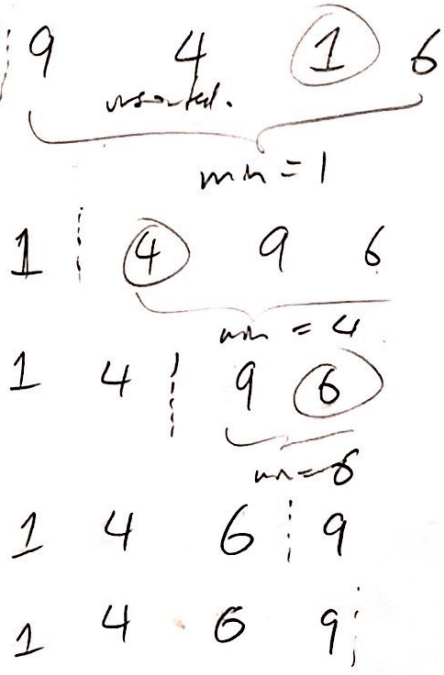
Eleni's
Sorts
Diagrams.

1. Selection Sort.

High level Concept
- Partition into Sorted / unsorted.
- find minimum element, in unsorted
- swaps with first unsorted, repeat.

ex 1.0)



Speed Analysis:

The most frequent and the expensive computation is running through the array to find the min.

As the unsorted Array gets smaller, the # of operations is as follows:

$= n + (n-1) + (n-2) \ldots + 1.$

by Gauss' formula.

$= \frac{n(n+1)}{2} = \frac{n^2+n}{2} = \boxed{O(n^2)}$

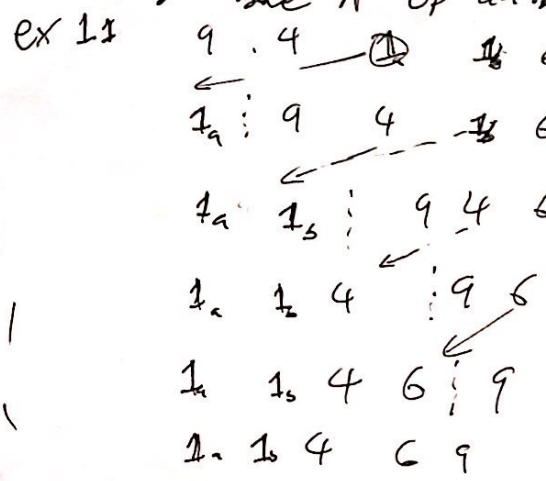
For Best, Ave & Worst Cases, since whole unsorted array must be traversed everytime, regardless of the array's sortedness.

Extension:

STABLE selection sort:

can be achieved by, instead of swapping w/ first unsorted value, just "slide" the lowest elem to the front, moving all the ones above it up an index.

ex 1.1



Definition:

a Stable sort is one where elements of equal "weight" are kept in the order they started in, always. In ex 1.1, both 1_a and 1_b are kept in their initial order.

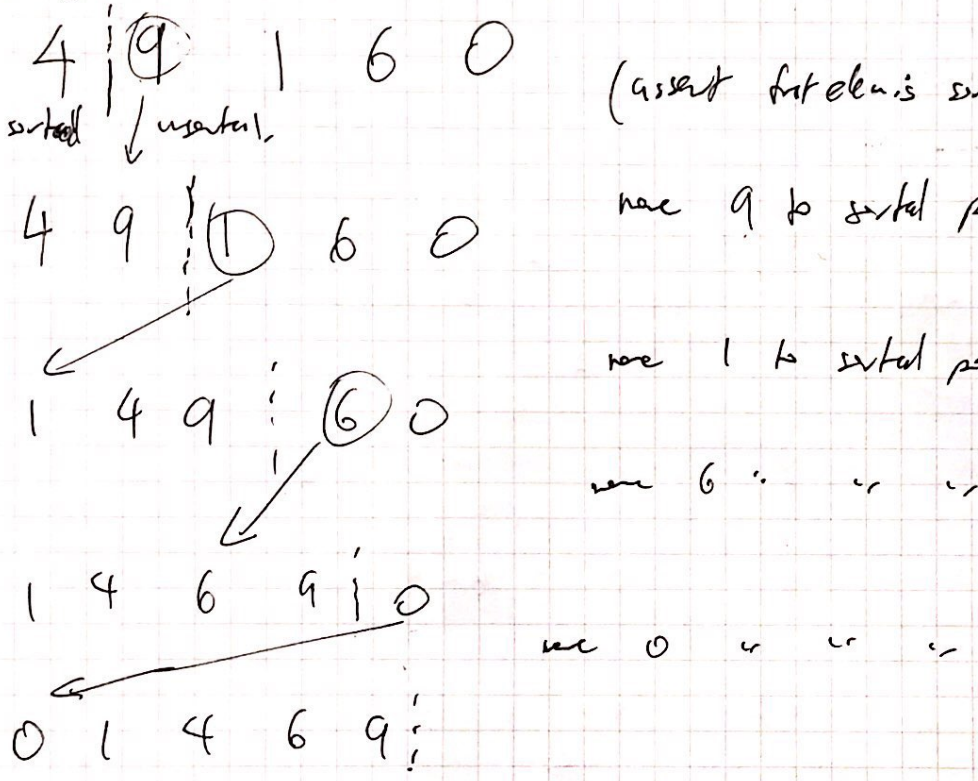
2) Insertion Sort.

