Data Structures and Algorithms

Lab 4
Algorithmic Strategies and Master Theorem

Agenda

- Recap
  - Divide and conquer algorithms
  - Master Theorem
- Divide-and-Conquer Merge sort
- Applying Master Theorem

Divide and conquer algorithms:

1. Divide: Divide the given problem into sub-problems using recursion.
2. Conquer: Solve the smaller sub-problems recursively. If the subproblem is small enough, then solve it directly.
3. Combine: Combine the solutions of the sub-problems that are part of the recursive process to solve the actual problem.

Finding Maximum with Divide-and-Conquer Strategy

2	6	3	19	0	283	2	87	82	49	4	7	28	74	8	29

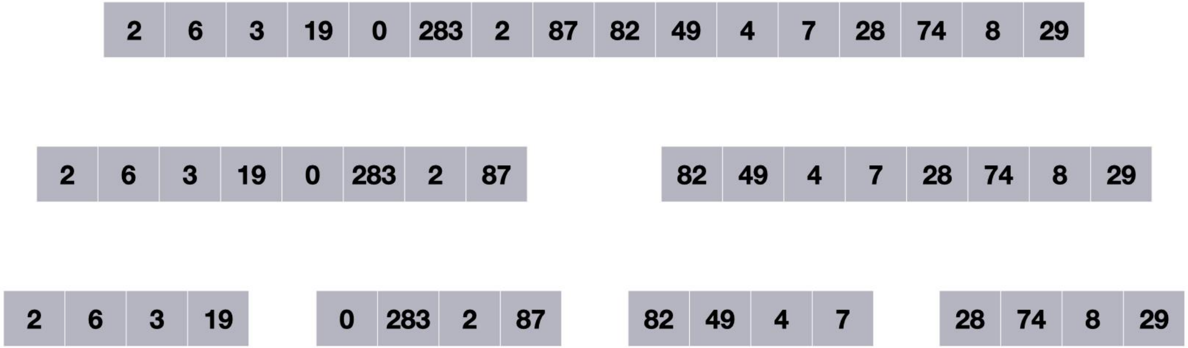
Finding Maximum with Divide-and-Conquer Strategy

