دانشگاه کردستان
University of Kurdistan

Divide and Conquer

Sadoon Azizi

s.azizi@uok.ac.ir

Department of Computer Engineering and IT

Spring 2019

Techniques for the design of Algorithms

- **Divide and Conquer**
- Dynamic Programming
- Greedy Algorithms
- Backtracking Algorithms
- Branch and Bound Algorithms

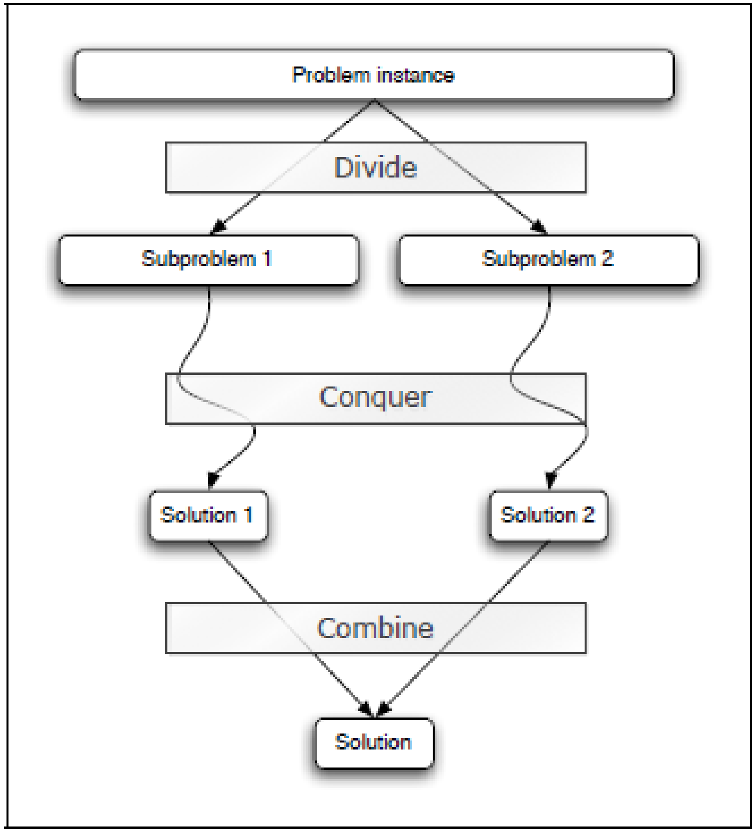
Divide and Conquer

- This approach involves three steps:

