Asymptotic Notations and Merge Sort

(Class 7)

From Book's Page No 34 (Chapter 2)

- In last class we calculated the running time:

$$f(n) = 4n^2 + 2n$$

- You select $g(n)$ from the $f(n)$ function as n^2.
- Based on dominating term.
- We can prove that the $g(n)$ we selected is correct or not.
- How to check the correctness of $g(n)$.

Asymptotic Notation Example

- Always remember that θ-notation is a boundary around the $f(n)$ function.
- We have established that $f(n) \in n^2$.
- Let's show why $f(n)$ is not in some other asymptotic class.
- First, let's show that $f(n) \notin \theta(n)$.
- Here we can see that, we cannot bound the quadratic $f(n)$ to the linear n.
- So, lets proof it mathematically.

- Show that $f(n) \notin \theta(n)$
- If this were true, we would have had to satisfy both the upper and lower bounds.
- The lower bound is satisfied because $f(n) = 8n^2 + 2n - 3$ does grow at least as fast asymptotically as n.

- But the upper bound is false.
- Upper bounds requires that:
  - there exist positive constants c_2 and $n_\circ$ such that $f(n) \leq c_2 n$ for all $n \geq n_\circ$.

- Informally we know that $f(n) = 8n^2 + 2n - 3$ will eventually exceed $c_2 n$ no matter how large we make c_2.
- However mathematically, to see this, suppose we assume that constants c_2 and $n_\circ$ did exist such that $8n^2 + 2n - 3 \leq c_2 n$ for all $n \geq n_\circ$.
- Since this is true for all sufficiently large n then it must be true in the limit as n tends to infinity.

- Now we prove that as n goes to infinity this condition cannot be satisfied.
- If we divide both sides by n, we have:

$$\lim_{n\to\infty}\left(\frac{8n^2+2n-3}{n}\right) \leq \frac{c_2 n}{n}$$

$$\lim_{n\to\infty}\left(8n+2-\frac{3}{n}\right) \leq c_2$$

- It is easy to see that in the limit, the left side tends to ∞.

$$\lim_{n\to\infty}(8n + 2 - \frac{3}{n}) \leq c_2$$

- So, at very large value of n the left side is larger than the right side.
- So, no matter how large c_2 is, the statement is violated. Thus:

$$f(n) \notin \theta(n)$$

Another Example

- Let's show that:

$$f(n) \notin \theta(n^3)$$

- The idea would be to show that the lower bound $f(n) \geq c_1 n^3$ for all $n \geq n_\circ$ is violated.
- Where c_1 and $n_\circ$ are positive constants.

- What is this idea?
- See the graphs of both quadratic and cubic functions.
- The quadratic will grow fast and will not bunded to the quadratic graph curve.

- Informally we know that $f(n) \notin \theta(n^3)$ is true because any cubic function will overtake a quadratic.
- If we divide both sides by n^3, we have:

$$\lim_{n\to\infty}\left(\frac{8n^2+2n-3}{n^3}\right) \leq \frac{c_1 n^3}{n^3}$$

$$\lim_{n\to\infty}\left(\frac{8}{n}+\frac{2}{n^2}-\frac{3}{n^3}\right) \leq c_1$$

- The left side tends to 0. The only way to satisfy this is to set $c_1 = 0$.
- But by hypothesis, c_1 is positive (by the definition of θ-notation).
- This means that:

$$f(n) \notin \theta(n^3)$$

- This discussion was about the lower and upper bounds.
- But we often want to know only the worst-case running time, means the upper bound.
- Or we only want to know the best-case running time, means the lower bound.
- For that purpose, we use O and Ω notations respectively.

Limit Rule

- Recall the definitions of the O, Ω and θ notations.

$$\theta\big(g(n)\big) : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$$O\big(g(n)\big) : 0 \leq f(n) \leq cg(n)$$

$$\Omega\big(g(n)\big) : 0 \leq cg(n) \leq f(n)$$

- These definitions are based on inequalities.
- We can also describe these notion using limits.
- These definitions suggest an alternative way of showing the asymptotic behavior.

Limit Rule for $\boldsymbol{\theta}$-Notation

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$$

- for some constant $c > 0$ (strictly positive but not infinity),
- then $f(n) \in \theta\big(g(n)\big)$.

Limit Rule for O-Notation

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$$

- for some constant $c \geq 0$ (non-negative but not infinity),
- then $f(n) \in O\big(g(n)\big)$.

Limit Rule for Ω-Notation

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0$$

- (Either a strictly positive constant or infinity),
- then $f(n) \in \Omega\big(g(n)\big)$.

- Why we discussed all the above mathematical procedures and equations?
- We compared two algorithms (2D maxima and Sweep-line) based on their analysis.
- Their polynomial equations were simple but in future we will see that there will be more complex equation.

- So, we will use these notations to simplify the mathematical procedures.
- Because we only want to find the better algorithm so we will simplify the complex equations to simple ones.

Asymptotic Intuition

- Here is a list of common asymptotic running times:
- $O(1)$: Constant time; can't beat it!
- $O(\log n)$: Inserting into a balanced binary tree; time to find an item in a sorted array of length n using binary search.
- $O(n)$: About the fastest that an algorithm can run.

- $O(\mathrm{n}\log n)$: Best sorting algorithms.
- $O(n^2), O(n^3)$: Polynomial time. These running times are acceptable when the exponent of n is small or n is not too large, e.g., $n \leq 1000$.
- $O(2^n), O(3^n)$: Exponential time. Acceptable only if n is small, e.g., $n \leq 50$.
- $O(n!), O(n^n)$: Acceptable only for really small n, e.g., $n \leq 20$.

