Quick Sort - Demonstration

Ex: 2 8 7 1. 3 5 6. 4.

- choose the pivot.

2 8 7 1 3 5 6 | 4.

- compare the pivot with each element.

2 | 8 7 1 3 5 6 | 4.

Start the "wall" for elements smaller than the pivot.

2 | 8 7 1 3 5 6 | 4

Swap 8 & 1. & increase the "wall"

2 1 | 7 8. 3. 5 6 | 4.

2 1. 3 | 8 7 5 6 | 4.

move the pivot to next to the wall (swap)

2 1 3 $\boxed{4}$ 7 5 6 8

recursive the q-sort

2 1 3.

2 | 1.

1 | 2.

2

Stop.

recursive the q-sort.

7 5 6 8

8

7 5 6.

5 7 6.

5 6 7.

5

Stop

7

Stop

=>

1 -> 2 -> 3 -> 4 -> 5 ->

6 -> 7 -> 8.

Sorted.

Quick sort - Algorithm:

1. Given an array of numbers, divide it into 2 sub arrays so that elements of one array are always larger than or equal to elements of the other array.
2. Sort the 2 arrays recursively
3. No additional work is needed, as sorting is in place.
4. This algo is an example of divide & conquer technique.

ANALYSIS: Let n = size of the array to be sorted.

1. Worst case: running time = $\Theta(n^2)$.
2. Average case: running time = $\Theta(n\lg n)$.
3.

PROVING: $T(n)$ = runtime of the algorithm.

1. WORST CASE:

i) Occurs when sub arrays are completely unbalanced. Have $\emptyset$ elements in one sub array & $(n-1)$ elements in the other sub array

ii) worst case occurs when the sorting occurs on a randomised array. Therefore the array of numbers are initially randomized.

iii) $T(n) = T(n-1) + \underbrace{T(0)}_{\text{constant, ignore}} + \underbrace{\Theta(n)}_{\text{partitioning}}$

$$= T(n-2) + \Theta(n-1) + \Theta(n).$$
$$= T(n-3) + \Theta(n-2) + \Theta(n-1) + \Theta(n).$$
$$= T(0) + \Theta(1) + \Theta(2) + \ldots + \Theta(n)$$
$$= T(0) + c\times 1 + c\times 2 + \ldots + c\times n.$$
$$= T(0) + c\times\frac{n(n+1)}{2}$$

For large n, ignore $T(0)$, $(n+1) \approx n$.

$$\Rightarrow T(n) \approx \frac{n^2}{2} = \Theta(n^2)$$

2. BEST CASE:

i) occurs when subarrays are completely balanced everytime

ii) each subarray has $\leq \frac{n}{2}$ elements.

iii) $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$

$\Theta(n)$: partitioning

$2T\left(\frac{n}{2}\right)$: 2 subarrays equal in size $\left(\frac{n}{2}\right)$.

$$= 2\left(2T\left(\frac{n}{2^2}\right) + \Theta\left(\frac{n}{2}\right)\right) + \Theta(n) \quad \left(\Theta(n) = cn, c > 0\right)$$

