→ Sorting algorithms.

→ Trees → {BST} ①
→ AVL

→ Heaps] Binary heap

→ Priority queues

→ Binomial queues

→ Hash maps

Sorting

- arrange items to
  - ascending
  - Descending order

In order to make operation more efficient.

Operations

- Changes to data { add items / Remove items
- Searching thro the data

9	14	32	38	-5	68

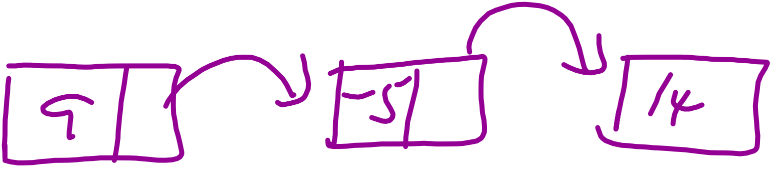
→ NOT sorted // unordered array

sorted array ≡ Ordered array

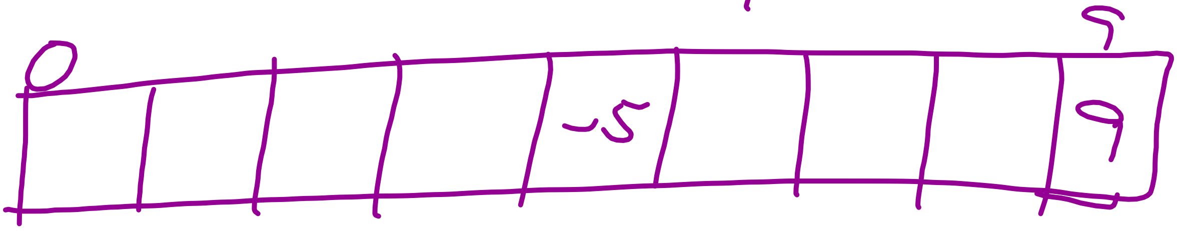
Array / Linked List

9 -5 14 35 19 . . .

an ordered collection of data



Associative array / Hash maps



9

Hash function =
Index = item % capacity

abs(-5) % cap

Un ordered collection of items

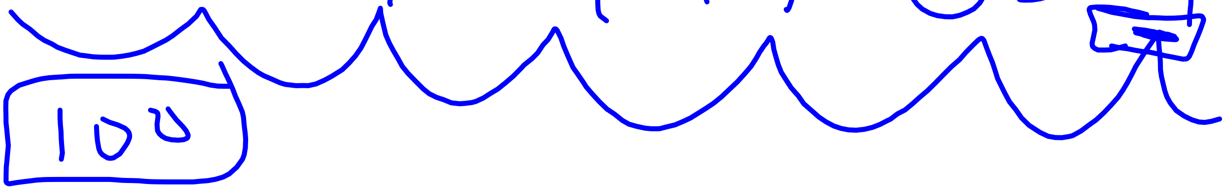
9 -5 14 68 17 5 71

Unordered array / unsorted array

O(1) O(n)

9 -5 14 68 17 5 71

-5 5 9 14 17 68 71



sorted } -1
ordered } O(n)

Best $\Theta(\quad)$

Worst $O(\quad)$

Average $\Theta(\quad)$

$\boxed{-5 \quad 5 \quad 17 \quad 19 \quad 6 \quad 8}$

- Binary searching
- Linear search $O(n)$

0 1 2 3 . . . . . 97, 98, 99

1) $\frac{n}{2^1} + \underline{i}$

2) $\frac{n}{4\ 2^2} + \underline{i + i}$ 12 - . 25

3) $\frac{n}{8\ 2^3} + \underline{i + i + i}$

4) $\frac{n}{16\ 2^4} + i + i + i$ 12 - - 18

5) $\frac{n}{\underline{32\ 2^5}} = \underline{\underline{15}}$ O(1)

if (n == m) return answer.