Divide and Conquer Strategy **(Book's Page No 34 (Chapter 2)**

- We will study the algorithm based on their classes or strategies.
- Some of these strategies are:
  - Divide and Conquer
  - Greedy Algorithms
  - Dynamic Programming
  - Graph Algorithms
  - Geometric Algorithms
  - Branch and Bound
- Now in future classes, we will start analysis of these different classes of algorithms.

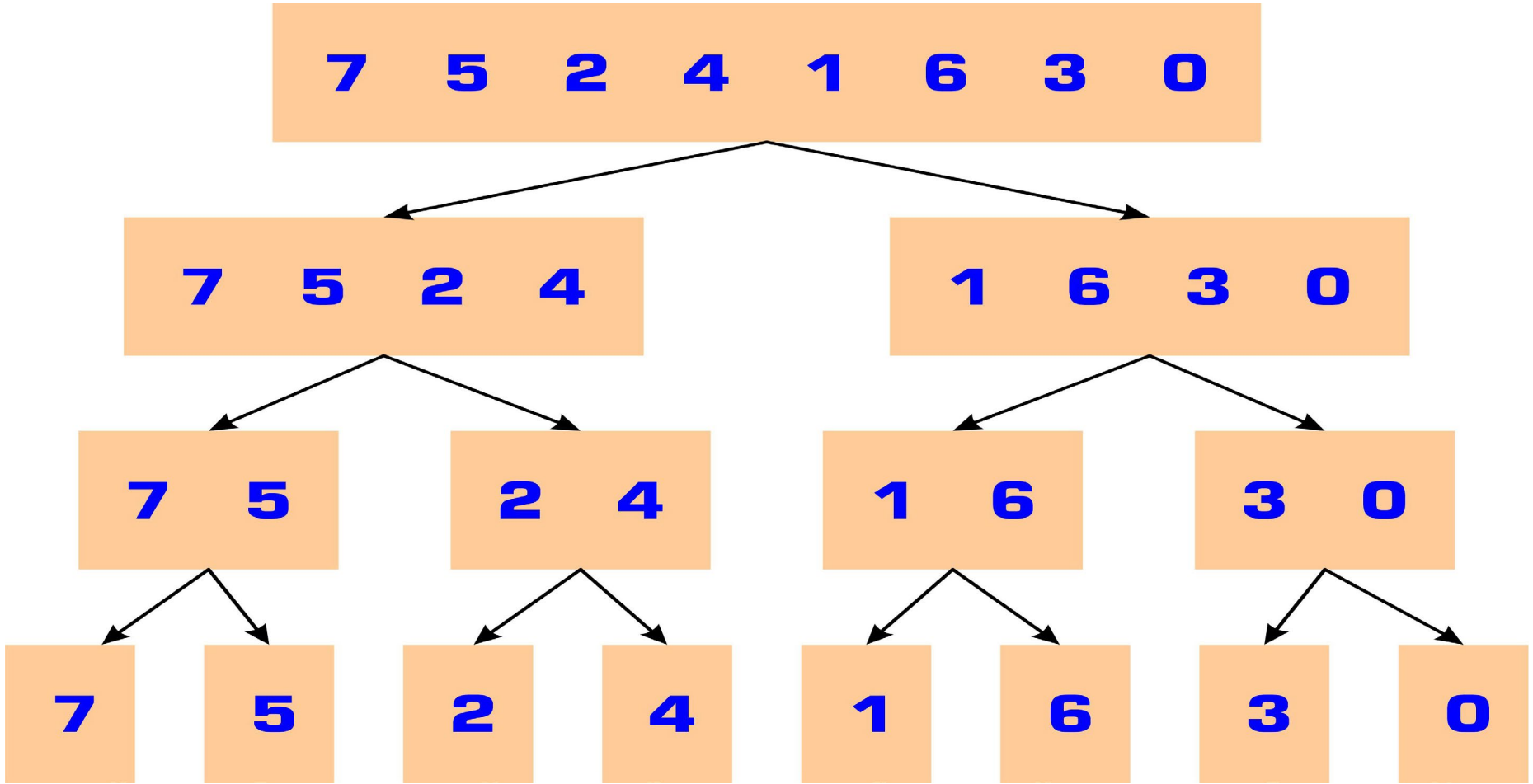
Divide and Conquer Strategy

- In this strategy, first we will discuss the merge sort.
- We will divide the input data then try to solve these parts individually.
- Hence it will be easy to solve smaller parts of the data
- We discussed merge sort in DSA course, but here will be performing mathematical analysis.

- Not limited to sorting, the divide and conquer is a strategy employed to solve a large number of computational problems:
  - **Divide:** the problem into a small number of pieces
  - **Conquer:** solve each piece by applying divide and conquer to it recursively
  - **Combine:** the pieces together into a global solution.

- In merge sort we want to sort the given numbers in ascending or descending order.
- Here is how the merge sort algorithm works:
  - **Divide:** split A down the middle into two subsequences, each of size roughly $n/2$.
  - **Conquer:** sort each subsequence by calling merge sort recursively on each.
  - **Combine:** merge the two sorted subsequences into a single sorted list.

- Recursive algorithms always depend on how we stop the recursion at the end.
- The dividing process ends when we have split the subsequences down to a single item.
- A sequence of length one is trivially sorted.
- The key operation is the combine stage which merges together two sorted lists into a single sorted list.
- Fortunately, the combining process is quite easy to implement.


7 5 2 4 1 6 3 0
7 5 2 4
1 6 3 0
7 5
2 4
1 6
3 0
7
5
2
4
1
6
3
0

- At the end of this algorithm, the sequence divided into single length integers.
- So, at this point each integer is trivially sorted.
- Next, we will merge these parts in such a way that the overall sequence will be sorted.
- In the next class we will study the algorithm of the merger sort.