$$= 2^2 T\left(\frac{n}{2^2}\right) + 2cn$$

$$= 2^r T\left(\frac{n}{2^r}\right) + rcn$$

Let $n = 2^r$ or $r = \log_2 n$, $T(1) = c$

$$T(n) = n.T(1) + rcn$$

$$= nc(1+r)$$

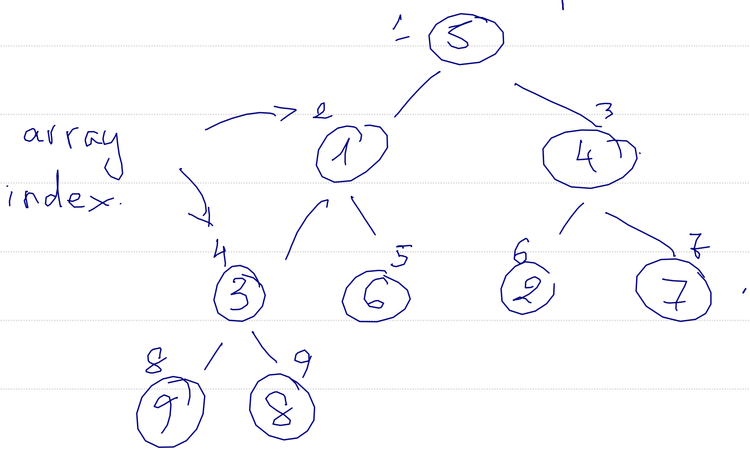
$$= nc(1 + \log_2 n)$$

$$\therefore T(n) = \Theta(n \lg n)$$

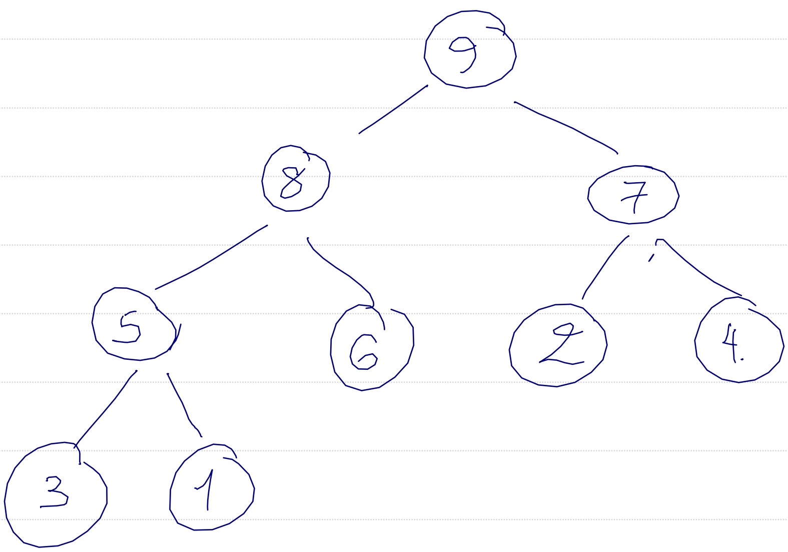
HEAP SORT. (suposed to be fast, but coding is messy).

Eg: 5 1 4 3 6 2. 7 9 8.

① construct heap tree.



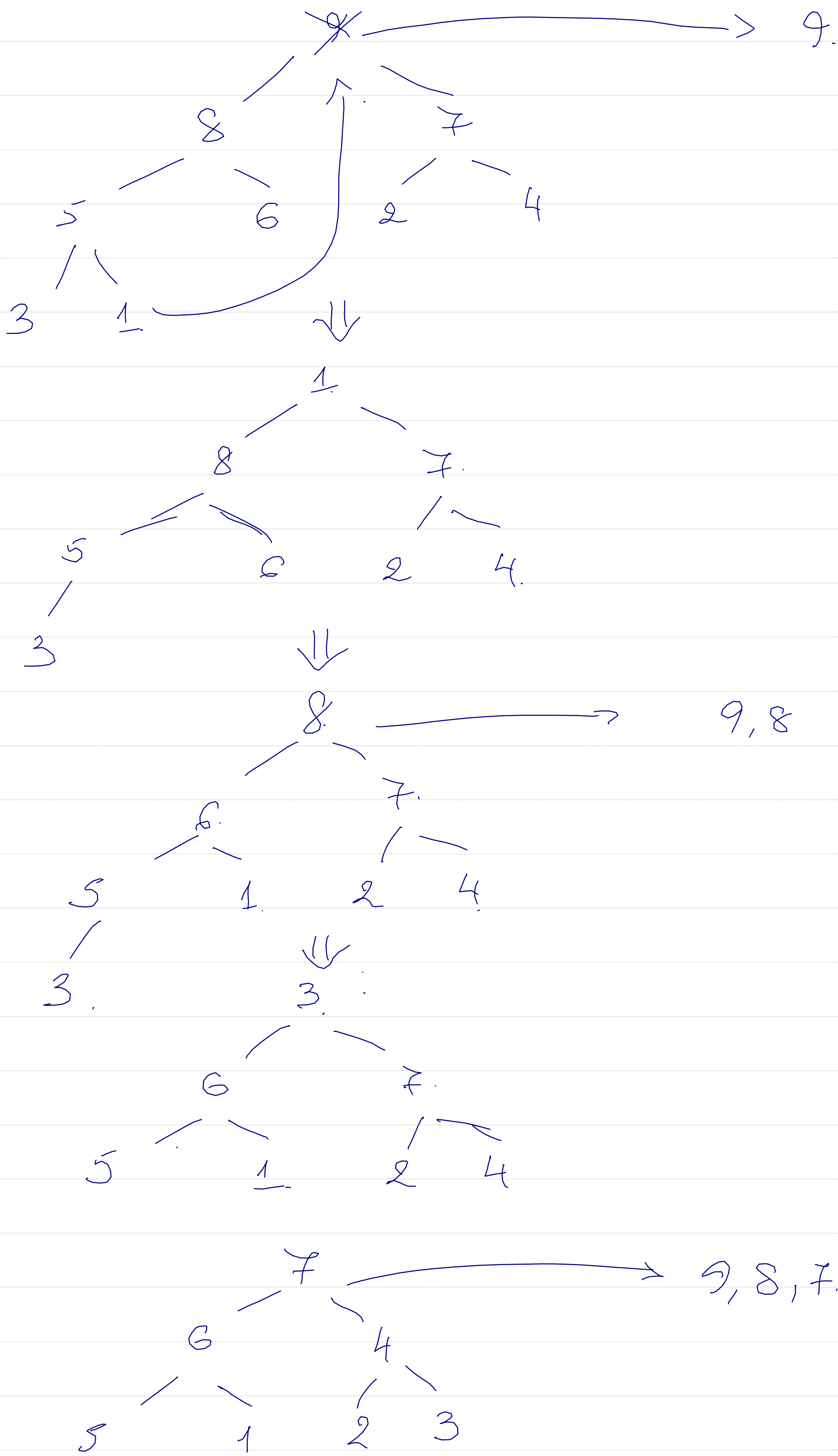
② MAX HEAPIFY.

