UNIT 3: Transform and Conquer

Transform and Conquer:

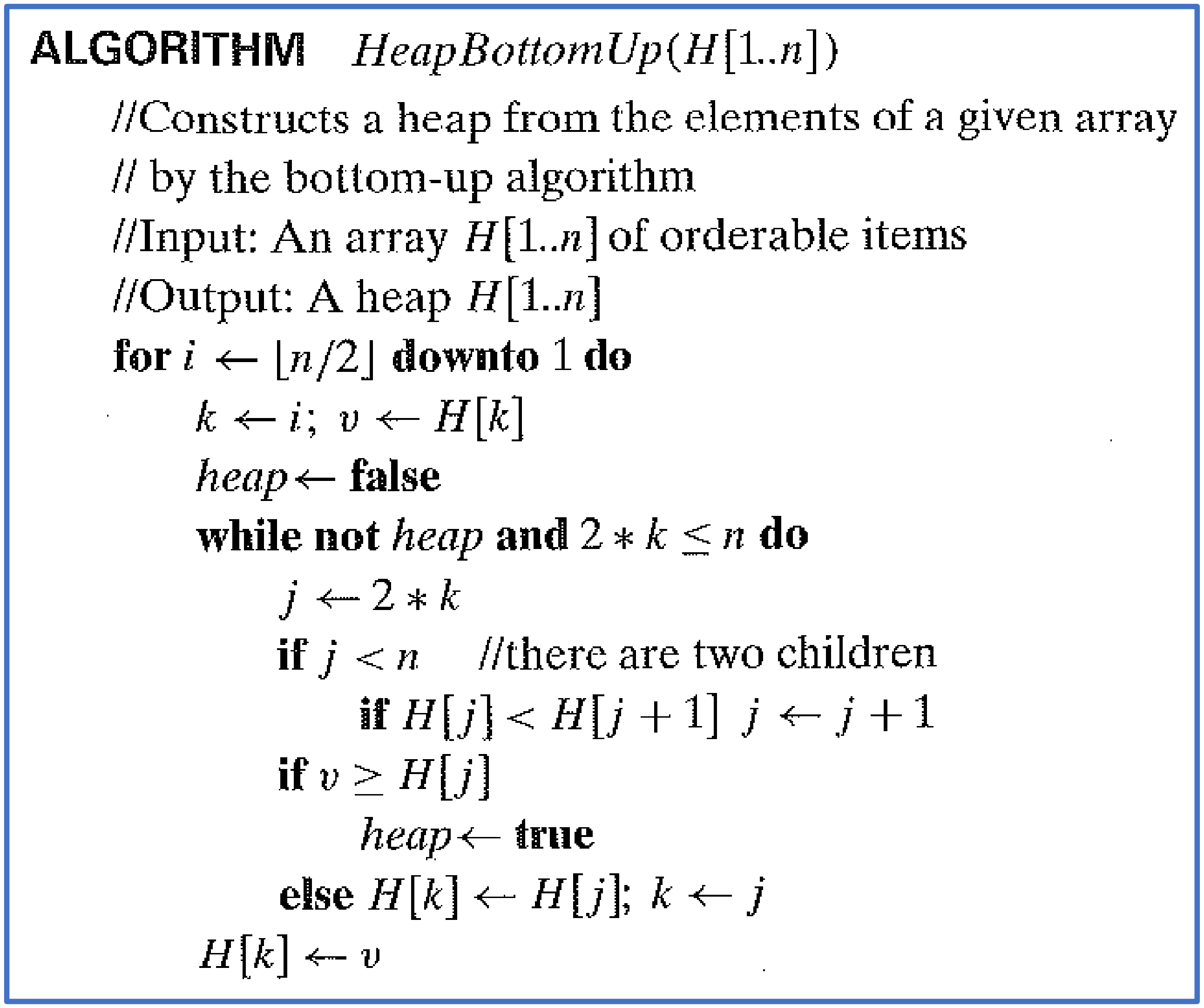
Heap and Heapsort

Heaps

- introduced by **J. W. J. Williams** in 1964
- specialized tree-based data structure
  1. Parental dominance – min/max heap
  2. Structural requirement – almost complete binary tree

