- Big O recap
  - EXT: master theorem
  - (A proof is in Levitin's Appendix B.)

    For integer constants $(a \geq 1)$ and $(b > 1)$, and function (f) with $(f(n)$ in $\Theta(n^d))$, $(d \geq 0)$, the recurrence :

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

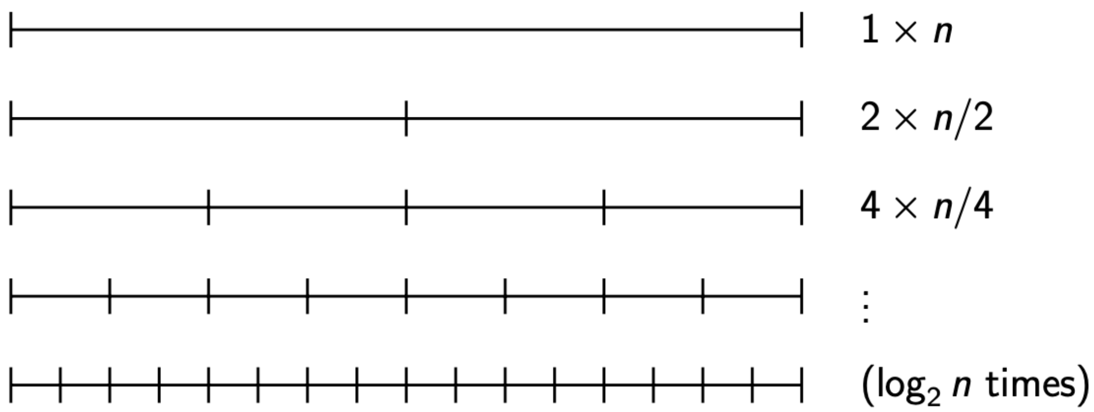
    (with $(T(1) = c)$) has solutions, and

    $$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

    Note that we also allow a to be greater than b.

    e.g.

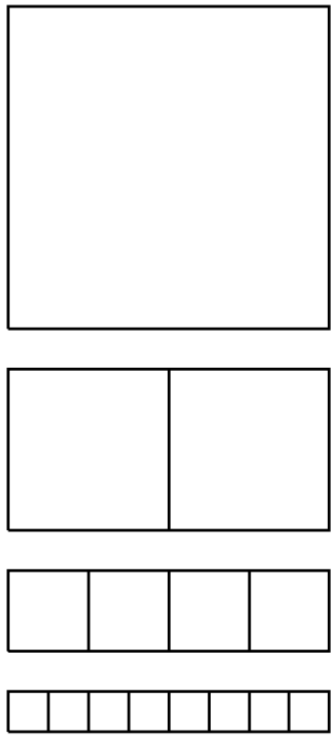
$$T(n) = 2T(n/2) + n \qquad a = 2, b = 2, d = 1$$



$$T(n) = 2T(n/2) + n^2$$

$$a = 2, b = 2, d = 2$$

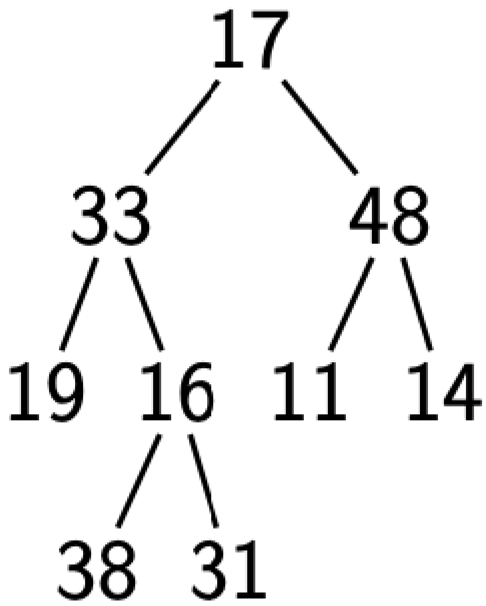
Here $a < b^d$ and we simply get n^d.