2	6	3	19	0	283	2	87	82	49	4	7	28	74	8	29

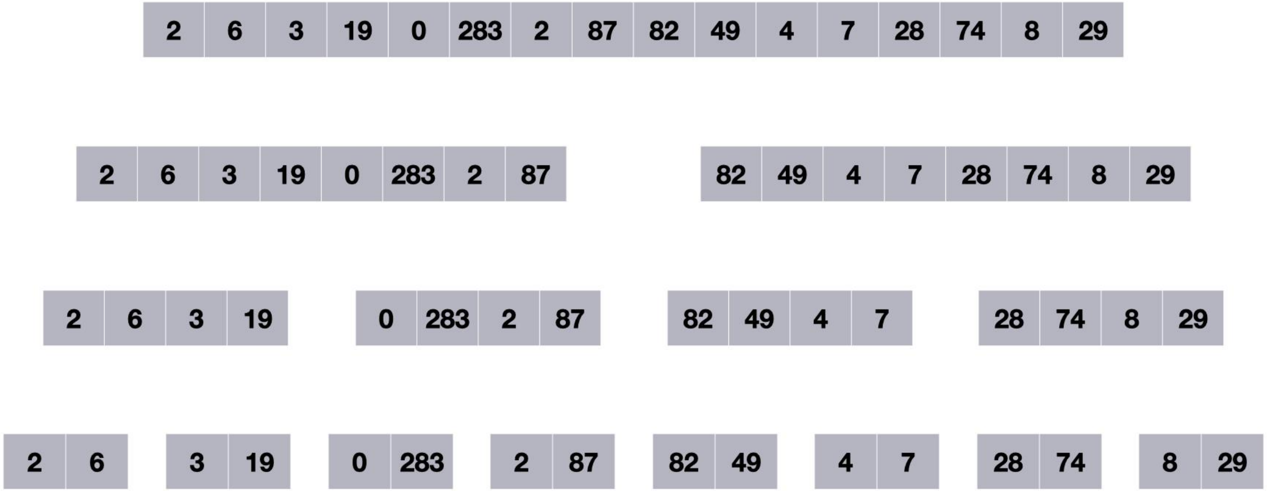
2	6	3	19	0	283	2	87

82	49	4	7	28	74	8	29

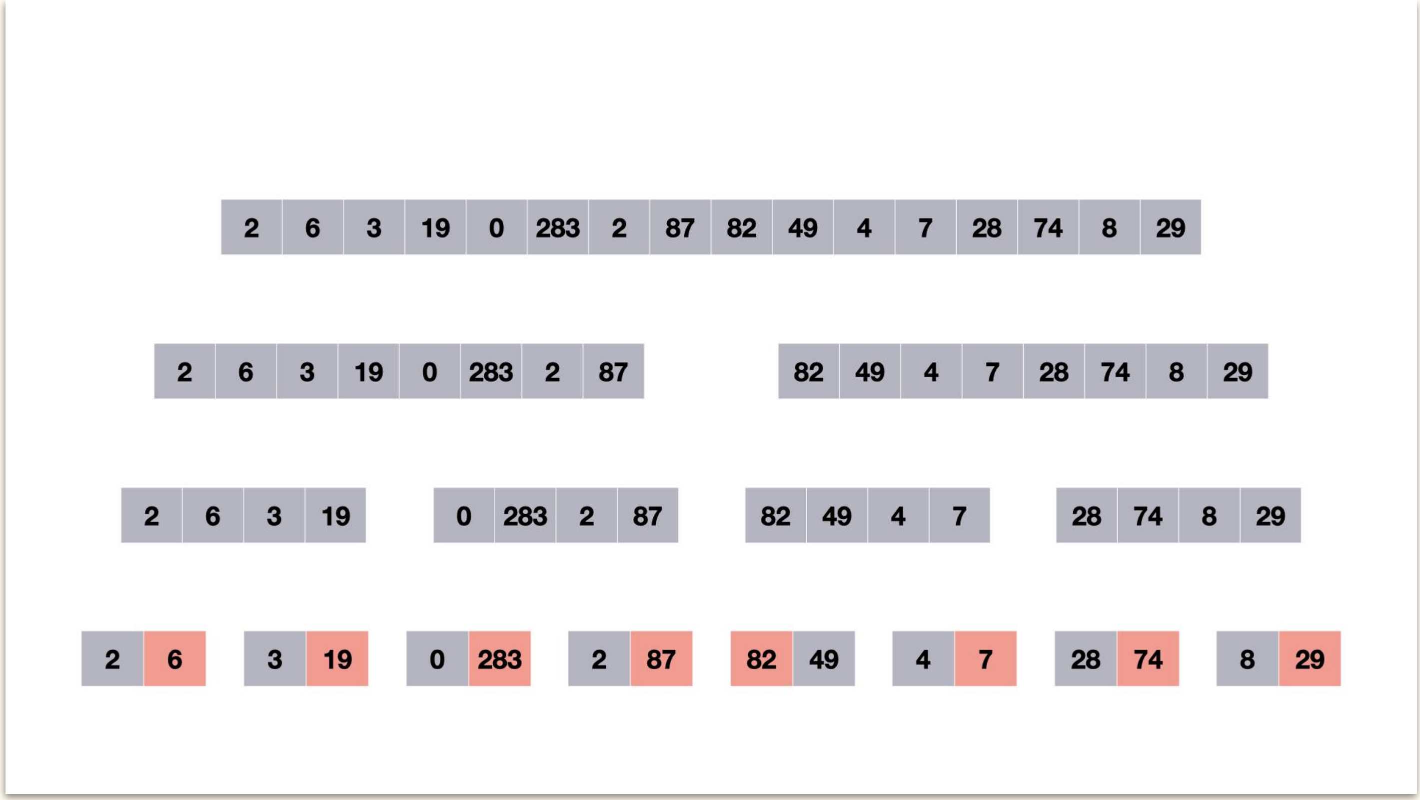
Finding Maximum with Divide-and-Conquer Strategy