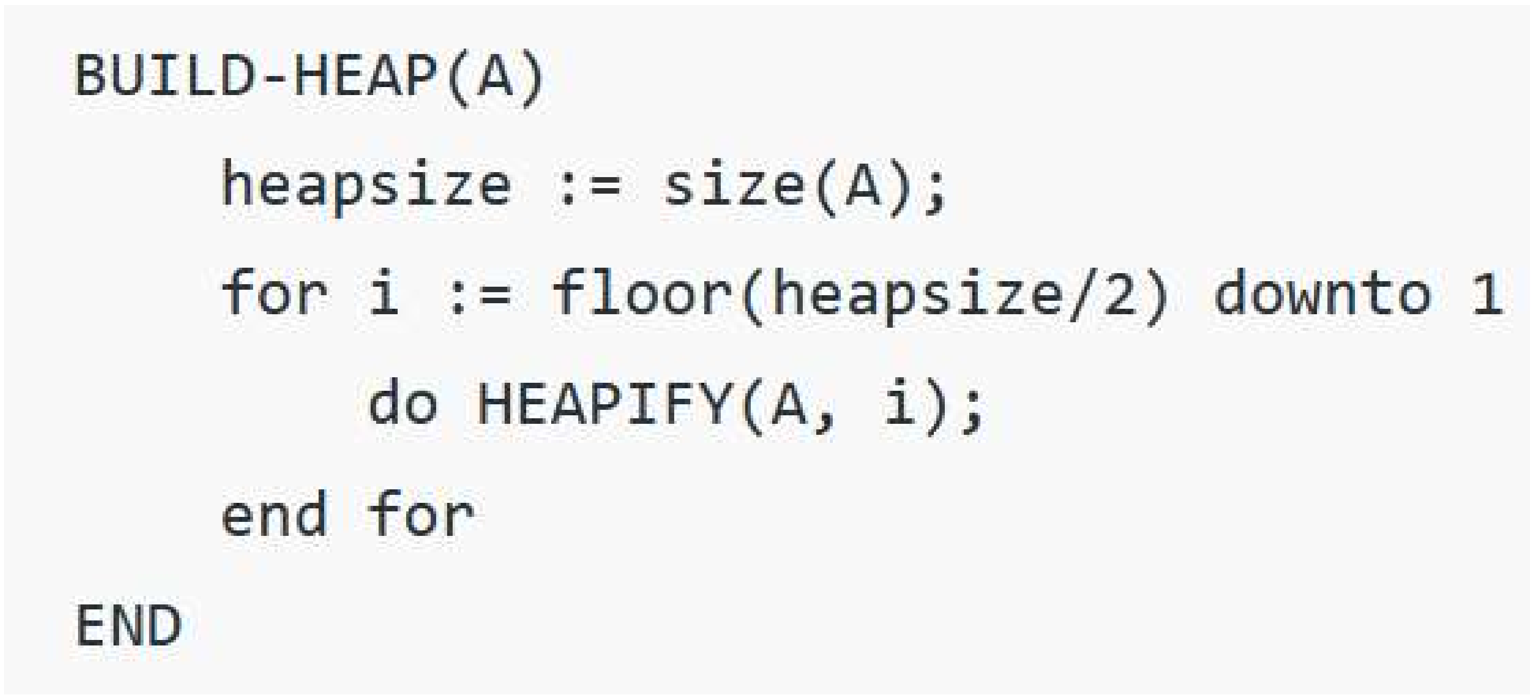
Heap construction

- Bottom – up construction
- Top – down construction



```
ALGORITHM  $HeapBottomUp(H[1..n])$
    //Constructs a heap from the elements of a given array
    // by the bottom-up algorithm
    //Input: An array $H[1..n]$ of orderable items
    //Output: A heap $H[1..n]$
    for $i \leftarrow \lfloor n/2 \rfloor$ downto 1 do
        $k \leftarrow i$; $v \leftarrow H[k]$
        $heap \leftarrow$ false
        while not $heap$ and $2 * k \leq n$ do
            $j \leftarrow 2 * k$
            if $j < n$    //there are two children
                if $H[j] < H[j+1]$  $j \leftarrow j + 1$
            if $v \geq H[j]$
                $heap \leftarrow$ true
            else $H[k] \leftarrow H[j]$; $k \leftarrow j$
        $H[k] \leftarrow v$
```

