Divide and Conquer Algorithm | Introduction

In this article, we are going to discuss how Divide and Conquer technique is helpful and how we can solve the problem with the DAC technique approach. In this section, we will discuss as the following topics.

```
1. Introduction to DAC.
2. Algorithms under DAC techniques.
3. Recurrence Relation for DAC algorithm.
4. Problems using DAC technique.
```

Divide And Conquer

This technique can be divided into the following three parts:

1. **Divide:** This involves dividing the problem into some sub problem.
2. **Conquer:** Sub problem by calling recursively until sub problem solved.
3. **Combine:** The Sub problem Solved so that we will get find problem solution.

The following are some standard algorithms that follows Divide and Conquer algorithm.

1. **Binary Search** is a searching algorithm. In each step, the algorithm compares the input element x with the value of the middle element in array. If the values match, return the index of the middle. Otherwise, if x is less than the middle element, then the algorithm recurs for left side of middle element, else recurs for the right side of the middle element.
2. **Quicksort** is a sorting algorithm. The algorithm picks a pivot element, rearranges the array elements in such a way that all elements smaller than the picked pivot element move to left side of pivot, and all greater elements move to right side. Finally, the algorithm recursively sorts the subarrays on left and right of pivot element.
3. **Merge Sort** is also a sorting algorithm. The algorithm divides the array in two halves, recursively sorts them and finally merges the two sorted halves.
4. **Closest Pair of Points** The problem is to find the closest pair of points in a set of points in x-y plane. The problem can be solved in O(n^2) time by calculating distances of every pair of points and comparing the distances to find the minimum. The Divide and Conquer algorithm solves the problem in O(nLogn) time.
5. **Strassen's Algorithm** is an efficient algorithm to multiply two matrices. A simple method to multiply two matrices need 3 nested loops and is O(n^3). Strassen's algorithm multiplies two matrices in O(n^2.8974) time.
6. **Cooley–Tukey Fast Fourier Transform (FFT) algorithm** is the most common algorithm for FFT. It is a divide and conquer algorithm which works in O(nlogn) time.
7. **Karatsuba algorithm for fast multiplication** it does multiplication of two *n*-digit numbers in at most single-digit multiplications in general (and exactly $n^{\log_2 3}$ when *n* is a power of 2). It is therefore faster than the classical algorithm, which requires n^2 single-digit products. If $n = 2^{10} = 1024$, in particular, the exact counts are $3^{10} = 59,049$ and $(2^{10})^2 = 1,048,576$, respectively.

Divide And Conquer algorithm :

```
DAC(a, i, j)
{
    if(small(a, i, j))
      return(Solution(a, i, j))
    else
      m = divide(a, i, j)               // f1(n)
      b = DAC(a, i, mid)                 // T(n/2)
      c = DAC(a, mid+1, j)            // T(n/2)
      d = combine(b, c)                 // f2(n)
   return(d)
}
```

Recurrence Relation for DAC algorithm :

This is recurrence relation for above program.

```
            O(1) if n is small
T(n) =      f1(n) + 2T(n/2) + f2(n)
```

Example:

To find the maximum and minimum element in a given array.

```
Input: { 70, 250, 50, 80, 140, 12, 14 }
Output: The minimum number in a given array is : 12
The maximum number in a given array is : 250
```

Approach: To find the maximum and minimum element from a given array is an application for divide and conquer. In this problem, we will find the maximum and minimum elements in a given array. In this problem, we are using a divide and conquer approach(DAC) which has three steps divide, conquer and combine.

- **For Maximum:**
  In this problem, we are using the recursive approach to find maximum where we will see that only two elements are left and then we can easily using condition i.e. if(a[index]>a[index+1].)
  In a program line a[index] and a[index+1])condition will ensure only two elements in left.

```
if(index >= l-2)
{
if(a[index]>a[index+1])
{
// (a[index]
// Now, we can say that the last element will be maximum in a given array.
}
else{
//(a[index+1]
// Now, we can say that last element will be maximum in a given array.
} }
```

In the above condition, we have checked the left side condition to find out the maximum. Now, we will see the right side condition to find the maximum.
Recursive function to check the right side at the current index of an array.

```
max = DAC_Max(a, index+1, l);
// Recursive call
```

Now, we will compare the condition and check the right side at the current index of a given array.
In the given program, we are going to implement this logic to check the condition on the right side at the current index.

```
// Right element will be maximum.
if(a[index]>max)
return a[index];

// max will be maximum element in a given array.
else
return max;
}
```

- **For Minimum:**
In this problem, we are going to implement the recursive approach to find the minimum no. in a given array.

```
int DAC_Min(int a[], int index, int l)
//Recursive call function to find the minimum no. in a given array.

if(index >= l-2)
// to check the condition that there will be two-element in the left
then we can easily find the minimum element in a given array.
{
// here we will check the condition
if(a[index]<a[index+1])
return a[index];
else
return a[index+1];
}
```

Now, we will check the condition on the right side in a given array.

```
// Recursive call for the right side in the given array.
min = DAC_Min(a, index+1, l);
```

Now, we will check the condition to find the minimum on the right side.

```
// Right element will be minimum
if(a[index]<min)
return a[index];
```

```
// Here min will be minimum in a given array.
else
return min;
```

Implementation:

```
// Code to demonstrate Divide and
// Conquer Algorithm
#include <stdio.h>
int DAC_Max(int a[], int index, int l);
int DAC_Min(int a[], int index, int l);

// function to find the maximum no.
// in a given array.
int DAC_Max(int a[], int index, int l)
{
    int max;
    if (index >= l - 2) {
        if (a[index] > a[index + 1])
            return a[index];
        else
            return a[index + 1];
    }

    // logic to find the Maximum element
    // in the given array.
    max = DAC_Max(a, index + 1, l);

    if (a[index] > max)
        return a[index];
    else
        return max;
}

// Function to find the minimum no.
// in a given array.
int DAC_Min(int a[], int index, int l)
{
    int min;
    if (index >= l - 2) {
        if (a[index] < a[index + 1])
            return a[index];
        else
            return a[index + 1];
    }

    // Logic to find the Minimum element
    // in the given array.
    min = DAC_Min(a, index + 1, l);

    if (a[index] < min)
        return a[index];
    else
        return min;
}
```

```
// Driver Code
int main()
{
    // Defining the variables
    int min, max, N;

    // Initializing the array
    int a[7] = { 70, 250, 50, 80, 140, 12, 14 };

    // recursion - DAC_Max function called
    max = DAC_Max(a, 0, 7);

    // recursion - DAC_Max function called
    min = DAC_Min(a, 0, 7);
    printf("The minimum number in a given array is : %d\n", min);
    printf("The maximum number in a given array is : %d", max);
    return 0;
}
```

Output:

```
The minimum number in a given array is : 12

The maximum number in a given array is : 250
```

Divide and Conquer (D & C) vs Dynamic Programming (DP)

Both paradigms (D & C and DP) divide the given problem into subproblems and solve subproblems. How to choose one of them for a given problem? Divide and Conquer should be used when same subproblems are not evaluated many times. Otherwise Dynamic Programming or Memoization should be used. For example, Binary Search is a Divide and Conquer algorithm, we never evaluate the same subproblems again. On the other hand, for calculating nth Fibonacci number, Dynamic Programming should be preferred (See this for details).