High level Concept.

- partition into sorted / unsorted.
- choose first element in unsorted,
- move it down sorted until in correct location.

Loop until All Sorted.

Ex 2:



Speed Analysis:

BEST: $O(n)$

Suppose the Array was already sorted, say $\{0, 1, 4, 6, 9\}$, then only one comparison would have to be made to "sort" each chosen element. ex: $(0 \mid 4 \mid 6\ 9) \Rightarrow 6 > 4$.

WORST: $O(n^2)$

Array is backwards, ie $\{9, 6, 4, 1, 0\}$, then the # of speedies and comparisons $\leq 1 + 2 \ldots (n-1) + n = \frac{n^2+n}{2}$.

Average: $O(n^2)$

If random, each would require a decreasing $\frac{n}{2}$ comparisons, leading to $\frac{1}{2}$ worst $= \frac{1}{2}\left(\frac{n^2+n}{2}\right) = O(n^2)$

3) Merge BU (Bottom Up)

High level concept,

- Divide Array whole until has arrays of size 1.
- Merge the mini-arrays together, only looking at the first elem of each mini-array for each merge.

loop: work way BACK UP until sorted whole array

Ex 3.0: {4 6 9 1 3} sz: 4

{4 6} {9 1 3} sz: 2

{4} {6} {9} {1 3} sz: 1

merge 4<6. 9>1

{4, 6} {1, 9 3}

only check 1st elem: so comparisons so.

@<4

@<9

6<9

{1, 4, 6, 9 3}

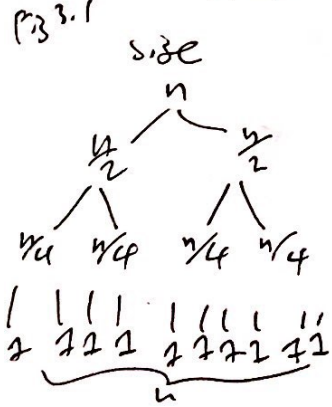
Speed Analysis.

Note: Regardless of pre-sorted Array state, all the same comparisons will be made, thus Best = Worst = Ave case.

BEST, WORST, AVE: $O(n \log n)$

the # of steps for each $\frac{1}{2}$ partition of the array (mini-array) takes n steps: demonstrated by Fig 3.1, to divide & to merge.

Fig 3.1



Lvl | # of steps.

$2 \cdot (\frac{n}{2}) = n$.

$4 \cdot (\frac{n}{4}) = n$

$n \cdot 1 = n$

Now, note by observation that for an array of size n, the number of levels is $\log_2 n + 1$ levels.