Build heap



```
BUILD-HEAP(A)
    heapsize := size(A);
    for i := floor(heapsize/2) downto 1
        do HEAPIFY(A, i);
    end for
END
```

Heap variants

2–3 heap,
B-heap,
Beap,
Binary heap,
Binomial heap,
Brodal queue,
d-ary heap,
Fibonacci heap,
K-D Heap,
Leaf heap,
Leftist heap

Pairing heap,
Radix heap,
Randomized meldable heap,
Skew heap ,
Soft heap,
Ternary heap,
Treap,
Weak heap

Heap applications

- **Heapsort**
- **Selection algorithms**: A heap allows access to the min or max element in constant time
- **Graph algorithms**: Prim's minimal-spanning-tree algorithm, Dijkstra's shortest-path algorithm.
- **Priority Queue**
- **K-way merge**
- **Order statistics**: The Heap data structure can be used to efficiently find the kth smallest (or largest) element in an array.

Let’s check our understanding…

Consider a binary max-heap implemented using an array. Which one of the following array represents a binary max-heap?

A. 25,12,16,13,14,8,10
B. 25,12,16,13,10,8,14
C. 25,14,16,13,10,8,12
D. 25,14,12,13,10,8,16

Heapsort

- introduced by **J. W. J. Williams** in 1964
- comparison-based sorting algorithm (thought of as an improved selection sort)
- has favorable worst-case **O(n log n)** runtime
- is an in-place algorithm, but not a stable sort
- same year, R. W. Floyd published an improved version that could sort an array in-place, continuing his earlier research into the **treesort** algorithm

Heapsort

A two-stage algorithm :

Stage 1 (heap construction):

Construct a heap for a given array.

Stage 2 (maximum deletions):

Apply the root-deletion operation n - 1 times to the remaining heap.

Heap sort : Visualization

https://visualgo.net/en/heap

https://www.cs.usfca.edu/~galles/visualization/HeapSort.html

Heapsort – Time complexity

Time Complexity of the heapify() Method:

heapify() function, we walk through the tree from top to bottom. The height of a binary tree (the root not being counted) of size n is $\log_2 n$ at most. The complexity for the heapify() function is accordingly O(log n).

Time Complexity of the buildHeap() Method:

To initially build the heap, the heapify() method is called for each parent node. (A heap of size n has n/2 parent nodes). Hence, the complexity for the buildHeap() method is, therefore, O(n log n).

Total Time Complexity of Heapsort:

The heapify() method is called n-1 times. So the total complexity for repairing the heap is also O(n log n).

Heapsort - variants

- Floyd's heap construction
- Bottom-up heapsort

Other variants

- Ternary heapsort uses a ternary heap
- Memory-optimized heapsort
- Out-of-place heapsort
- smoothsort algorithm (developed by Edsger Dijkstra)

Heapsort - applications

- embedded systems,
- real-time computing,
- systems concerned with maliciously chosen inputs such as the Linux kernel.
- any application which does not expect to be bottlenecked on sorting.

Heapsort primarily competes with Quicksort

Heapsort advantages:

- simple, non-recursive code, minimal auxiliary storage requirement, and reliably good performance.

Disadvantages:

- poor locality of reference, inherently serial in nature
- the accesses to the implicit tree are widely scattered and mostly random, and there is no straightforward way to convert it to a parallel algorithm.

Points to remember:

- Most commercial applications would use **quicksort** for its better average performance: they can tolerate an occasional long run in return for shorter runs most of the time.
- However, quick sort should **never be used in applications which require a guarantee of response time**, unless it is treated as an $O(n^2)$ algorithm in calculating the worst-case response time.
- If you have to assume $O(n^2)$ time, then - if n is small, you're better off using **insertion sort** - which has simpler code and therefore smaller constant factors.
- And if n is large, you should obviously be using **heapsort**, for its guaranteed $O(n\log n)$ time.