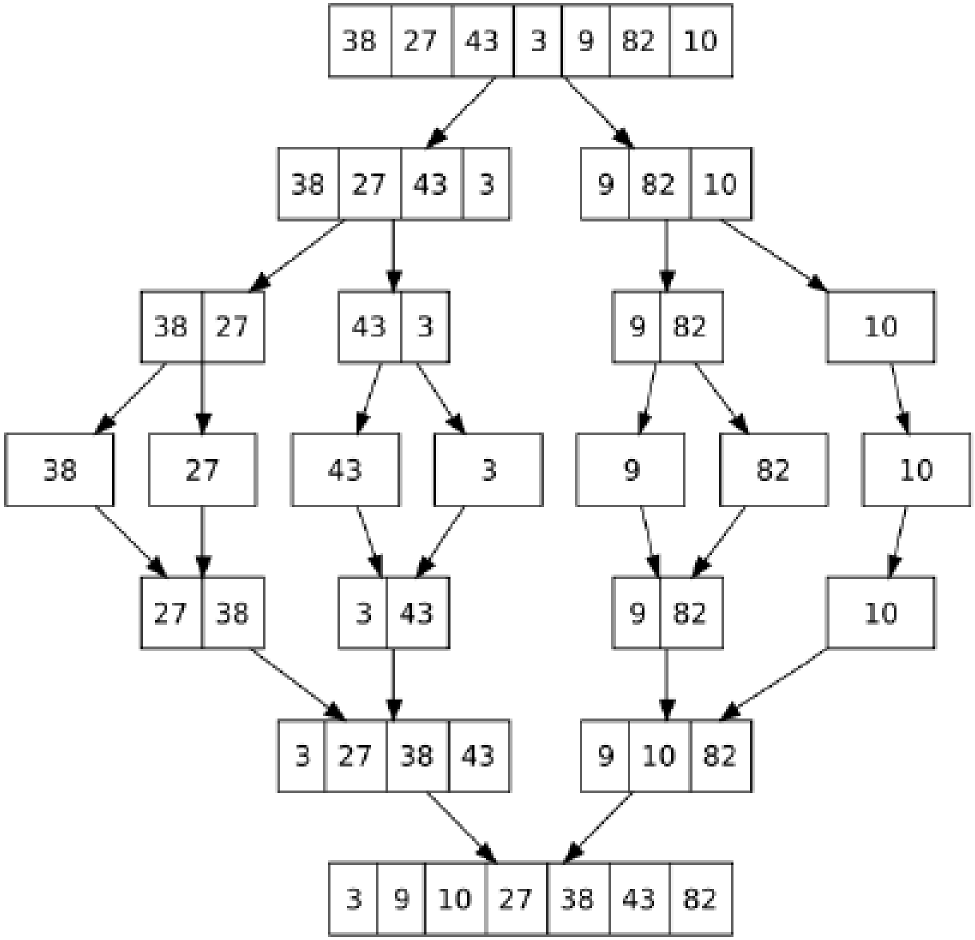
- **Divide:** Break down the problem into two or more subproblems. These subproblems should be similar to the original problem, but smaller in size.

- **Conquer:** Recursively solve the subproblems (If they are small enough, just solve them in a straightforward manner).

- **Combine:** Combine the solutions to the subproblems into a solution for the original problem (optional).



Divide and Conquer (Merge Sort)



MERGE-SORT(A, p, r)

if $p < r$

$\quad q = \lfloor (p + r)/2 \rfloor$

$\quad$ MERGE-SORT(A, p, q)

$\quad$ MERGE-SORT$(A, q + 1, r)$

$\quad$ MERGE(A, p, q, r)

