ThS.Trần Lê Như Quỳnh

# DIVIDE AND CONQUER ALGORITHMS

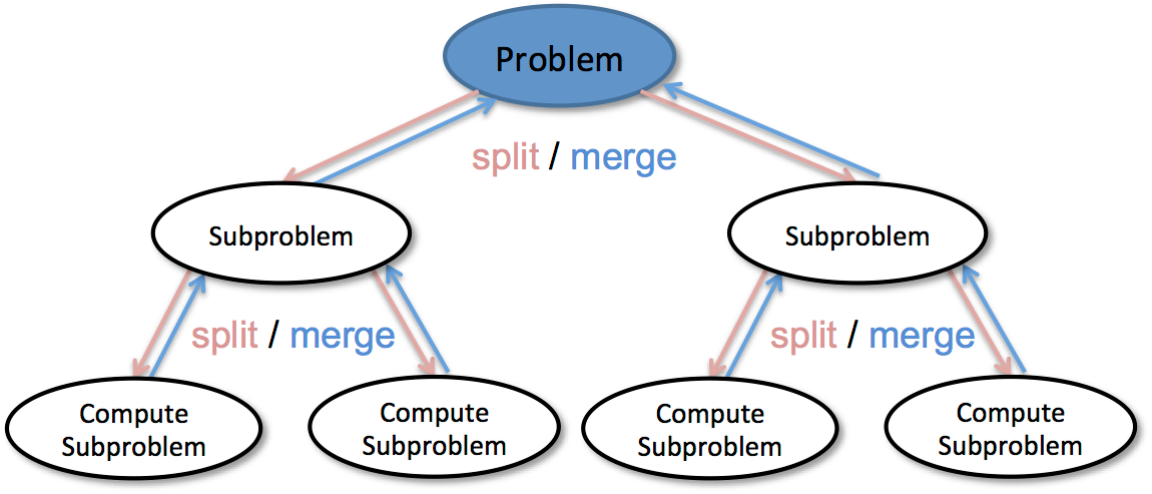
DATA STUCTURE

# DIVIDE-AND-CONQUER



ThS.Trần Lê Như Quỳnh

- Divide: If the input size is smaller than a certain threshold (say, one or two elements), solve the problem directly using a straightforward method and return the solution so obtained. Otherwise, divide the input data into two or more disjoint subsets.
- Conquer: Recursively solve the sub problems associated with the subsets.
- Combine: Take the solutions to the sub problems and merge them into a solution to the original problem



ThS.Trần Lê Như Quỳnh

# MERGE SORT

## SORTING ALGORITHMS

# DEFINITION



ThS.Trần Lê Như Quỳnh

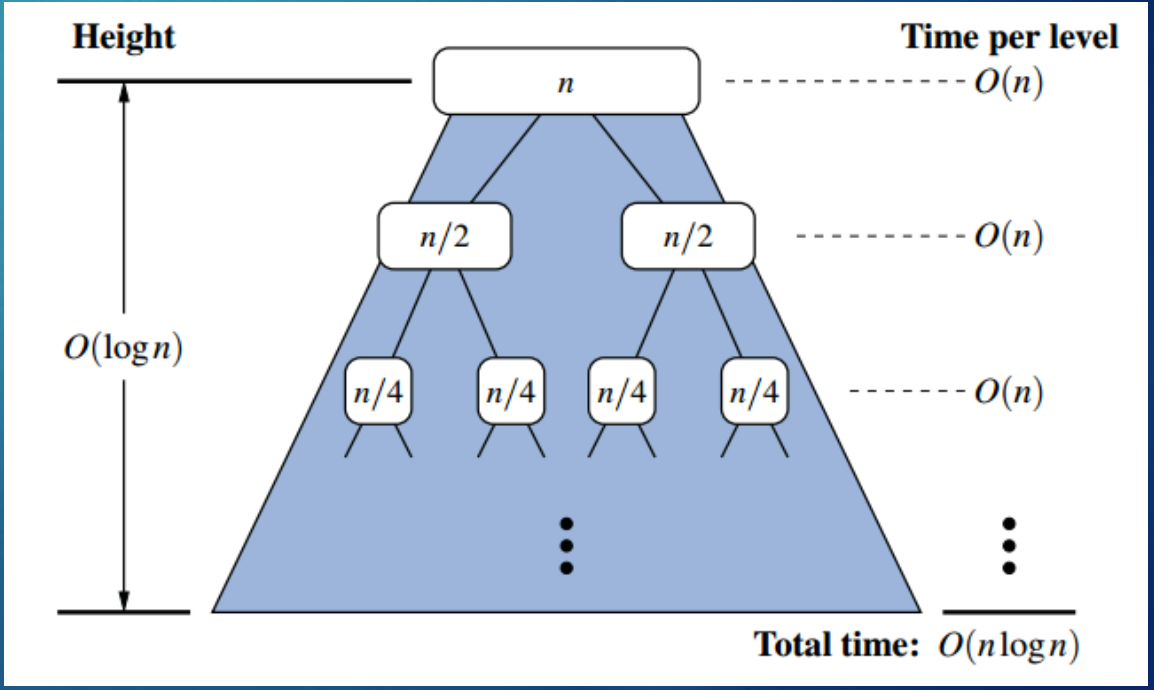
- **Divide** : First divides the n element sequence into two sub sequences having size n / 2 elements each.
- **Conquer** : Then sort the each sub sequences recursively using merge sort.
- **Combine** : Then merge the two sub sequences which are sorted to produce the sorted answer