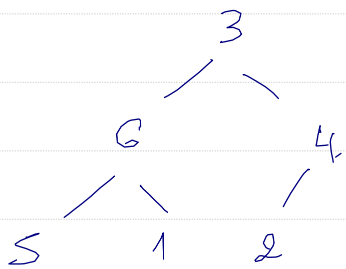


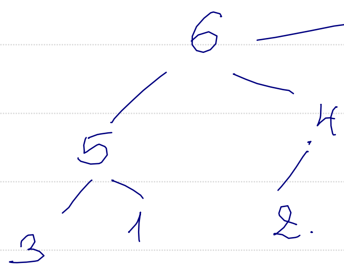
Process:

- start from bottom subtree, from right to left.
- switch child value with parent value if child > parent.
- repeat until becomes max heap tree.

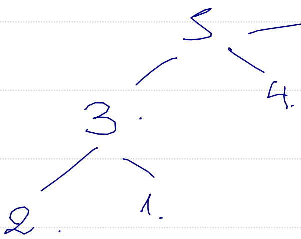
③ - Take out top, move to sorted array
- move bottom node to top
- Re-max heapify.


9.
9, 8
9, 8, 7.

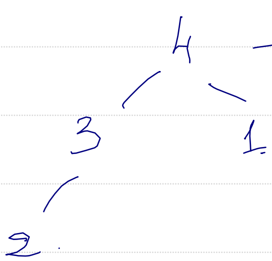




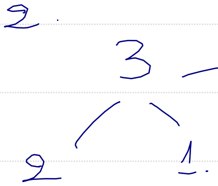
→ 9, 8, 7, 6



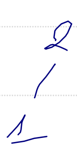
→ 9, 8, 7, 6, 5



→ 9, 8, 7, 6, 5, 4.



→ 9, 8, 7, 6, 5, 4, 3.



→ 9, 8, 7, 6, 5, 4, 3, 2.

1 → 9, 8, 7, 6, 5, 4, 3, 2, 1.

Summary

- What is the advantages of MERGE SORT?

  <u>Ans</u>: $O(n\lg n)$ worst case running time.

- What is the advantage of INSERTION SORT?

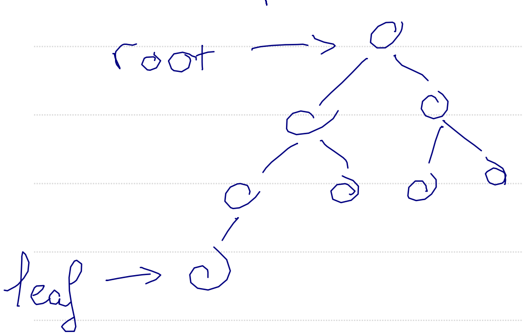
  <u>Ans</u>: sorts in place.

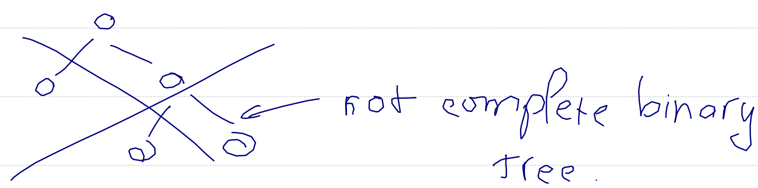
  Also, when array "nearly sorted", runs fast in practice.

$\Downarrow$

<u>HEAP SORT</u>: COMBINES BOTH ADVANTAGES.

- A heap can be seen as a <u>complete binary tree</u>.