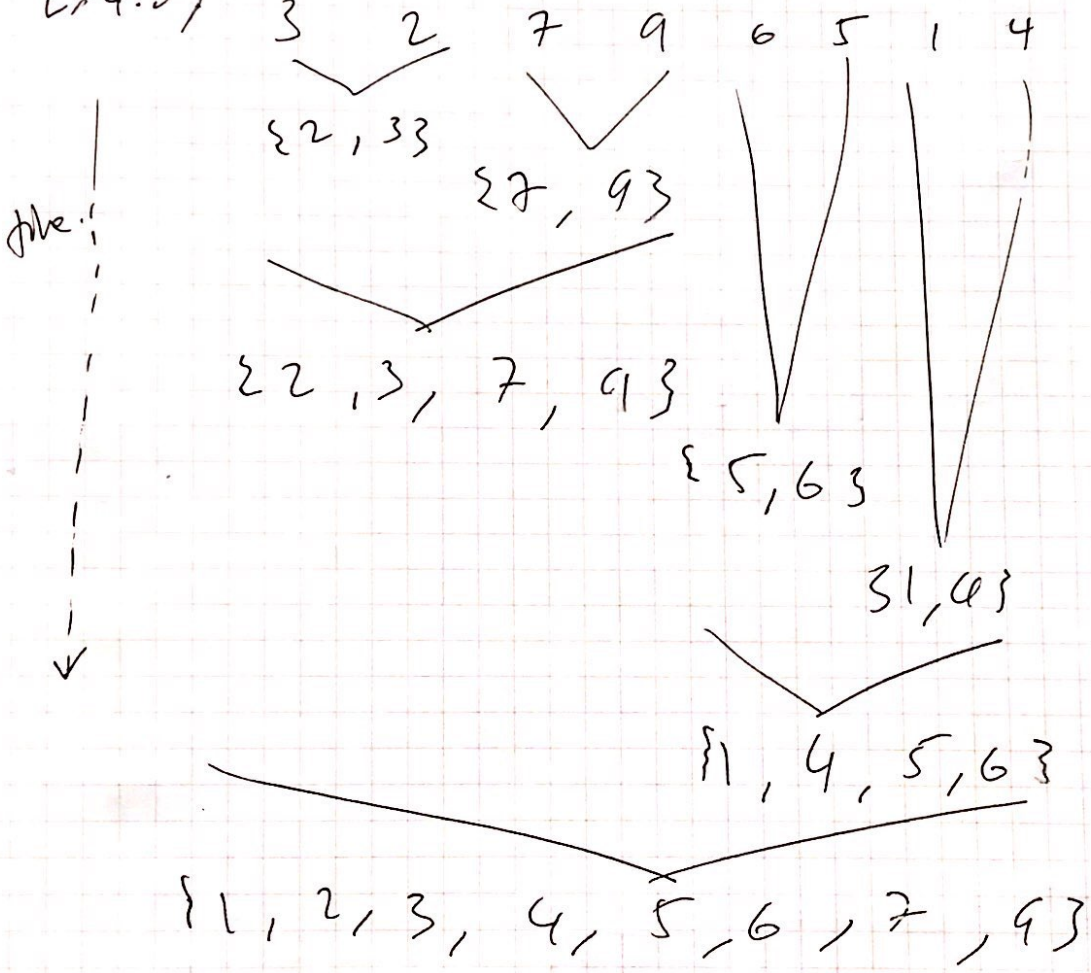
Thus, multiply both aggregates of # of levels and the (which is n) to yield.

$n(\log_2 n + n) = O(n \log n)$

4) Merge TD (Up Dam.)

High level concept:
- essentially the same as a Merge TD, except that we don't split everything first.
- Instead we iterate thru array & merge as needed.

Ex 4.0)



Analysis of Time: $O(n \log n)$

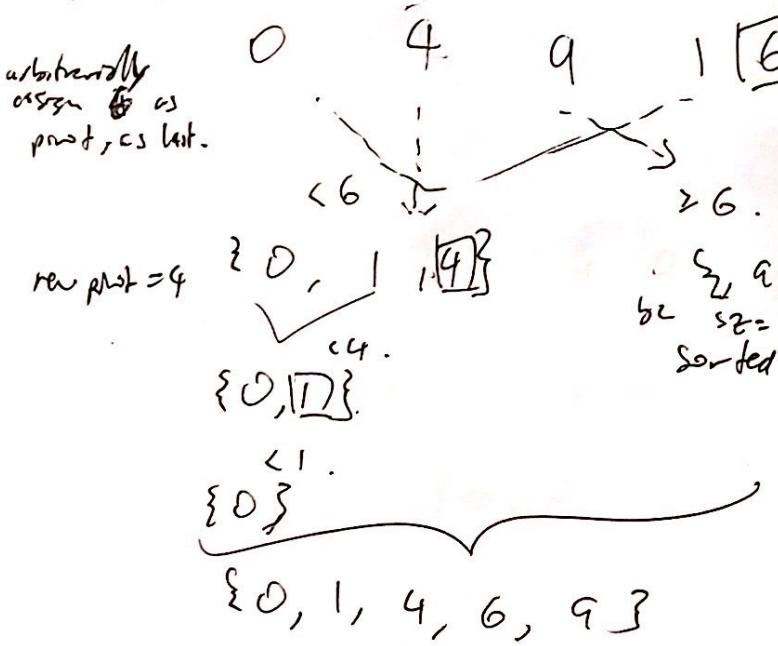
The logic is precisely the same as in Ex 3. For Merge Bu, only thus is in a different order. It is left as an exercise for the reader, as it is nearly trivial given 3).

5) Quick Sort.