# RUNNING TIME



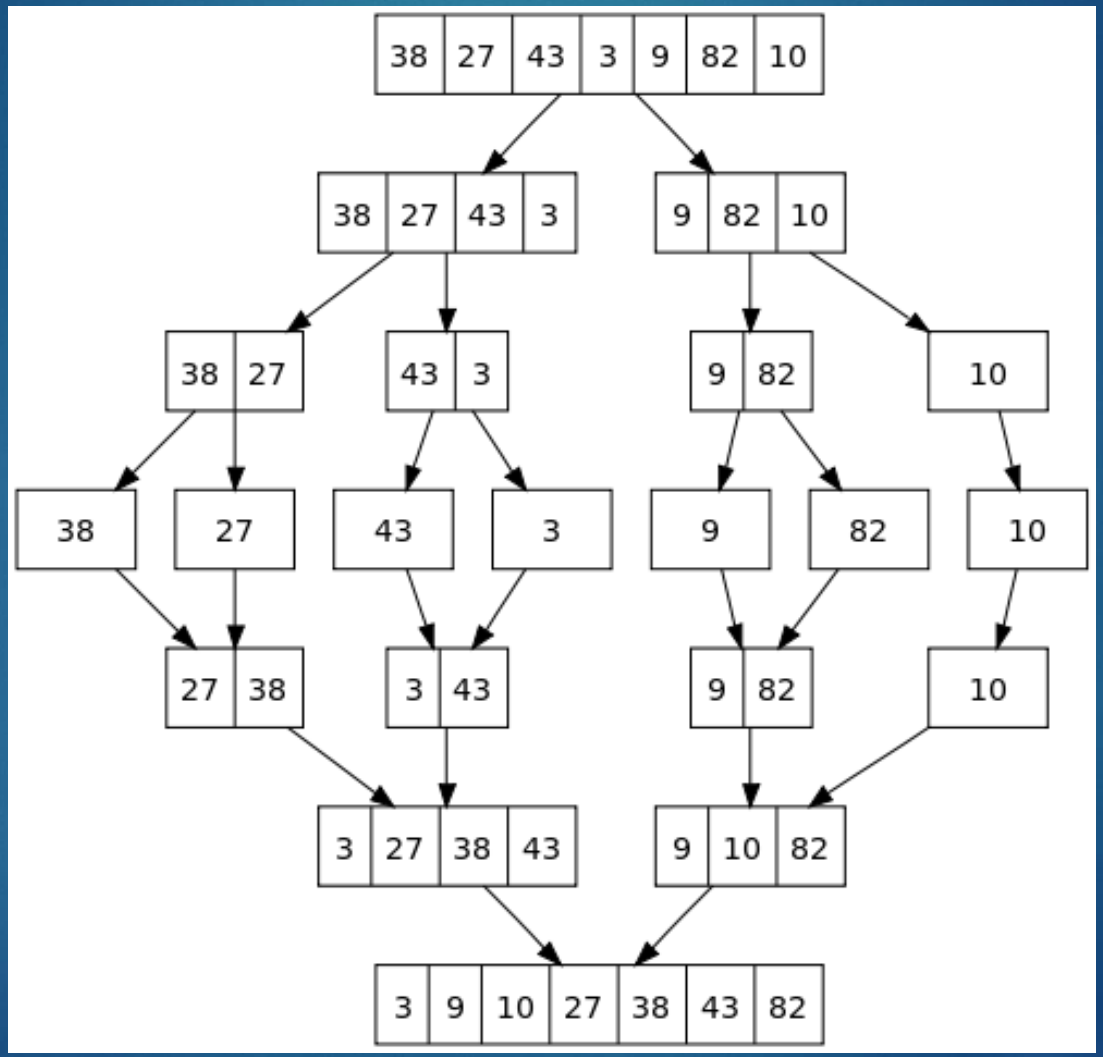
- The run time is: O(n log n).
- Reason: This algorithm splits the items to be sorted into 2 groups, recursively sorts each group, and merges them into a final sorted array. The Run time is O(n log n).