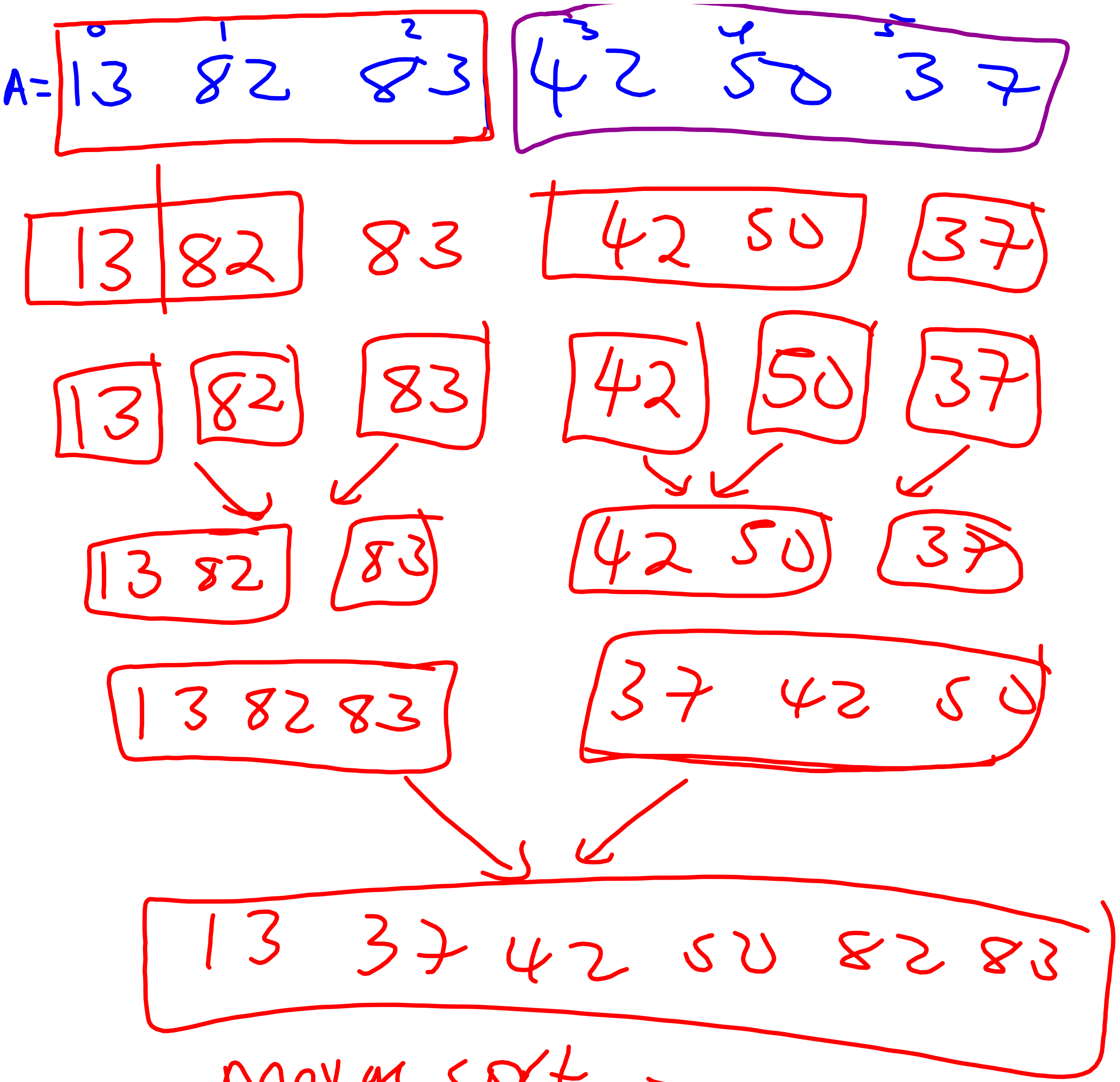
$$\frac{n}{\underline{32}} = \underline{1}$$