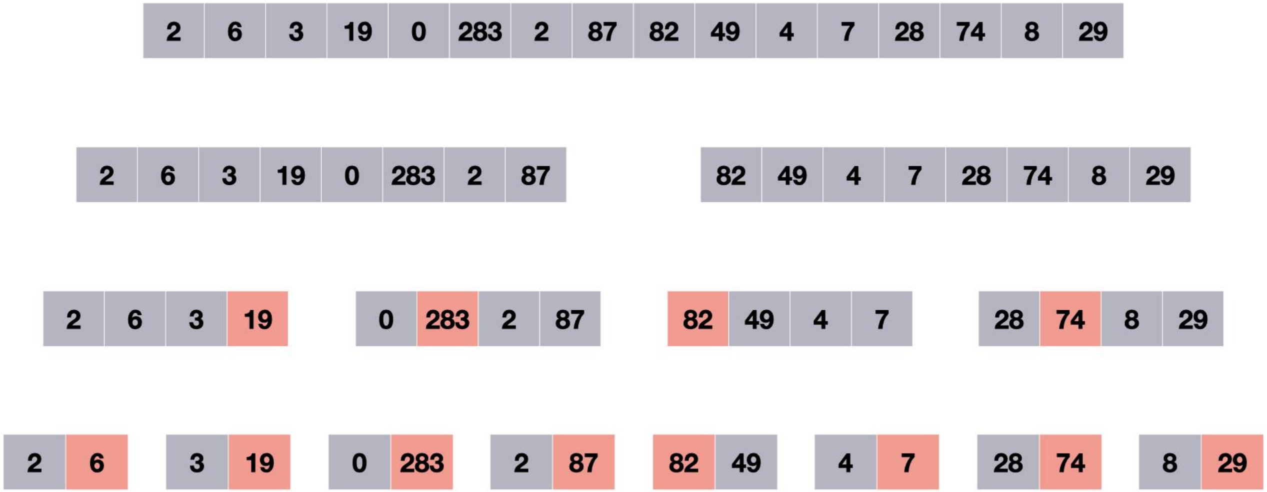
Finding Maximum with Divide-and-Conquer Strategy



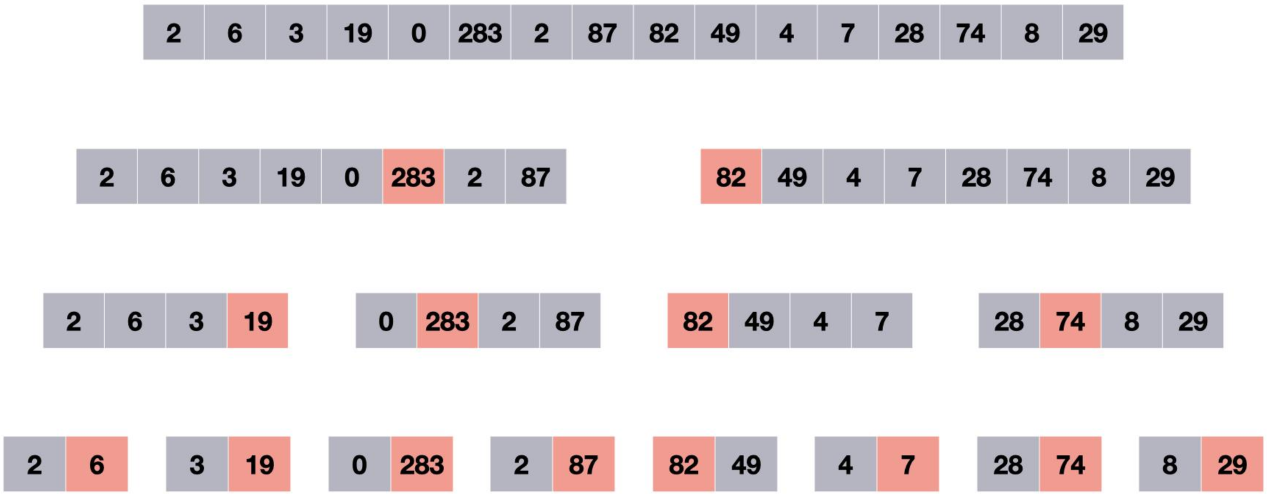
Finding Maximum with Divide-and-Conquer Strategy



