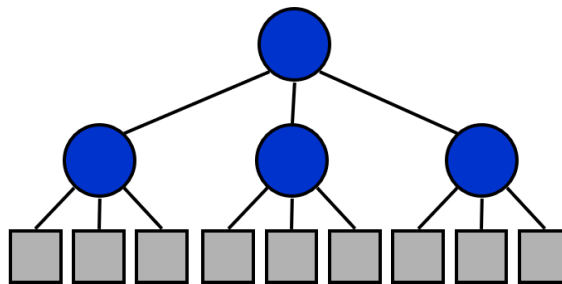


In computer science, divide and conquer is an algorithm design paradigm. A divide-and-conquer algorithm recursively breaks down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

### Divide and conquer steps:

1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem
2. **Conquer** the subproblems by solving them recursively
3. **Combine** the solutions to the subproblems into the solution for the original problem
4. The **base case** for the recursion is subproblems of constant size



### Practice problems:

#### Instructions:

1. Do not adopt unfair means. **10 marks will be deducted from the final marks for adopting unfair means.**
2. No more than 40% marks for uncompileable codes.

#### 1. Find the max and min element of an array.

##### For loop version:

```

1  Function MaxMin(A) :
2      fmax ← fmin ← A (1);
3      for i ← 2 to n do
4          if (A (i) > fmax) then fmax ← A (i);
5          if (A (i) < fmin) then fmin ← A (i);
6      return fmax, fmin
  
```

**Divide and Conquer version:**

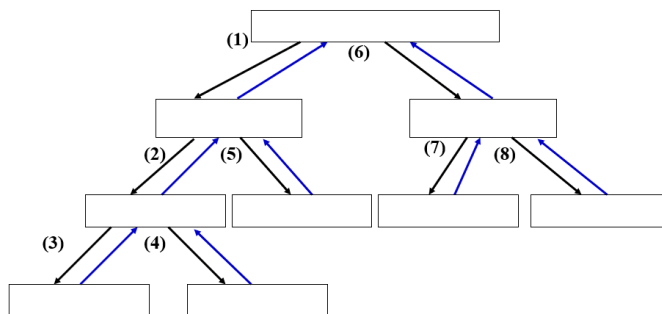
```

1  Function RMaxMin(A, i, j):
2      if i==j:
3          return A[i], A[i]
4      else
5          mid ← (i+j)/2
6          gmax, gmin ← RMaxMin(A, i, mid)
7          hmax, hmin ← RMaxMin(A, mid+1, j)
8          fmax ← max(gmax, hmax)
9          fmin ← min(gmin, hmin)
10         return fmax, fmin

```

**Recursion Tree:**

Index: 1 2 3 4 5 6 7 8 9  
 Array: 22 13 -5 -8 15 60 17 31 47

**2.  $X^Y$** **Hint:**

$$3^{100} = 3^{50} \cdot 3^{50} = \left(3^{\frac{100}{2}}\right) \cdot \left(3^{\frac{100}{2}}\right)$$

$$3^{50} = 3^{25} \cdot 3^{25}$$

$$3^{25} = 3^{12} \cdot 3^{12} \cdot 3$$

$$3^{12} = 3^6 \cdot 3^6$$

$$3^6 = 3^3 \cdot 3^3$$

$$3^3 = 3^1 \cdot 3^1 \cdot 3$$

$$3^1 = 3^0 \cdot 3^0 \cdot 3$$

### 3. Merge sort

<pre> MERGE-SORT(<math>A, p, r</math>) 1  <b>if</b> <math>p &lt; r</math> 2    <b>then</b> <math>q \leftarrow \lfloor (p + r)/2 \rfloor</math> 3          MERGE-SORT(<math>A, p, q</math>) 4          MERGE-SORT(<math>A, q + 1, r</math>) 5          MERGE(<math>A, p, q, r</math>) </pre>	<pre> MERGE(<math>A, p, q, r</math>) 1  <math>n_1 \leftarrow q - p + 1</math> 2  <math>n_2 \leftarrow r - q</math> 3  create arrays <math>L[1 \dots n_1 + 1]</math> and <math>R[1 \dots n_2 + 1]</math> 4  <b>for</b> <math>i \leftarrow 1</math> <b>to</b> <math>n_1</math> 5    <b>do</b> <math>L[i] \leftarrow A[p + i - 1]</math> 6  <b>for</b> <math>j \leftarrow 1</math> <b>to</b> <math>n_2</math> 7    <b>do</b> <math>R[j] \leftarrow A[q + j]</math> 8  <math>L[n_1 + 1] \leftarrow \infty</math> 9  <math>R[n_2 + 1] \leftarrow \infty</math> 10 <math>i \leftarrow 1</math> 11 <math>j \leftarrow 1</math> 12 <b>for</b> <math>k \leftarrow p</math> <b>to</b> <math>r</math> 13   <b>do if</b> <math>L[i] \leq R[j]</math> 14     <b>then</b> <math>A[k] \leftarrow L[i]</math> 15           <math>i \leftarrow i + 1</math> 16     <b>else</b> <math>A[k] \leftarrow R[j]</math> 17           <math>j \leftarrow j + 1</math> </pre>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

### 4. Count Inversion

<https://www.cp.eng.chula.ac.th/~prabhas//teaching/algo/algo2008/count-inv.htm>

The sequence 2, 4, 1, 3, 5 has three inversions (2,1), (4,1), (4,3).

The idea is similar to "merge" in merge-sort. Merge two sorted lists into one output list, but we also count the inversion.

- divide: size of sequence  $n$  to two lists of size  $n/2$
- conquer: count recursively two lists
- combine: this is a trick part (to do it in linear time)

### 5. Quick Sort

The quick sort uses divide and conquer just like merge sort but without using additional storage.