ThS.Trần Lê Như Quỳnh



# MERGE SORT EXAMPLE






ThS.Trần Lê Như Quỳnh

# PSEUDO-CODE



```
function mergesort(m)
    var list left, right, result
    if length(m) ≤ 1
    return m
    else
    var middle = length(m) / 2
    for each x in m up to middle - 1
            add x to left
    for each x in m at and after middle
            add x to right
    left = mergesort(left)
    right = mergesort(right)
    if last(left) ≤ first(right)
    append right to left
    return left
    result = merge(left, right)
return result
```

```
function merge(left,right)
    var list result
    while length(left) > 0 and length(right) > 0
            if first(left) ≤ first(right)
                    append first(left) to result
                    left = rest(left)
            else
                    append first(right) to result
                    right = rest(right)
            if length(left) > 0
                    append rest(left) to result
            if length(right) > 0
    append rest(right) to result
return result
```

# Chạy tay



| -9 | 19 | 9 | 0 | 12 |
|---|---|---|---|---|

Step1: n = length – 1= 5-1 =4 => mid = 4/2 = 2
Arr 1 = [-9; 19; 9] => recursive arr1
Arr 2 = [0; 12] => recursive arr2

Step2:
Arr1.1 =[-9; 19] => recursive arr1.1
Arr1.2 = [9]
Arr2.1 =[0]
Arr2.2 =[12]

Step3:
Arr1.1.1 =[-9]
Arr1.1.2=[ 19]
Arr1.2 = [9]
Arr2.1 =[0]
Arr2.2 =[12]

ThS.Trần Lê Như Quỳnh

# Chạy tay



ThS.Trần Lê Như Quỳnh

| -9 | 19 | 9 | 0 | 12 |
|---|---|---|---|---|

Step4: merger
Arr1.1.1 =[-9]
Arr1.1.2=[ 19]
Arr1.2 = [9]
Arr2.1 =[0]
Arr2.2 =[12]
=> Sort merger arr1.1= [-9; 19]

Step5: merger
Arr1.1= [-9; 19]
Arr1.2 = [9]
Arr2.1 =[0]
Arr2.2 =[12]
⇒ Sort merger arr1
⇒ -9 vs 9 => min = -9 => [-9]
⇒ 9 vs 19 => min = 9 => [-9,9]
⇒ 19 =>[-9; 9; 19]

Step3:

# Chạy tay



| -9 | 19 | 9 | 0 | 12 |
|---|---|---|---|---|

Step5: merger
Arr1.1= [-9; 19]
Arr1.2 = [9]
Arr2.1 =[0]
Arr2.2 =[12]
⇒ Sort merger arr2
⇒ 0 vs 12 => min = 0=> [0]
⇒ 12=> [0;12]

Step6: merger
Arr1 =[- 9; 9; 19]
Arr2 =[0;12]
⇒ Sort merger arr
⇒ -9 vs 0 =>[-9]
⇒ 9 vs 0 =>[0;-9]
⇒ 9 vs 12 => [0;-9; 9]
⇒ 19 vs 12 => [0;-9; 9; 12]
⇒ 19 => [0;-9; 9; 12; 19]