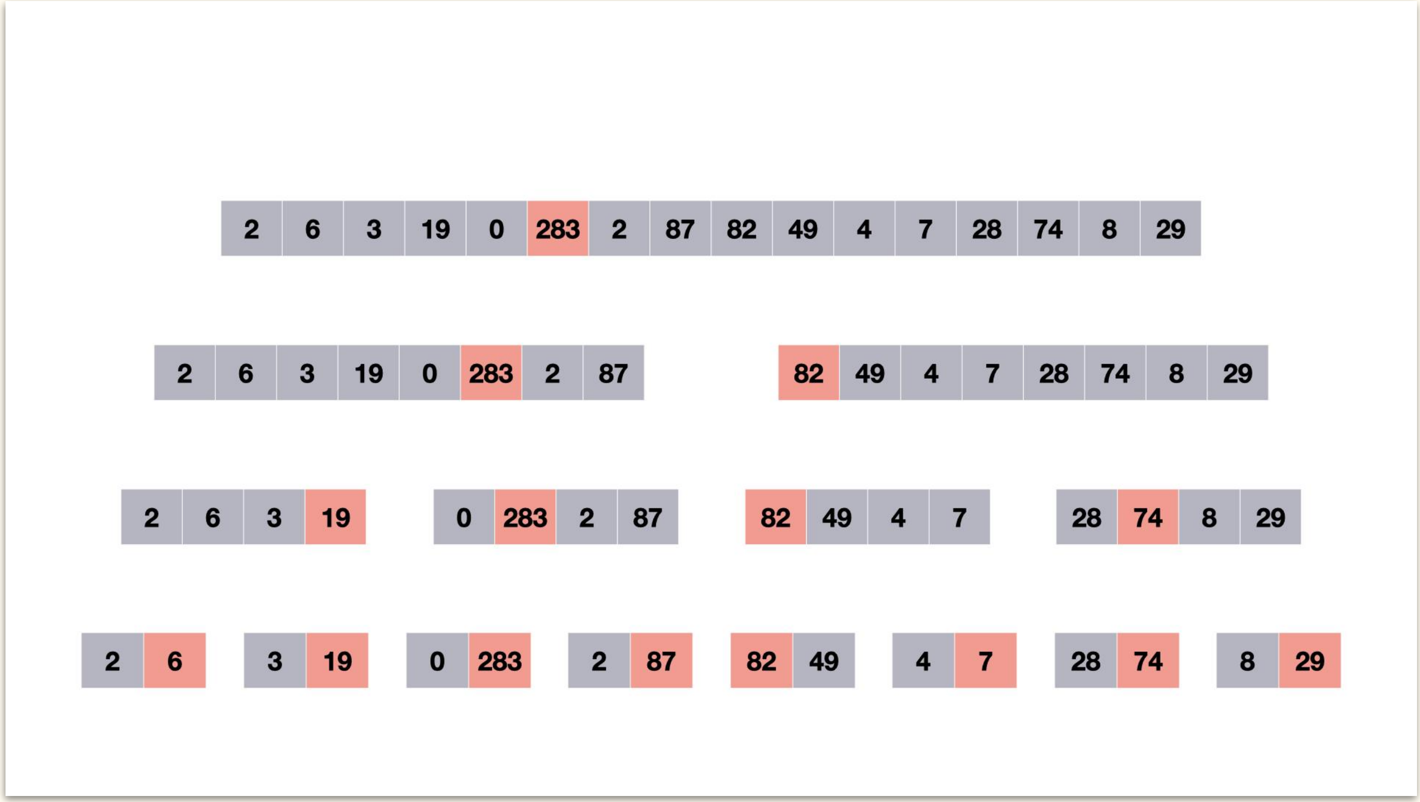
Finding Maximum with Divide-and-Conquer Strategy



