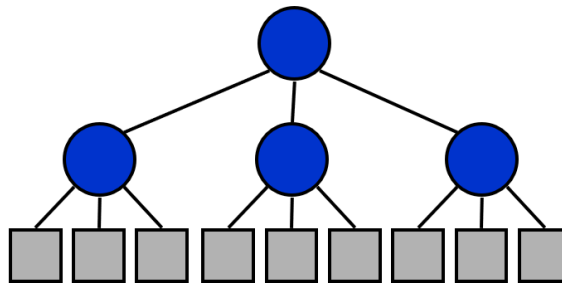


In computer science, divide and conquer is an algorithm design paradigm. A divide-and-conquer algorithm recursively breaks down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined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

```

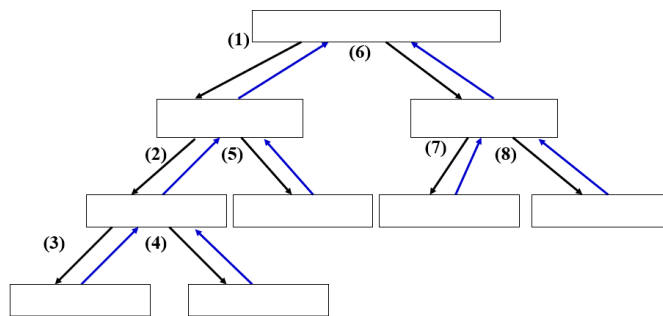
1  Function MaxMin(A) :
2      fmax ← fmin ← A (1);
3      for i ← 2 to n do
4          if (A (i) > fmax) then fmax ← A (i);
5          if (A (i) < fmin) then fmin ← A (i);
6      return fmax, fmin
  
```

```

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

```

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



2. X^Y

3. Merge sort

MERGE-SORT(A, p, r)

```

1  if  $p < r$ 
2      then  $q \leftarrow \lfloor (p+r)/2 \rfloor$ 
3          MERGE-SORT( $A, p, q$ )
4          MERGE-SORT( $A, q+1, r$ )
5          MERGE( $A, p, q, r$ )

```

MERGE(A, p, q, r)

```

1   $n_1 \leftarrow q - p + 1$ 
2   $n_2 \leftarrow r - q$ 
3  create arrays  $L[1..n_1+1]$  and  $R[1..n_2+1]$ 
4  for  $i \leftarrow 1$  to  $n_1$ 
5      do  $L[i] \leftarrow A[p+i-1]$ 
6  for  $j \leftarrow 1$  to  $n_2$ 
7      do  $R[j] \leftarrow A[q+j]$ 
8   $L[n_1+1] \leftarrow \infty$ 
9   $R[n_2+1] \leftarrow \infty$ 
10  $i \leftarrow 1$ 
11  $j \leftarrow 1$ 
12 for  $k \leftarrow p$  to  $r$ 
13     do if  $L[i] \leq R[j]$ 
14         then  $A[k] \leftarrow L[i]$ 
15              $i \leftarrow i + 1$ 
16     else  $A[k] \leftarrow R[j]$ 
17          $j \leftarrow j + 1$ 

```

Simulation in slide.

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

Practice yourself.

6. 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)

```

7. Longest common prefix of n strings:
8. Closest pair of points
9. More practice problems: <https://leetcode.com/tag/divide-and-conquer/>