Design and Analysis of Algorithm

Lecture-4:

Contents

Divide and Conquer

Definition

Divide:

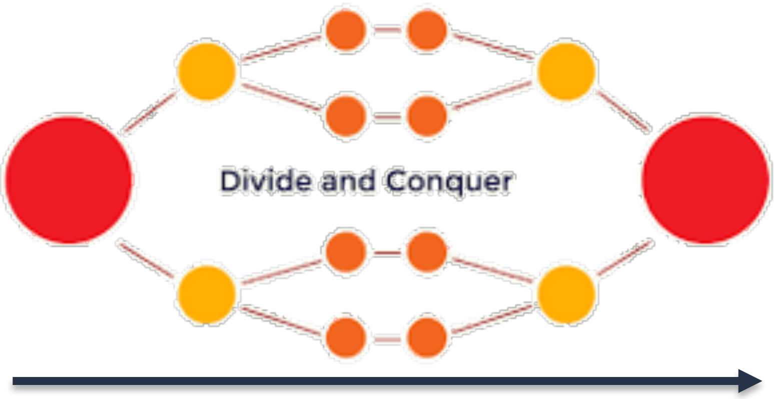
Divide the given problem into sub-problem

Conquer:

Conquer the sub-problem to get the solution recursively. If the sub-problem is small directly return the solution

Combine:

Combine the sub-problem solution to get original problem solution



General Concept

```
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));
    }
}
```

$$T(n) = \begin{cases} c & n = 1 \\ a\,T\left(\frac{n}{b}\right) + f(n) & n > 1 \end{cases}$$

$$T(n) = \begin{cases} c & if \;\; n \; is \;\; small \\ T(n_1) + T(n_2) + T(n_3) + \cdots \ldots \ldots \ldots \ldots T(n_k) + f(n) & otherwise \end{cases}$$

Approach of divide and conquer

$$n \Rightarrow small \Rightarrow c$$

$$\begin{tikzcd}
 & n \arrow[dl] \arrow[dr] & \\
\frac{n}{2} \arrow[dr] & & \frac{n}{2} \arrow[dl] \\
 & f(n) &
\end{tikzcd}$$

No. of elements in each sub-problems

$$T(n) = 2T(n/2) + f(n)$$

No. of sub-problems

Power of an element

Input: An element $a > 0$ and another element $n > 0$
Output: a^n

Without Divide and Conquer

```
Power(a, n)
{
  int f=1;
  for (i = 1; i ≤ n; i++)
  {
   f = f * a
  }
  return (f);
}
```

With Divide and Conquer

```
Power(a,n)
{
   If (n==1)
   {
      return (a)
    }
    else
    {
          b = n/2
          c= power(a, b)
          return (c*c)
     }
   }
}
```

Complexity of the two approach

Without Divide and conquer

Complexity = O(n)

With Divide and conquer

$$T(n) = \begin{cases} 1 & if\ n = 1 \\ T\left(\frac{n}{2}\right) + c & otherwise \end{cases}$$

$$T(n) = T\left(\frac{n}{2}\right) + c$$
$$= T\left(\frac{n}{2^2}\right) + c + c$$
$$= T\left(\frac{n}{2^3}\right) + 3c$$

$$T\left(\frac{n}{2^k}\right) + kc$$
$$T(n) = c + \mathrm{c.}\log_2 n$$
$$T(n) = O(\log_2 n)$$

Application of divide and conquer

- Finding maximum and minimum
- Binary search
- Merge sort
- Quick sort
- Selection sort
- Strassen's matrix multiplication.

Finding The Maximum And Minimum

The problem is to find the maximum and minimum Items in a set of n elements.

Input: An array of n elements
Output: Find maximum and minimum in the given array

Min: 1

e.g.

A [10,20,30,1,2,3,11,21,31]

Max: 31

Without Divide and Conquer

```
Algorithm 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];
        else if (a[i] < min) then min := a[i];
    }
}
```

Complexity

Best Case: It occurs when the numbers are in increasing order

`[10,20,30,40,50]`

No. of comparisons: $n - 1$ Complexity: $\Omega(n)$

Worst Case: It occurs when the numbers are in decreasing order

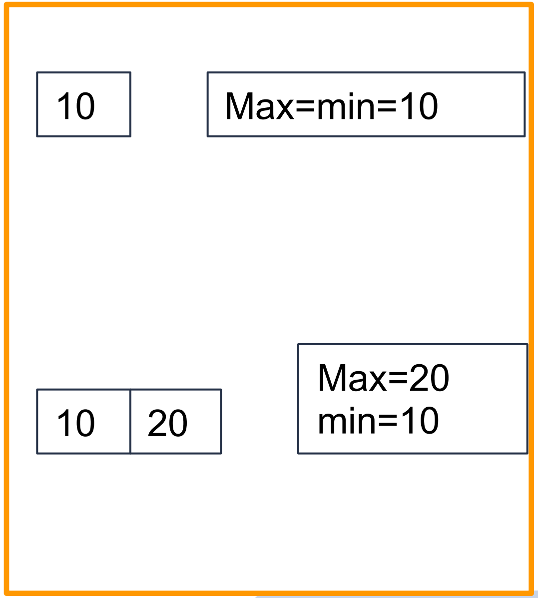
`[70,60,50,40,30]`

No. of comparisons: $2(n - 1)$ Complexity: $O(n)$

Average Case: It occurs when the numbers are in increasing order
Complexity: $\theta(n)$

With Divide and Conquer

```
Algorithm MaxMin(i, j, max, min)
// a[1 : n] is a global array. Parameters i and j are integers,
// 1 ≤ i ≤ j ≤ n. The effect is to set max and min to the
// largest and smallest values in a[i : j], respectively.
{
    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];
            }
    }
```



With Divide and Conquer

{ // If P is not small, divide P into subproblems.
// Find where to split the set.
$mid := \lfloor (i+j)/2 \rfloor$;
// Solve the subproblems.
MaxMin(i, mid, max, min);
MaxMin($mid+1, j, max1, min1$);
// Combine the solutions.
if $(max < max1)$ **then** $max := max1$;
if $(min > min1)$ **then** $min := min1$;
}

$$A\,[10, 20, 30, 1, 2, 3, 11, 21, 31]$$