Divide and Conquer (Merge Sort)

```
MERGE(A, p, q, r)
n1 = q - p + 1
n2 = r - q
let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

Analyzing the Divide-and-Conquer Algorithms

In general we have the following recurrence equation:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \le c\ , \\ aT(n/b) + D(n) + C(n) & \text{otherwise}\ . \end{cases}$$

T(n): is the time required for an input of size n

n: is the size of problem

c: is a constant number

a: is the number of subproblems

b: is the size of each subproblem

D(n): is the time needed for Divide

C(n): is the time needed for Combine

Solving the recurrence equations

There are different approaches to do this:

- Performing Substitution
- Constructing Recursion Tree
- Master Theorem

Analyzing Merge Sort by Performing Substitution

$$T(n) = \begin{cases} c & \text{if } n = 1 \,, \\ 2T(n/2) + cn & \text{if } n > 1 \,, \end{cases}$$

$$T(n) = 2T(\frac{n}{2}) + n$$

$$=2(2T(\frac{n}{4}) + \frac{n}{2}) + n = 2^2(T\left(\frac{n}{2^2}\right) + 2n$$

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

.

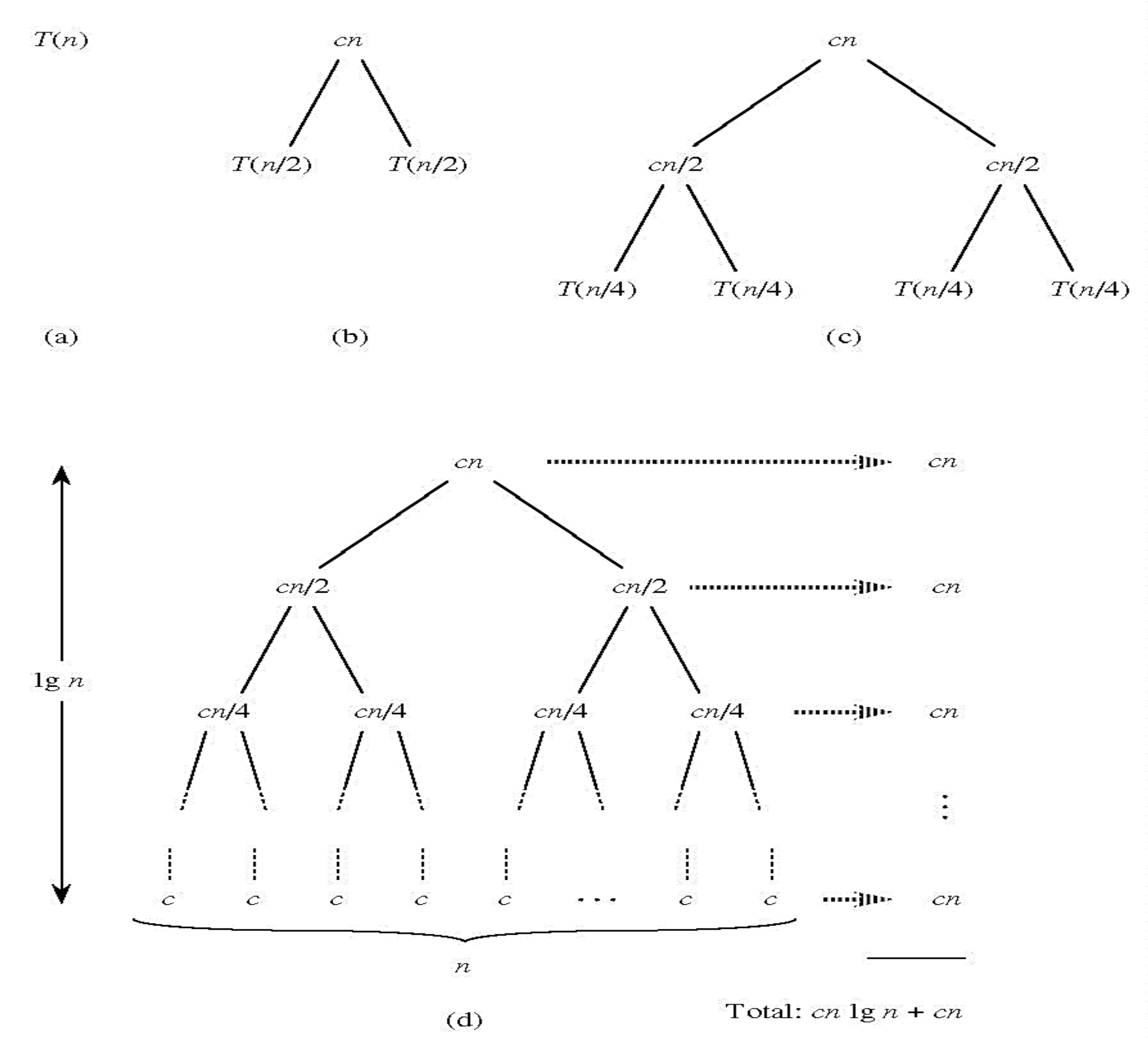
.

.

$$=2^{\log n}T(1) + n\log n = cn + n\log n$$

$$=\theta(n\log n)$$

Analyzing Merge Sort by constructing recursion tree



Master Theorem

Theorem (Master Theorem)

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n).$$

Then $T(n)$ can be bounded asymptotically as follows:

I. *If $f(n) = O(n^{\log_b(a) - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b(a)})$.*

II. *If $f(n) = \Theta(n^{\log_b(a)})$, then $T(n) = \Theta(n^{\log_b(a)} \log(n))$.*

III. *If $f(n) = O(n^{\log_b(a) + \varepsilon})$ for some constant $\varepsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.*

Analyzing Merge Sort by Master Theorem

$$T(n) = \begin{cases} c & \text{if } n = 1 \,, \\ 2T(n/2) + cn & \text{if } n > 1 \,, \end{cases}$$

Here we have, a=2, b=2, f(n)=cn

$$n^{\log_b^a} = n$$

So

$$T(n) = \theta(n\log n)$$

Maximum-subarray problem

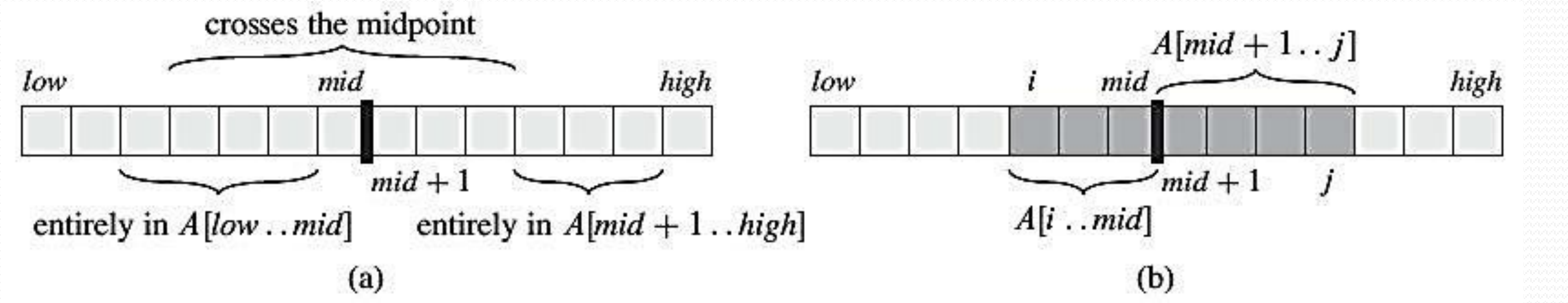
- ***Input:*** an array $A[1\ldots n]$ of n numbers