ThS.Trần Lê Như Quỳnh

ThS.Trần Lê Như Quỳnh

# QUICK SORT

## SORTING ALGORITHMS

# DEFINITION

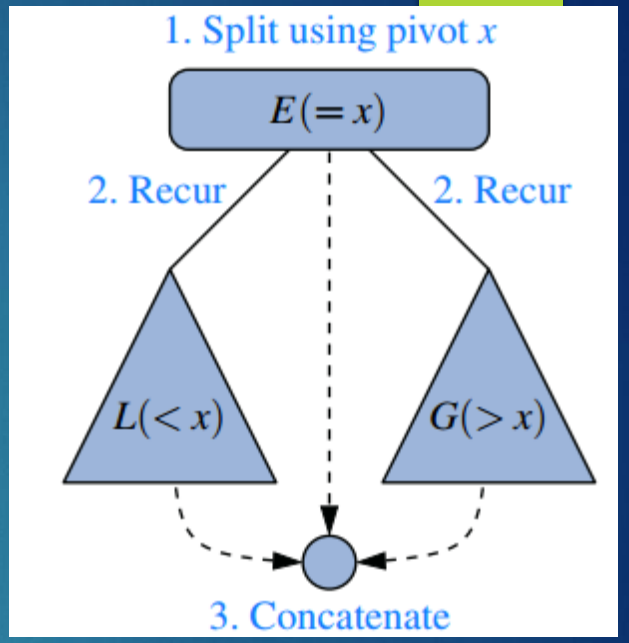


- **Divide**: If S has at least two elements (nothing needs to be done if S has zero or one element), select a specific element x from S, which is called the pivot. As is common practice, choose the pivot x to be the last element in S.
- Remove all the elements from S and put them into three sequences:
  - L, storing the elements in S less than x
  - E, storing the elements in S equal to x
  - G, storing the elements in S greater than x
- Of course, if the elements of S are distinct, then E holds just one element—the pivot itself.
- **Conquer**: Recursively sort sequences L and G.
- **Combine** : Put back the elements into S in order by first inserting the elements of L, then those of E, and finally those of G.

ThS.Trần Lê Như Quỳnh

# PSEUDO-CODE






- function quicksort(array)
- var list less, equal, greater
- if length(array) ≤ 1
- return array

- select a pivot value pivot from array
- for each x in array
- if x < pivot then append x to less
- if x = pivot then append x to equal
- if x > pivot then append x to greater
- return concatenate(quicksort(less), equal, quicksort(greater))

# HOW TO PICK PIVOT



ThS.Trần Lê Như Quỳnh

- Picking the first element as pivot
- Picking Pivots at Random
- Picking Median-of-three As pivot
- Using partition algorithm

# PICK THE FIRST ELEMENT AS PIVOT

- $\Theta(n^2)$-time worst case, most notably when the original sequence is already sorted, reverse sorted, or nearly sorted