Finding Maximum with Divide-and-Conquer Strategy



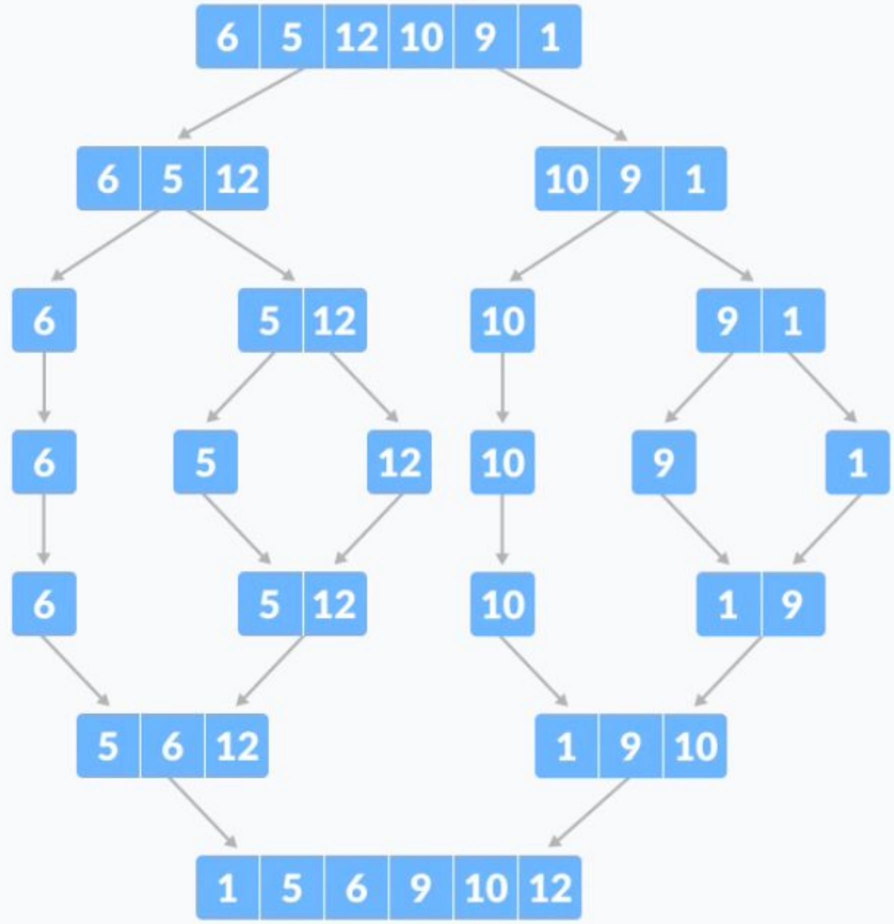
Finding Maximum with Divide-and-Conquer Strategy



Merge sort algorithm:

1. The MergeSort function repeatedly divides the array into two halves until we reach a stage where we try to perform MergeSort on a subarray of size 1 i.e. p == r.
2. After that, the merge function comes into play and combines the sorted arrays into larger arrays until the whole array is merged.

```
MergeSort(A, p, r):
    if p > r
        return
    q = (p+r)/2
    mergeSort(A, p, q)
    mergeSort(A, q+1, r)
    merge(A, p, q, r)
```