Life-critical (medical monitoring, life support in aircraft and space craft) and mission-critical (monitoring and control in industrial and research plants handling dangerous materials, control for aircraft, defence, etc) software will generally have a response time as part of the system specifications. In all such systems, it is not acceptable to design based on average performance, you must always allow for the worst case, and thus treat quicksort as $O(n^2)$.

Heapsort vs Quicksort

- A well-implemented quicksort is usually 2–3 times faster than heapsort.
- **Quicksort has better locality of reference**: partitioning is a linear scan with good spatial locality, and the recursive subdivision has good temporal locality.
- With additional effort, quicksort can also be implemented in mostly branch-free code, and multiple CPUs can be used to sort sub-partitions in parallel.

- Quicksort is preferred when the additional performance justifies the implementation effort.

Heapsort vs Merge sort

- Merge Sort is also usually faster than Heapsort. Besides, unlike Heapsort, Merge Sort is stable.

- Heapsort has an advantage over Merge Sort in that it does not require additional memory, while Merge Sort requires additional memory in the order of O(n).

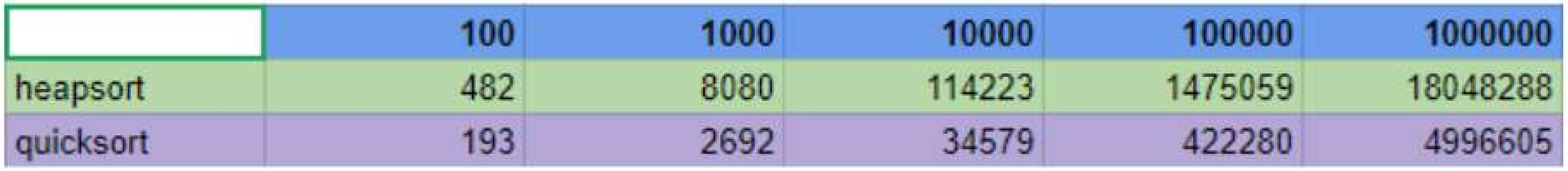
- Merge sort has a clear advantage:
  - When a stable sort is required
  - When taking advantage of (partially) pre-sorted input
  - Sorting linked lists (in which case merge sort requires minimal extra space)
  - Parallel sorting; merge sort parallelizes even better than quicksort and can easily achieve close to linear speedup
  - External sorting; merge sort has excellent locality of reference