High Level Concept:

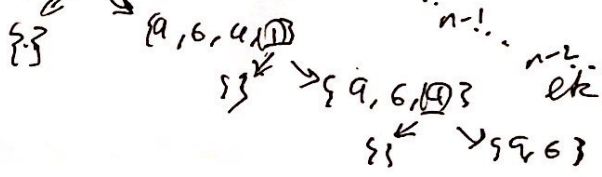
- choose a "pivot", which divides the array into
  - ↳ left of pivot: less than pivot.
  - ↳ Right of pivot: greater than pivot.
- Sort the left by above method,
- Sort the right by above method.

Ex 5.



Speed Analysis.

WORST: $O(n^2)$, if tree is Reversed, ie, {9, 6, 4, 1, 0}.
Then the following worst case occurs.. (Fig 5.1)

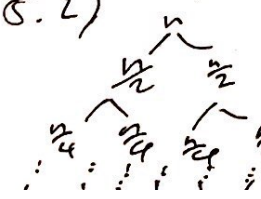


So the 5.1 case requires
$= n + (n-1) + (n-2) \dots 1.$
$= \frac{n^2+n}{2} = O(n^2)$.

AVG + BEST = $O(n \log n)$.

Both the Average and the Best are when the tree is "balanced" requiring a $\frac{1}{2}$ split every time, the diagram 5.2 is as follows.

Fig (5.2)



comps:
$2(\frac{n}{2}) = n$
$4(\frac{n}{4}) = n$
} # of $\log_2 n + 1$ times.
thus: $O(n \log n)$
when balanced.

6) Radix Sort.

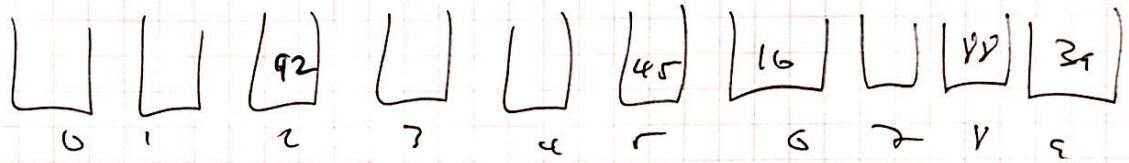
High Level Concept:
- Sort by digits starting w/ LSD. → MSD.
- "keep order".

Ex. 6.0)

{45 16 39 88 92}

Make "stacks" for each digit.

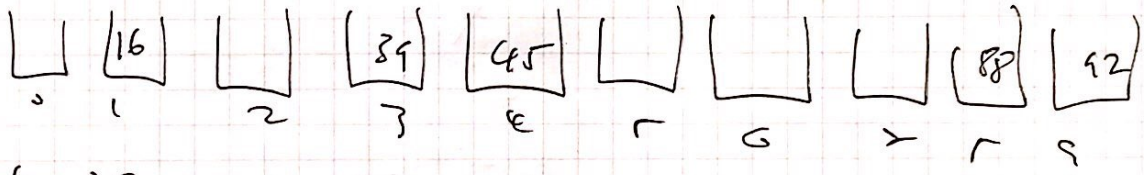
Use LSD.



Now Remove back into array in order

{92, 45, 16, 88, 39}.

Use Next Significant Digit.



{16, 39, 45, 88, 92}

Speed. Analysis:

Note, Radix sort is only an Integer sort not a comparison sort like those previously mentioned. Thus, it is not bounded to $O(n \log n)$.

By observation:

If n is the # of items to sort.
and w is the # of digits of the items,

then the speed is:

$O(nw)$.

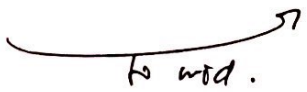
7) Binary Search.

High Level Concept.
- to search for something in already sorted array, look in the middle of array.
- compare to item.
- move by half of remaining array size.

Ex 7.0)

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

Find (8).



1, 2, 3, [4], 5, 6, 7, 8

8 > 4.

5. [6] 7, 8

8 > 6

[7], 8

8 > 7, 9.

[8] Found!

Speed Analysis:

Best. Found on first search, ie Σ 1, 3, [8], Found B.
Ω(1) Progress.

WORST: O(lg N).
proof: let any position of the Find Item be represented in Binary. Here 1 is a left, 0 is a right. ie. 0 1 1 0. These 4 moves would equate to possible combos, or 2^4. For any n "moves" 2^n possible places could be encoded. This give a N possible locations ie size, $\log_2 N$ moves at MAXIMUM can be used to find it.