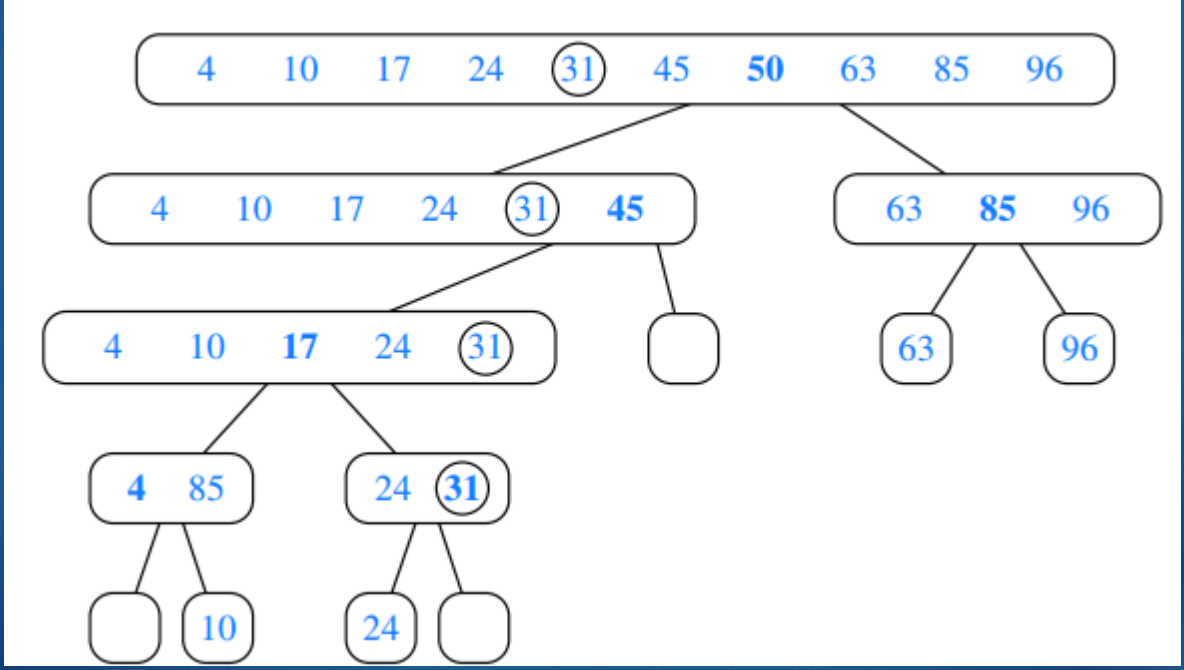
ThS.Trần Lê Như Quỳnh

# PICKING PIVOTS AT RANDOM



ThS.Trần Lê Như Quỳnh

- The expected running time of randomized quick-sort on a sequence S of size n is O(n logn).



# PICKING MEDIAN-OF-THREE AS PIVOT





- This median-of-three heuristic will more often choose a good pivot and computing a

Example: Median-of-three Partitioning

- Let input `S = {6, 1, 4, 9, 0, 3, 5, 2, 7, 8}`
- `left=0` and `S[left] = 6`
- `right=9` and `S[right] = 8`
- `center = (left+right)/2 = 4` and `S[center] = 0`
- Pivot
  - = Median of `S[left]`, `S[right]`, and `S[center]`
  - = median of 6, 8, and 0
  - = `S[left] = 6`

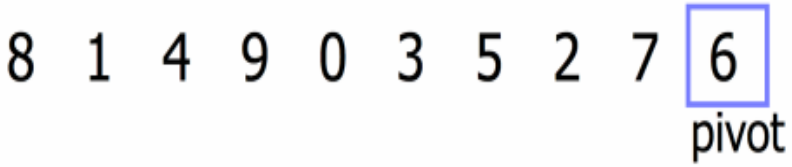
# PARTITION ALGORITHM



ThS.Trần Lê Như Quỳnh

Original input : `S = {6, 1, 4, 9, 0, 3, 5, 2, 7, 8}`

Get the pivot out of the way by swapping it with the last element




Have two 'iterators' – `i` and `j`

- `i` starts at first element and moves forward
- `j` starts at last element and moves backwards

# PARTITION ALGORITHM



ThS.Trần Lê Như Quỳnh

**While (i < j)**

- Move `i` to the right till we find a number greater than **`pivot`**
- Move `j` to the left till we find a number smaller than **`pivot`**
- **`If (i < j) swap(S[i], S[j])`**
- (The effect is to push larger elements to the right and smaller elements to the left)

Swap the **`pivot`** with **`S[i]`**

# QUICK SORT RECURSIVE



ThS.Trần Lê Như Quỳnh

- function quicksort(array, 'left', 'right')
- // If the list has 2 or more items
- if 'left' < 'right'
- // See "Choice of pivot" section below for possible choices
- choose any 'pivotIndex' such that 'left' ≤ 'pivotIndex' ≤ 'right'
- // Get lists of bigger and smaller items and final position of pivot
- 'pivotNewIndex':= partition(array, 'left', 'right', 'pivotIndex')
- // Recursively sort elements smaller than the pivot
- quicksort(array, 'left', 'pivotNewIndex' - 1)
- // Recursively sort elements at least as big as the pivot
- quicksort(array, 'pivotNewIndex' + 1, 'right')

# QUICK SORT NON_RECURSIVE



ThS.Trần Lê Như Quỳnh

```
Procedure QuickSort(a[1..n]) {
  Var list S, E; Int m:=1
  S(m):=1; E(m):= n;
  While m>0 {
        k=S(m); l=E(m)
        m:=m-1;
        if l<k then {
              i=Part(k,l);
              m=m+1;
        S(m):=i+1
        E(m):=l
} }
}
```