- Sorting data structure will help improve search times

1. Bubble sort

- Basic but inefficient
- compare adjacent elements, largest element "bubble" rightwards
- repeat until all sorted, reducing 1 check each run through
  (don't have to check "bubbled" largest elements

• $O(n^2)$: worst & avg case • $O(n)$: Best case (array already sorted)

- code implemented easily.

2. Selection Sort (most inefficient sorting)

Steps:

1. Find smallest element in array.
2. swap pos. of smallest element with first element.
3. Start with next idx, find smallest element, perform swap.
4. repeat until all sorted

left portion = sorted portion

• Average & worst case & Best case = $O(n^2)$

even with sorted array, need to shift sorted boundary, resulting in n^2

3. Insertion sort - like reverse bubble sort.

Pull out element, if smaller, propagate to front

Steps:

- Pull out 2nd element.
- compare with 1st element, if smaller, insert in front of element.
  else, insert back
- Pull out 3rd element.
- compare with 1st & 2nd element & sort accordingly.
- Pull out rest of the element & perform similar sorting.
  if no change required, insert back

Avg & worst case = $O(n^2)$ Best case = $O(n)$

Recursion

- fn calling itself until base case, else recursive case
  else, stack overflow
- After base case, propagate upwards back to origin

eg:

```
int sumBy3(int n, int x) {
    if (n <= 1)
        return x
    return sumBy3(n-3, x+n)
```

(1) n = 12, x = 0 — 30 returned
return sumBy3(9, 12)

(2) n = 9, x = 12 — 30
return sumBy3(6, 21)

(3) n = 6, x = 21 — 30
return sumBy3(3, 27)

(4) n = 3, x = 27 — 30
return sumBy3(0, 30)

(5) n = 0, x = 30 — 30
return x, x = 30

4. Quick sort

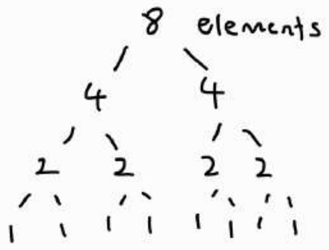
- Break array into subarrays (divide & conquer)

Steps:

1. choose leftmost element as pivot number.
2. shift all numbers < pivot left of pivot & > pivot to right
3. Repeat same trick for subarrays i.e left subtree
   right subtree
   └ until only single number remain, then it will be left most element of that tree.

Algorithm:

- Keep going down left subtree until single element
- Place center element
- traverse right subtree



deepest level = 3
if 16 elements, deepest = 4

- Best case : $O(n \log(n))$
  Avg case
- worst case: $O(n^2)$
  └ All numbers > pivot number causing no split

5. Merge Sort (divide & conquer) (stable & optimal)

steps:

1. Divide array into left half & right half (until 1 element)
2. Go up tree & recombine, while comparing values. (left cursor & right cursor), add smaller element to array.

Best, avg & worst = $O(n \log(n))$

Fastest comparison sort = $O(n \log(n))$
└ sorting by comparison

* There are sorting that does not use comparison

Stable vs Non stable

1 2_a 5 2_b 1 2_a 2_b 5 : stable (order preserved)

1 2_b 2_a 5 : unstable (only focus on values)

Stable
- Bubble
- Insertion
- Merge

Unstable
- Selection
- Quick

} order may be changed after sort

eg. | 2_a | 2_b | 1 | selection sort
↓
1 2_b 2_a done.

or if 2_b is pivot