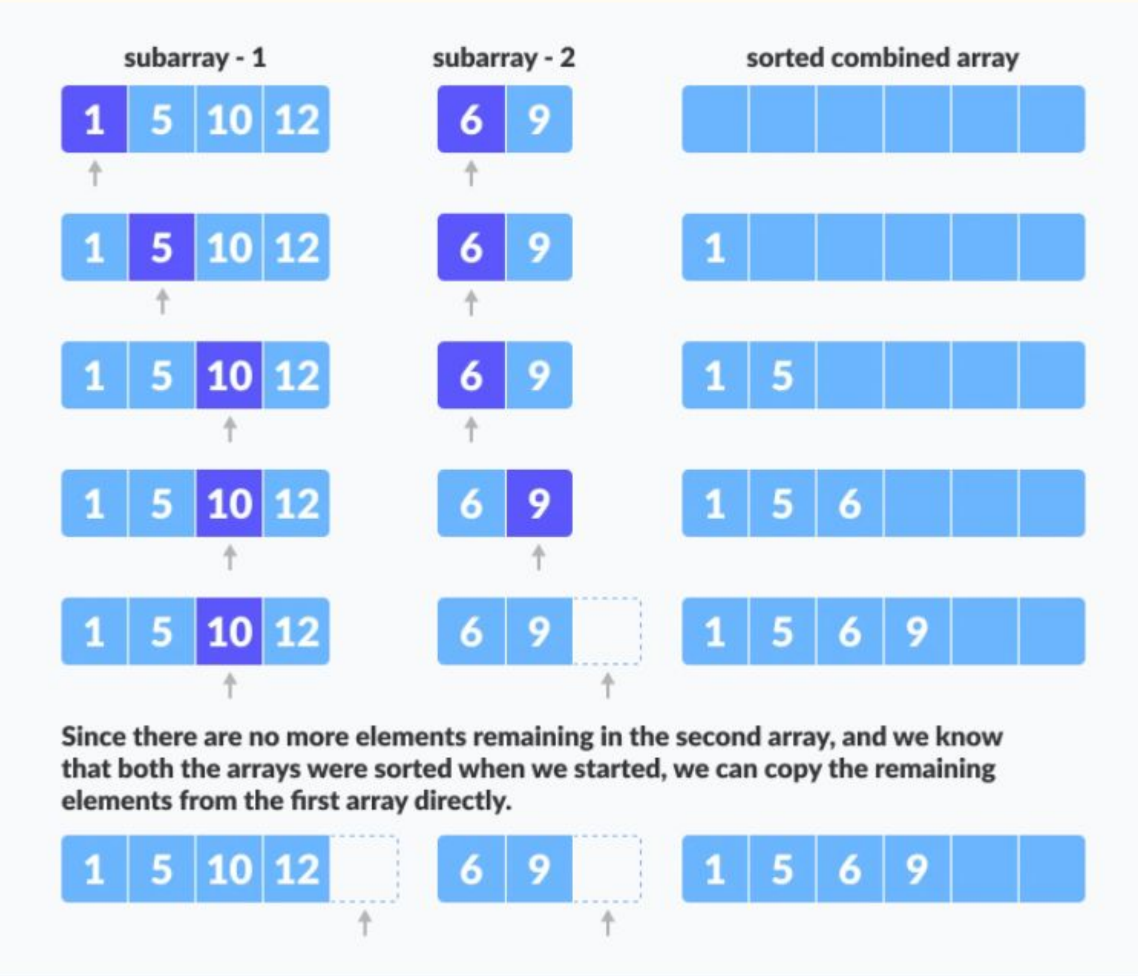


Merge Sort example

The merge Step of Merge Sort:

The merge step is the solution to the simple problem of merging two sorted lists(arrays) to build one large sorted list(array).

```
Have we reached the end of any of the arrays?
    No:
        Compare current elements of both arrays
        Copy smaller element into sorted array
        Move pointer of element containing smaller element
    Yes:
        Copy all remaining elements of non-empty array
```


subarray - 1
subarray - 2
sorted combined array
Since there are no more elements remaining in the second array, and we know that both the arrays were sorted when we started, we can copy the remaining elements from the first array directly.