(source : Algorithm and data structure lecture slide 2022)

- Searching
  - Linear search
  - Binary search
- Sorting -- See slides
  - Insertion
  - Selection
  - quick
  - Merge -- merge_sort.c
    - Can also do a bottom up merge
  - Heap --- Maybe later, let's get the idea first
    - Binary tree
      - A full binary tree: Each node has 0 or 2 children.
      - A complete tree: Each level filled left to right.
      - Height of a tree : starting from 0, empty tree has height -1
      - BT traversal :
        - **Preorder traversal** visits the root, then the left subtree, and finally the right subtree : 17, 33, 19, 16, 38, 31, 48, 11, 14 -- Create/copy tree
        - **Inorder traversal** visits the left subtree, then the root, and finally the right subtree : 19, 33, 38, 16, 31, 17, 11, 48, 14 -- Sorted array
        - **Postorder traversal** visits the left subtree, the right subtree, and finally the root : 19, 38, 31, 16, 33, 11, 14, 48, 17 -- delete node
        - **Level-order traversal** visits the nodes, level by level, starting from the root : 17, 33, 48, 19, 16, 11, 14, 38, 31 -- Nice for printing, help

      reconstruct the tree.



      -

  - heap condition: The parents is bigger than the children
    - Heapify
    - https://visualgo.net/en/heap?fbclid=IwAR1WX9g46cVWIFO09hHQVAAmO309B5ng23mDzbnIDbuC7AGm7_HFkzsDvZg
  - Properties of tree:
    - The root of the tree $H[1]$ holds a maximal item; the cost of Eject is O(1) plus time to restore the heap.
    - The height of the heap is ⌊log2 n⌋.
    - Each subtree is also a heap.
    - The children of node i are 2i and 2i + 1 -- We can utilise the completeness of the tree and place its elements in level-order in an array H. ($H[0]$ = Null)
      - Heap condition :

$$\forall i \in 0, 1, \ldots, n, \text{we must have } H[i] \leq H[i/2]$$

    - The nodes which happen to be parents are in array positions 1 to ⌊n/2⌋.
  - FINALLY, heap sort:
    - Simply heapify then eject($O(logn)$) the largest element n times
- Counting sort
  - non-comparison-based sorting algorithm, ideal for strings in a limited range, e.g. alphabets .
  - occurrences of each value,
  - place elements directly into their sorted position

- efficient for small integer ranges but uses extra space
- Time complexity is $O(n + k)$, n number of elements and k is the range of input.

Sorting summary

Insertion Sort:

- Best Time Complexity: (O(n)) (already sorted array).
- Average, Worst Time Complexity: (O(n^2)) (average or reverse sorted array).
- Space Complexity: (O(1)) (in-place).

Selection Sort:

- Best, Average, Worst Time Complexity: (O(n^2)) (irrespective of input order).
- Space Complexity: (O(1)) (in-place).

Merge Sort:

- Best, Average, Worst Time Complexity: (O(n \log n)).
    - Example: Any array gets split into halves and merged in sorted order.
- Space Complexity: (O(n)) (not in-place due to additional arrays).

Quick Sort:

- Best, Average Time Complexity: (O(n \log n)) (pivot splits array into equal halves).
- Worst Time Complexity: (O(n^2)) (pivot is the smallest or largest element).
- Space Complexity: (O(\log n)) (in-place but stack space for recursion).

Heap Sort:

- Best, Average, Worst Time Complexity: (O(n \log n)) (heapify operations).
- Space Complexity: (O(1)) (in-place).

Counting Sort:

- Best, Average, Worst Time Complexity: (O(n+k)) where (n) is the number of elements in the input array and (k) is the range of the input.
    - Example: Sorting an array of English letters (k is small, 26).
- Space Complexity: (O(k)) for the count array.