- The total time taken by a heapsort is

$$T(n) = O(n) + (n-1)\,O(\lg n)$$
$$= O(n) + O(n\lg n)$$
$$= O(n\lg n) \quad ???$$

LINEAR TIME SORT

- Sorting algorithms that runs in linear time.
- includes : COUNTING SORT, RADIX SORT, & BUCKET SORT.

SUMMARY:

So far:
- WORST CASE : $O(n^2)$.
- GOOD CASE: $O(n \lg n)$.
- BEST CASE: $O(n)$

INSERTION SORT:

+ easy to code.
+ fast on small inputs (less than ~50 elements)
+ fast on nearly-sorted inputs
- $O(n^2)$ worst case.
- $O(n^2)$ average (equally-likely inputs) case.
- $O(n^2)$ reverse-sorted case.

MERGE SORT: (divide & conquer).

- Split array in half
- recursively sort sub arrays.
- linear time merge steps.

+ $O(n \lg n)$ worst case.
- Does not sort in place.

HEAP SORT:

- Use the very helpful heap data structure.
- complete binary tree.
- heap property: parent key > children key (max heap)

+ $O(n \lg n)$ worst case.
+ sorts in place.
- fair amount of shuffling memory around.

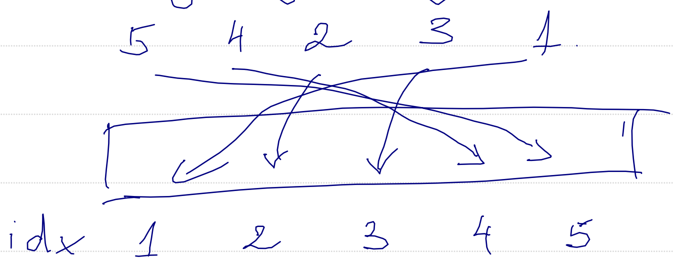
QUICK SORT: divide & conquer

- Partition array into 2 subarrays, recursively sort.
- All of first subarray < all of second array.
- No merge step needed.

+ $O(n \lg n)$ average case.
+ fast in practice.
- $O(n^2)$ worst case.
  - Naive implementation: worst case on sorted input
  - Address this with randomized quick sort.

COUNTING SORT.

Not very useful algorithm.



Match the value to the index (can only work with int & small set).

RADIX SORT

In general, RADIX SORT uses counting sort mechanism:

- fast
- asymptotically fast ($O(n)$)
- simple to code
- a good choice.

Qn. Can RADIX sort be used on floating numbers?

Ans: yes. 0.5967×10^{23}.

sort each digit

MEDIAN SELECTION

FIND kth SMALLEST ELEMENT IN A LIST.

Ex: Find the 7th smallest number of

6 10 13 5 8 3 2 11.

- Pick a pivot randomly → 6
- Partition the list into 3 sublists
  - + L: elements smaller than 6
  - + E: elements equal to 6.
  - + G: elements larger than 6.

⇒ 5 3 2 | 6 | 10 13 8 11.

↑ idx = 4.

- since we look for 7th smallest no. (7 > 4) so we recursively apply the pivot-partition on the 2nd set - G.

10 13 8 11.

- In this new set, we look for the 3rd (7-4) smallest element. We choose 11 to be the pivot. (randomly).

⇒ 10 . 8 11 13 .

11 happens to be the 3rd smallest number.

⇒ 11 is the 7th smallest in the original list

2 3 5 6 8 . 10 11 13.

7th.

DATA STRUCTURE:

- Set in MATHS does not have repeated no.
- Set in Algo can have repeated no.

STACK & QUEUE

- STACK: LIFO.
- QUEUE: FIFO.

i). Q_h = pointer to HEAD of the Queue.
Q_t = pointer to TAIL of the Queue

ii) Find the number of elements in the queue.