Merge step

Recurrence Relation

$$T(n) = aT(\frac{n}{b}) + f(n)$$

Recurrence Relation

?

$$T(n) = \boxed{a}\,T\left(\boxed{\frac{n}{b}}\right) + \boxed{f(n)}$$

? ?

Recurrence Relation

Size of each subproblem

$$T(n) = \boxed{a}\,T\left(\boxed{\frac{n}{b}}\right) + \boxed{f(n)}$$

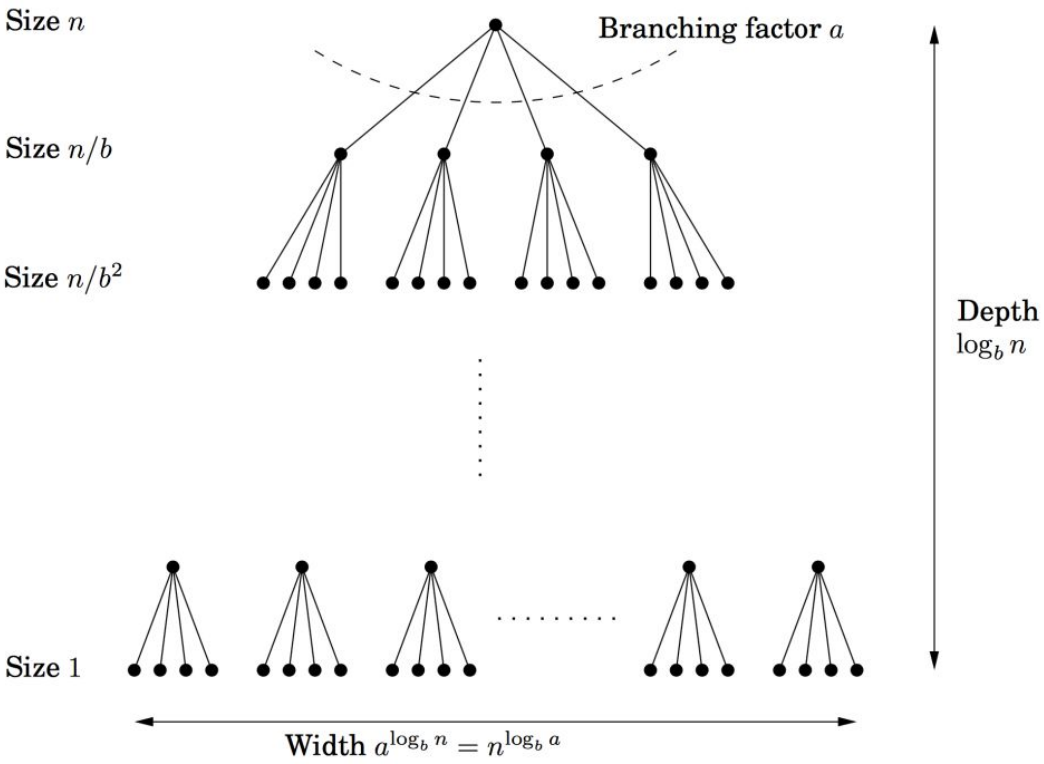
Number of subproblems.

