithm DAndC(P)
{
    if Small(P) then return S(P);
    else
    {
        divide P into smaller instances P_1, P_2, ..., P_k, k ≥ 1;
        Apply DAndC to each of these subproblems;
        return Combine(DAndC(P_1),DAndC(P_2),...,DAndC(P_k));
    }
}
```

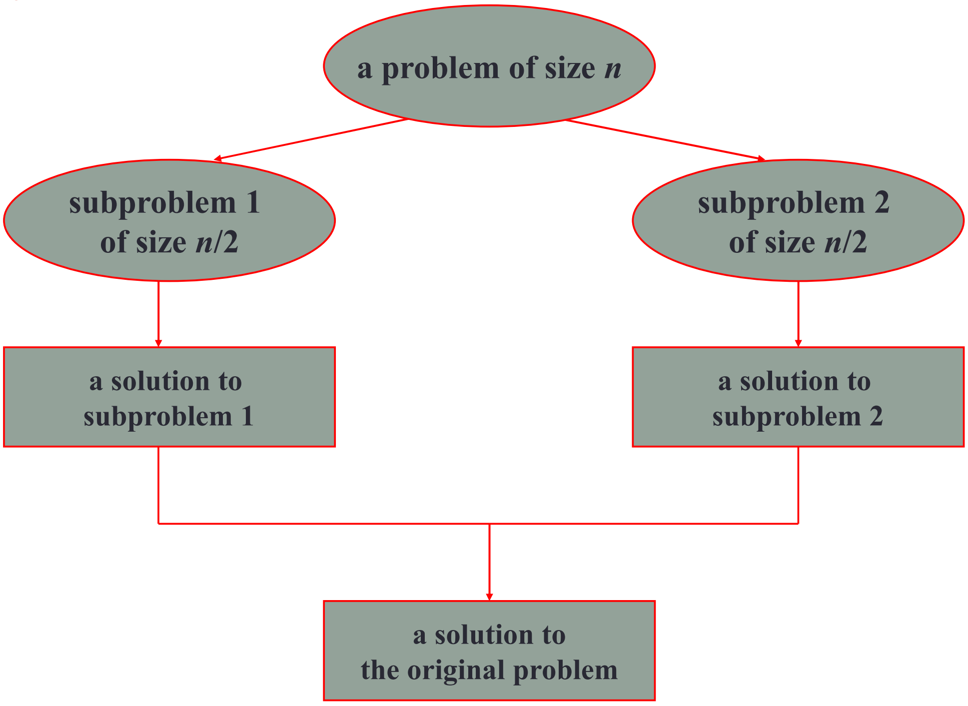
Analysis of Divide and Conquer

- When the subproblems are large enough to solve recursively, we call that the ***recursive case***. Once the subproblems become small enough that we no longer recurse, we say that the recursion "bottoms out" and that we have gotten down to the ***base case***.

- When an algorithm contains a recursive call to itself, we can often describe its running time by a ***recurrence equation*** or ***recurrence***, which describes the overall running time on a problem of size n in terms of the running time on smaller inputs.

- We can then use mathematical tools to solve the recurrence and provide bounds on the performance of the algorithm.

A Typical Divide and Conquer Case



Min Max Algorithm

- Problem: The problem is to find the maximum and minimum items in a set of n elements
-

Straight forward Max-Min

- Algorithm for straight forward maximum and minimum
- StraightMaxMin(a,n,max,min)
- // set max to the maximum
- and min to the minimum of a[1:n].
- {
- max := min := a[1];
- for i := 2 to n do
- {
- if(a[i] > max) then max := a[i];
- if(a[i] < min) then min := a[i];
- }
- }

Divide-and-Conquer approach

- – For n≤ 2, make 1 comparison
- – For large n, divide set into two smaller sets and determine largest/smallest element for each set
- – Compare largest/smallest from two subsets to determine smallest/largest of combined sets
- –Do recursively

Divide-and-Conquer approach

```
MaxMin(a[ ],i, j, max, min)
{
    if (i = j) then max := min := a[i]; //Small(P)
    else if (i=j-1) then // Another case of Small(P)
        {
            if (a[i] < a[j]) then
               max := a[j]; min := a[i];
            else max := a[i]; min := a[j];
        }
```

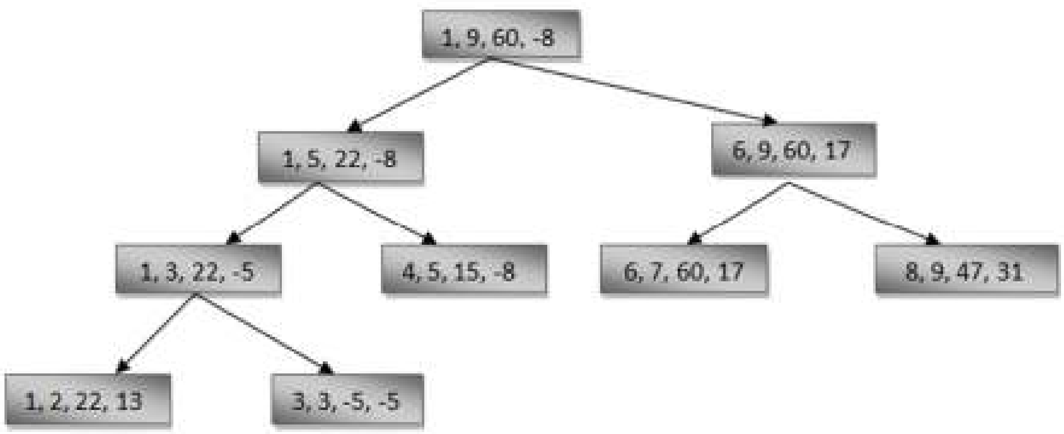
Divide-and-Conquer approach

```
else
    {
        mid := ( i + j )/2;
        MaxMin( i, mid, max, min );
        MaxMin( mid+1, j, max1, min1 );
        // Combine the solutions.
        if (max < max1) then max := max1;
        if (min > min1) then min := min1;
    }
}
```

Divide-and-Conquer approach

- Find Minimum and maximum of the following elements
- 22 13 -5 -8 15 60 17 31 47

Divide-and-Conquer approach



Divide-and-Conquer approach

If T(n) represents this number, then the resulting recurrence relation is

$$T(n) = \begin{cases} 0 & n=1 \\ 1 & n=2 \\ T(n/2) + T(n/2) + 2 & n>2 \end{cases}$$

Complexity = 3n/2 – 2

Compared with the 2n – 2 comparisons for the Straight Forward method, this is a saving of 25% in comparisons

When n is a power of two, n = 2^k
-for some positive integer k, then

$$\begin{aligned} T(n) &= 2T(n/2) + 2 \\ &= 2(2T(n/4) + 2) + 2 \\ &= 4T(n/4) + 4 + 2 \\ &\;. \\ &\;. \\ &\;. \\ &= 2^{k-1}\,T(2) + \textstyle\sum(1\le i\le k-1)\, 2^k \\ &= 2^{k-1} + 2^k - 2 \\ &= 3n/2 - 2 = O(n) \end{aligned}$$

Note that 3n/2 – 2 is the best, average, worst case number of comparison when n is a power of two.