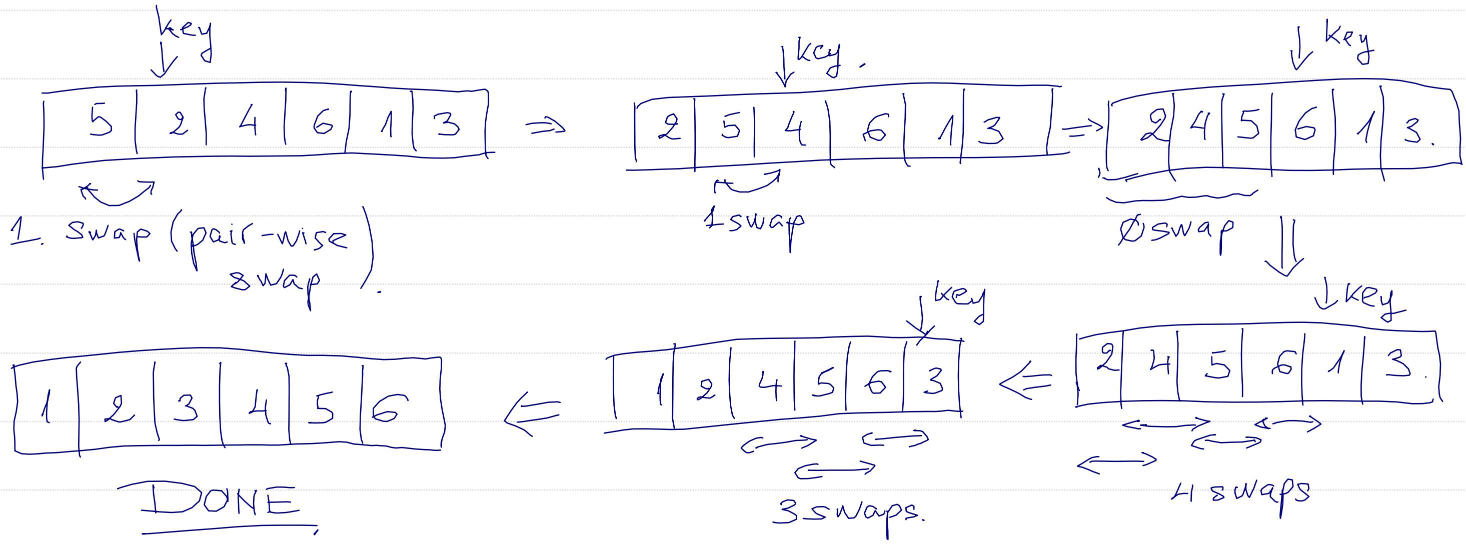
iii) Ex1: $Q_t - Q_h = 12 - 7 = 5$ OK ✓
Ex2: $Q_t - Q_h = 3 - 7 = -4$ Not OK ✗

iv) Find # of elements in the queue
Let it be equal to n.

- $x = Q_t - Q_h$
  if $x \geq 0 \Rightarrow n = x$.
  else $n = |A| + x$.
- Alternatively,
  $n \equiv x \bmod |A|$
  ← Modulo operation

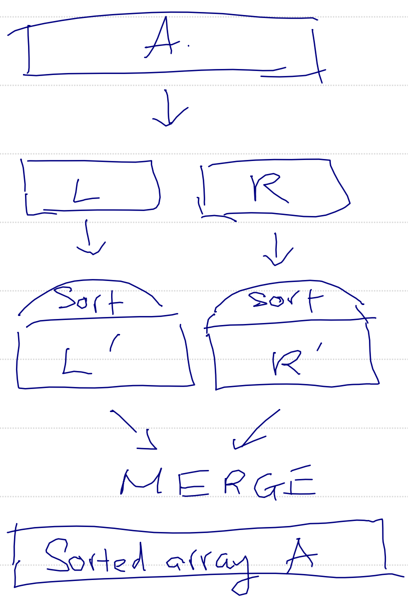
ex: $17 \pmod 3 = 2 \pmod 3$
$6 \pmod{13} = 7 \pmod{13}$.

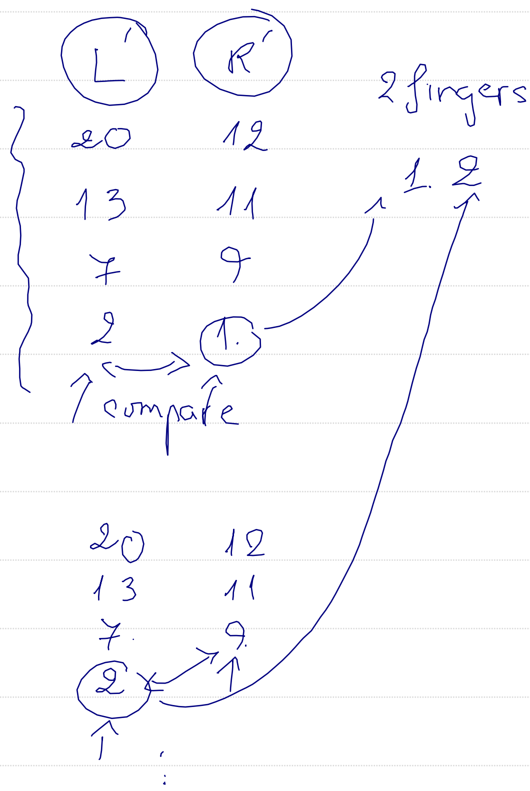
INSERTION SORT.



MERGE SORT :

Assume 2 sorted arrays as input.





2 fingers algo.