Cost of dividing the problem and merging i.e., anything done outside the recursion call

Divide-and-Conquer

- Master Theorem

$$T(n) = \boxed{aT(n/b)} + \boxed{f(n)}$$



Master theorem (link to demonstration)

Assumptions:

$$T(n) = aT(\frac{n}{b}) + f(n)$$

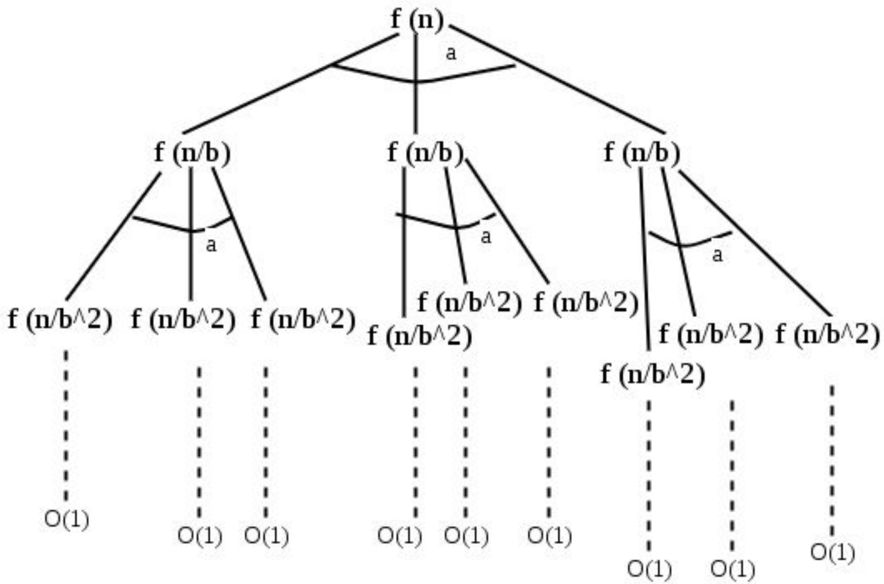
- a $\geq$ 1 and b $>$ 1 are constants
- f(n) is asymptotically positive function

Then T(n) has the following asymptotic bounds.

Case 1. If $f(n) = O(n^{log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{log_b a})$

Case 2. If $f(n) = \Theta(n^{log_b a})$ then $T(n) = \Theta(n^{log_b a} \lg n)$

Case 3. If $f(n) = \Omega(n^{log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if a f(n/b) ≤ c f(n) for some constant c < 1 and all sufficiently large n, then $T(n) = \Theta(f(n))$.



$$T(n) = aT(\frac{n}{b}) + f(n)$$

- In recurrence tree method, we calculate total work done.
- If the work done at leaves is polynomially more, then leaves are the dominant part, and our result becomes the work done at leaves (Case 1).
- If work done at leaves and root is asymptotically same, then our result becomes width multiplied by work done at any level (Case 2).
- If work done at root is asymptotically more, then our result becomes work done at root (Case 3).

Divide-and-Conquer Merge Sort (time complexity)

Exercise 1. Write down recurrence relation for the running time of merge sort algorithm:

Exercise 2. Apply Master Theorem and get asymptotic complexity of T(n) in closed form.

Divide-and-Conquer Merge Sort (time complexity)

Exercise 1. Write down recurrence relation for the running time of merge sort algorithm:

$$T(n) = 2T(n/2) + \Theta(n)$$

Exercise 2. Apply Master Theorem and get asymptotic complexity of T(n) in closed form.

It falls in case 2 as f(n) = c*n and $Log_b(a)$ is 1. So the solution is $\Theta(n\ Log(n))$

Apply master theorem

Exercise 4.4. Find asymptotic complexity for the running time T(n) using Master Theorem for the following recurrence relations:

1. $T(n) = 3T(n/2) + n^2$
2. $T(n) = 4T(n/2) + n^2$
3. $T(n) = T(n/2) + 2^n$
4. $T(n) = 2^n T(n/2) + n^2$
5. $T(n) = 16T(n/4) + n$
6. $T(n) = 2T(n/2) + n \log n$
7. $T(n) = 2T(n/2) + \frac{n}{\log n}$
8. $T(n) = 2T(n/4) + n^{0.51}$
9. $T(n) = \frac{T(n/2)}{2} + \frac{1}{n}$
10. $T(n) = 16T(n/4) + n!$

See You next week!