The steps are:

1. Select an element  $q$ , called a pivot, from the array. In this algorithm we have chosen the last index as the pivot.

2. The PARTITION function finds the location of the pivot in such a way that all the elements smaller than the pivot is on the left side and all the element on the right-hand side of the pivot is greater in value. (Items with equal values can go either way).
3. Recursively call the QUICKSORT function which perform quicksort on the array on the left side of the pivot and then on the array on the right side, thus dividing the task into sub tasks. This is carried out until the arrays can no longer be split.

Implement Quick sort algorithm. The pseudo code is given below:

<b>QUICKSORT</b> ( $A, p, r$ ) 1 <b>if</b> $p < r$ 2 $q = \text{PARTITION}(A, p, r)$ 3 <b>QUICKSORT</b> ( $A, p, q - 1$ ) 4 <b>QUICKSORT</b> ( $A, q + 1, r$ )	<b>PARTITION</b> ( $A, p, r$ ) 1 $x = A[r]$ 2 $i = p - 1$ 3 <b>for</b> $j = p$ <b>to</b> $r - 1$ 4 <b>if</b> $A[j] \leq x$ 5 $i = i + 1$ 6         exchange $A[i]$ with $A[j]$ 7 exchange $A[i + 1]$ with $A[r]$ 8 <b>return</b> $i + 1$
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## 6. Binary Search

```

1  /**
2   the following function returns
3   the index of x in A; if x exists,
4   NOT_FOUND;           if x does not exist
5  */
6  function Binary_Search (A, start, _end, x)
7  {
8      if start <= _end then
9
10         mid = (start+_end)/2
11
12         if A[mid] == x then
13             return mid
14         else if A[mid] > x then
15             return Binary_Search(A, start, mid-1, x);
16         else /// if A[mid] < x
17             return Binary_Search(A, mid+1, _end, x);
18
19     return NOT_FOUND; /// base case
20 }
```

Sample Input	Sample Output
Number of Elements: 5 Enter elements: 3 4 5 7 2 Key: 4	4 found in index 1
Number of Elements: 5 Enter elements: 3 4 5 7 2 Key: 14	14 not found

7.

Write a function `print_odd` using divide-and-conquer algorithm to print the odd numbers of an array of `n` integers.

8.

Write a function `calc_sum` using divide-and-conquer algorithm to calculate the sum of an array of `n` integers.

9.

Write a function `calc_sum` using divide-and-conquer algorithm to calculate the sum of the even numbers of an array of `n` integers.

## 10. Maximum-sum subarray

```

1  Function FIND-MAXIMUM-SUBARRAY (A, low, high):
2
3      if high == low /// base case: only one element
4          return (low, high, A[low])
5      else
6          mid = (low+high)/2
7
8          left-low, left-high, left-sum = FIND-MAXIMUM-SUBARRAY(A, low, mid)
9          right-low, right-high, right-sum = FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
10         cross-low, cross-high, cross-sum = FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
11
12         if left-sum >= right-sum and left-sum >= cross-sum
13             return (left-low, left-high, left-sum)
14         else if right-sum >= left-sum and right-sum >= cross-sum
15             return (right-low, right-high, right-sum)
16         else
17             return (cross-low, cross-high, cross-sum)

```

---

```

1  Function FIND-MAX-CROSSING-SUBARRAY (A, low, mid, high):
2
3      ///Find a maximum subarray of the form A[i..mid]
4      left-sum = -∞
5      sum = 0
6      for i = mid downto low
7          sum = sum + A[i]
8          if sum > left-sum
9              left-sum = sum
10             max_left = i
11
12     ///Find a maximum subarray of the form A[mid + 1 .. j]
13     right-sum = -∞
14     sum = 0
15     for j = mid + 1 to high
16         sum = sum + A[j]
17         if sum > right-sum
18             right-sum = sum
19             max_right = j
20
21     ///Return the indices and the sum of the two subarrays
22     return (max_left, max_right, left-sum + right-sum)

```

## 11. Longest common prefix of n strings

Sample Input	Sample Output
3 Algolab Algorithms Algeria	Alg
4 Algolab Algorithms Algeria UIU	No common prefix

## 12. Closest pair of points

Ref: <https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pearson/05DivideAndConquer.pdf>

<b>Closest-Pair</b> ( $p_1, \dots, p_n$ ) {	
<b>Compute</b> separation line $L$ such that half the points are on one side and half on the other side.	$O(n \log n)$
$\delta_1 = \text{Closest-Pair}(\text{left half})$	
$\delta_2 = \text{Closest-Pair}(\text{right half})$	$2T(n/2)$
$\delta = \min(\delta_1, \delta_2)$	
<b>Delete</b> all points further than $\delta$ from separation line $L$	$O(n)$
<b>Sort</b> remaining points by y-coordinate.	$O(n \log n)$
<b>Scan</b> points in y-order and compare distance between each point and next 11 neighbors. If any of these distances is less than $\delta$ , update $\delta$ .	$O(n)$
<b>return</b> $\delta$ .	
}	

Running time:

$$T(n) \leq 2T\left(\frac{n}{2}\right) + O(n \log n) \Leftrightarrow T(n) = O(n \log^2 n)$$

Can we achieve  $O(n \log n)$ ?

Yes. Don't sort points in strip from scratch each time.

- Each recursive returns two lists: all points sorted by y coordinate, and all points sorted by x coordinate.
- Sort by merging two pre-sorted lists.
- $T(n) \leq 2T\left(\frac{n}{2}\right) + O(n) \Leftrightarrow T(n) = O(n \log n)$

## 13. More practice problems

<https://leetcode.com/tag/divide-and-conquer/>

## Reference:

- Slides of Dr. Md. Abul Kashem Mia, Professor, CSE Dept, BUET