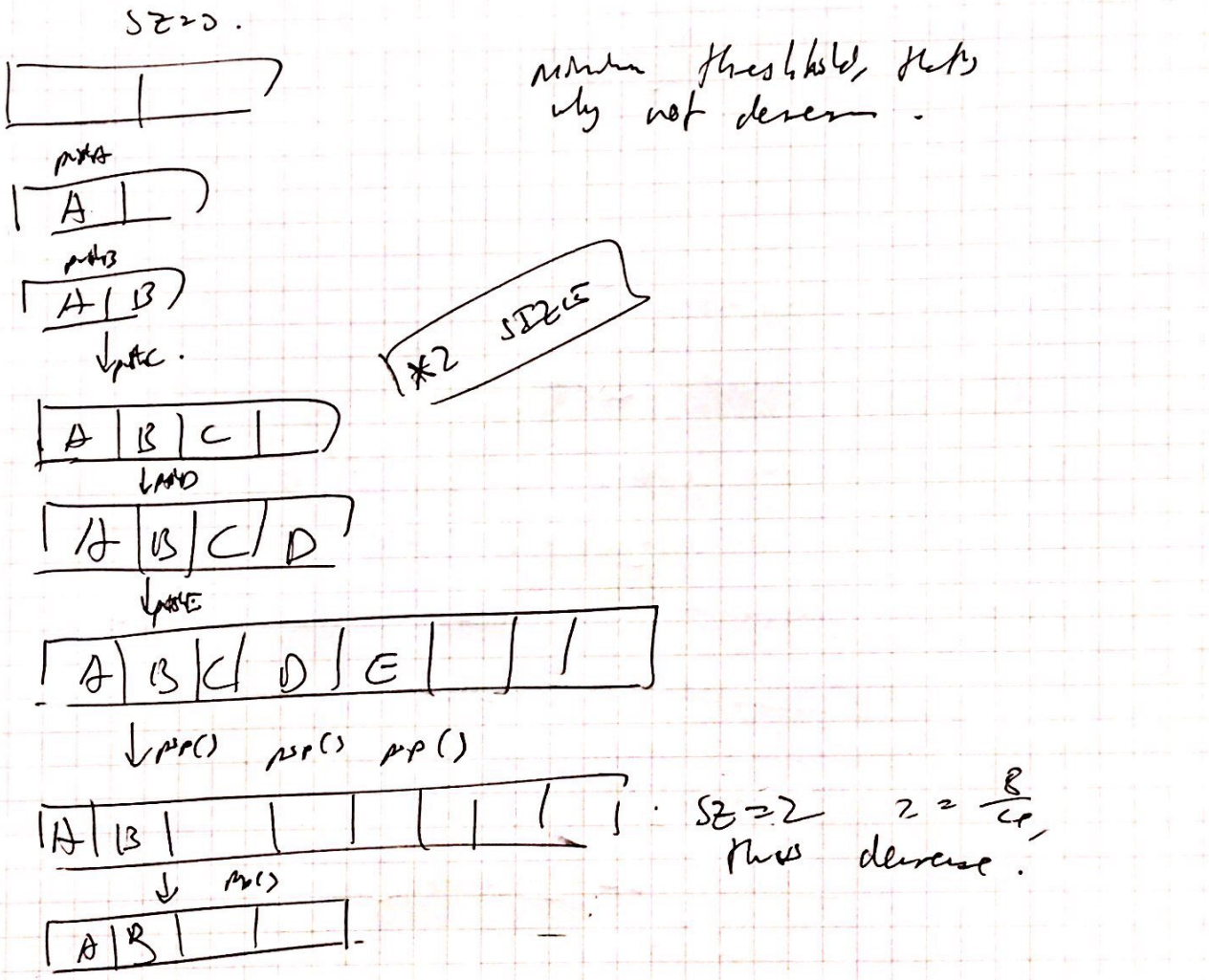
Ave: O(lg N),
by the previous reasoning, the average case will take approximately half of the worst case time, but still reads O(lg N).

8) resizable Array.

High level concept:

- create a resizable Array, but that changes size when:
  needs more space, Increase: ×2 size
  Can decrease: when the Arr elements is $\frac{1}{4}$ Arr size, then $\frac{1}{2}$ the size.

ex 8.



Maybe threshold, then why not decrease.

a) Birthday Problem.

- Create a random birthday rep.
- keep makes more via loop until a birthday has been repeated.
- check and calculate compared to:

Rearrange: $\sqrt{\frac{\pi N}{2}}$.

b) California.

(Alternate Alphabet ~~Sort~~)

- sort as a human would, ie starts with the first char & name, "the most significant char".
- Essentially, perform a MSD Radix, where, your precedence order is based on a precursored list of strings.

(arbitrary example).

"Queen vs Qwerty"

Q same

W < U, thus.

{Qwerty, Queen}