  - Assume that some of the numbers are negative, because this problem is trivial when all numbers are nonnegative
- ***Output:*** a nonempty subarray $A[i\ldots j]$ having the largest sum $S[i, j] = a_i + a_{i+1} + \ldots + a_j$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	**13**	**-3**	**-25**	**20**	**-3**	**-16**	**-23**	**18**	**20**	**-7**	**12**	**-5**	**-22**	**15**	**-4**	**7**

maximum subarray

A divide and conquer solution

- Possible locations of a maximum subarray $A[i..j]$ of $A[low..high]$, where $mid = \lfloor (low+high)/2 \rfloor$
  - entirely in $A[low..mid]$ ($low \le i \le j \le mid$)
  - entirely in $A[mid+1..high]$ ($mid < i \le j \le high$)
  - crossing the midpoint ($low \le i \le mid < j \le high$)



A divide and conquer solution

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
left-sum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > left-sum
        left-sum = sum
        max-left = i
right-sum = -∞
sum = 0
for j = mid + 1 to high
    sum = sum + A[j]
    if sum > right-sum
        right-sum = sum
        max-right = j
return (max-left, max-right, left-sum + right-sum)
```

A divide and conquer solution

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
if high == low
    return (low, high, A[low])                // base case: only one element
else mid = ⌊(low + high)/2⌋
    (left-low, left-high, left-sum) =
        FIND-MAXIMUM-SUBARRAY(A, low, mid)
    (right-low, right-high, right-sum) =
        FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
    (cross-low, cross-high, cross-sum) =
        FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
    if left-sum ≥ right-sum and left-sum ≥ cross-sum
        return (left-low, left-high, left-sum)
    elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
        return (right-low, right-high, right-sum)
    else return (cross-low, cross-high, cross-sum)
```

Analyzing time complexity

- FIND-MAX-CROSSING-SUBARRAY : $\Theta(n)$,
  where $n = high - low + 1$

- FIND-MAXIMUM-SUBARRAY
  $T(n) = 2T(n/2) + \Theta(n)$(with $T(1) = \Theta(1)$)

  $= \Theta(n\lg n)$ (similar to merge-sort)