Case study:
Implementation at V8 engine in Chrome browser

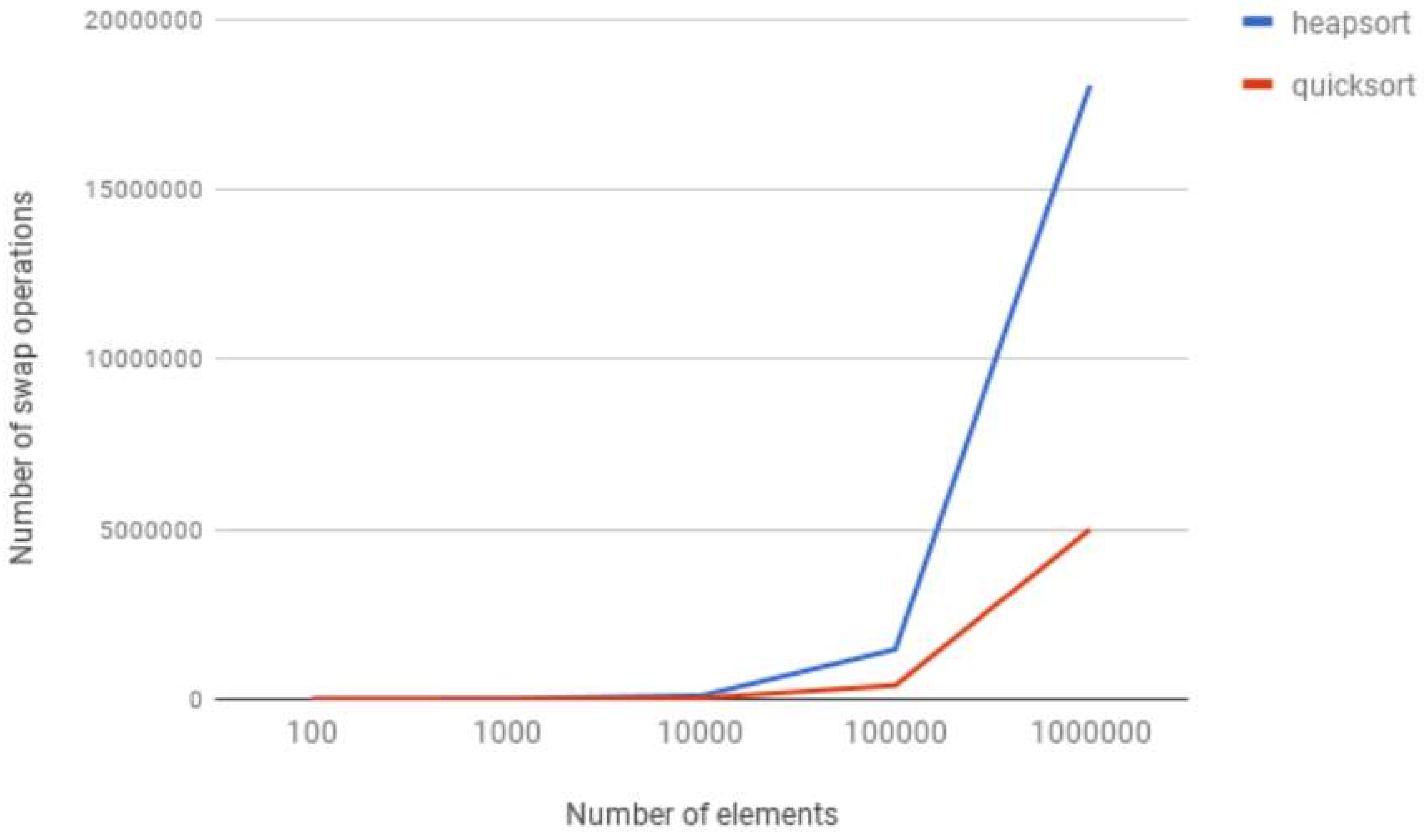
sort function is on Array object in Javascript. In old chrome browser, sort algorithm found in array.js file has heap sort implementation. But this algorithm was soon changed to quicksort.

- Quicksort: partition :: Heapsort : heapify

- Quicksort and heapsort needs swap operation in there core logic.



	100	1000	10000	100000	1000000
heapsort	482	8080	114223	1475059	18048288
quicksort	193	2692	34579	422280	4996605



- It is not big difference in little number of items. But if data size bigger, difference also become larger.

Latest V8 engine has branch to determine which sort algorithm have to use. If number of items is less then 10, V8 use selection sort, otherwise use quicksort.

Let’s check our understanding…

Suppose we are sorting an array of eight integers using heapsort, and we have just finished some heapify (either maxheapify or minheapify) operations. The array now looks like this: 16 14 15 10 12 27 28.

How many heapify operations have been performed on root of heap?

A. 1
B. 2
C. 3 or 4
D. 5 or 6

Let’s check our understanding…

A 3-ary max heap is like a binary max heap, but instead of 2 children, nodes have 3 children. A 3-ary heap can be represented by an array as follows: The root is stored in the first location, a[0], nodes in the next level, from left to right, is stored from a[1] to a[3]. The nodes from the second level of the tree from left to right are stored from a[4] location onward. An item x can be inserted into a 3-ary heap containing n items by placing x in the location a[n] and pushing it up the tree to satisfy the heap property.

Which one of the following is a valid sequence of elements in an array representing 3-ary max heap?

A. 1, 3, 5, 6, 8, 9
B. 9, 6, 3, 1, 8, 5
C. 9, 3, 6, 8, 5, 1
D. 9, 5, 6, 8, 3, 1