$$\frac{n}{2^k} = 1$$

$$\boxed{n = 2^k}$$

$$k = \log_2 n$$

Binary Search $\underline{\underline{O(\log n)}}$



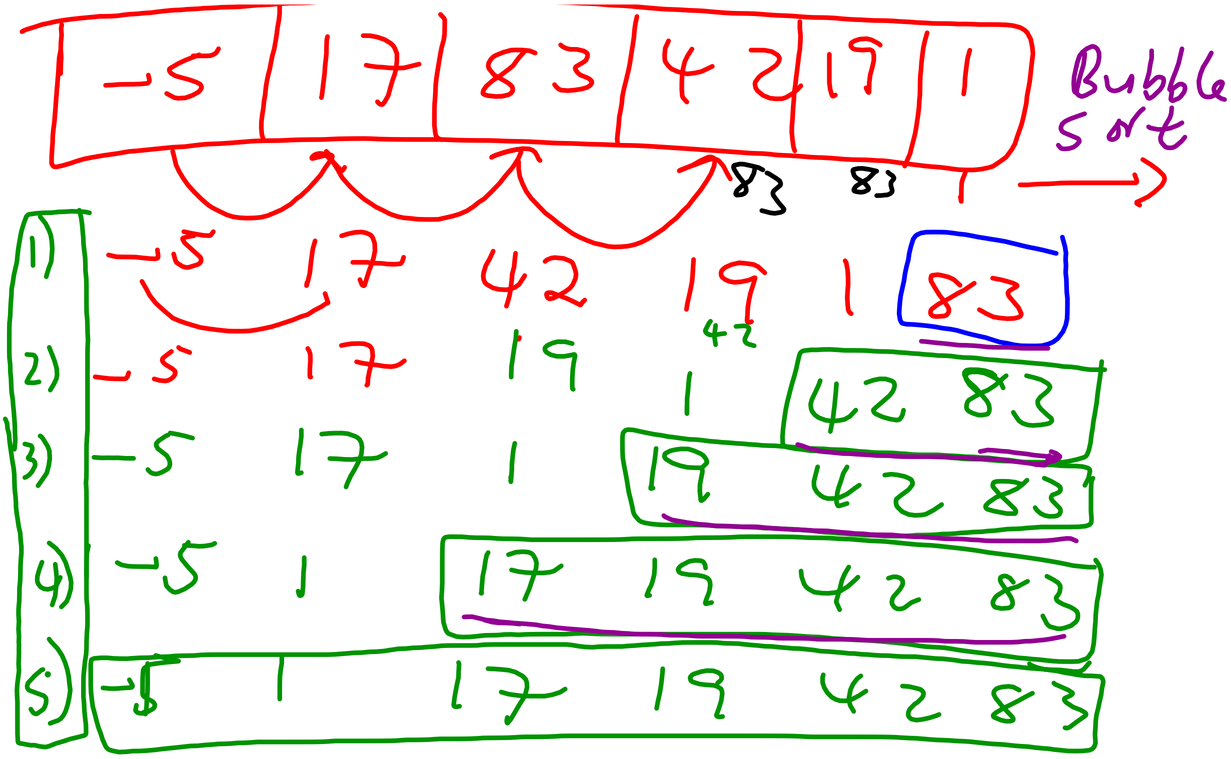
Merge sort ≡

n log n

- Repeatedly sub-dividing the array and merging these smaller arrays while sorting them.

Time complexity ⇒ $O(\log n)$

① Merge sort → Divide & Conquer



Bubble sort

-5 17 83 42 19 1

1) -5 17 42 19 1 83

2) -5 17 19 1 42 83

3) -5 17 1 19 42 83

4) -5 1 17 19 42 83

5) -5 1 17 19 42 83

```
for (int i=0; i<n ; ++i) {
    for (int j=0; j<n-1 ; j++) {
        if ( A[j] > A[j+1]) {
            int temp = A[j];
            A[j ] = A[j+1];
            A[j+1] = temp;
```

n x n

$O(n^2)$

3 3 3

Space complexity

Time complexity

1) Merge sort → $O(n(\log n))$

2) Bubble sort → $O(n^2)$

```
 0    1   2   3    4    5    6
24    3   80       50   89   -1

for (int j = 0; j < size; j++) {
    int index = j;
    for (int i = j; i < size; i++) {
        if (A[i] < A[index]) {
            index = i;
        }
        int temp = A[j]
        A[0] = A[index]
        A[index] = temp;
    }
```

$O(n^2)$

```
      24   3   80   50   89   -1
j=0    0   1    2    3    4    5
      -1   3   80   50   89   24
      -1   3   80   50   89   24
      -1   3   24   50   89   80
      -1   3   24   50   89   80
      -1   3   24   50   80   89
```

Selection sort

Insertion

24 -1 89 80 5 7

24

-1 24 89 80 5 7

-1 24 80 5 7 89

-1 80 89

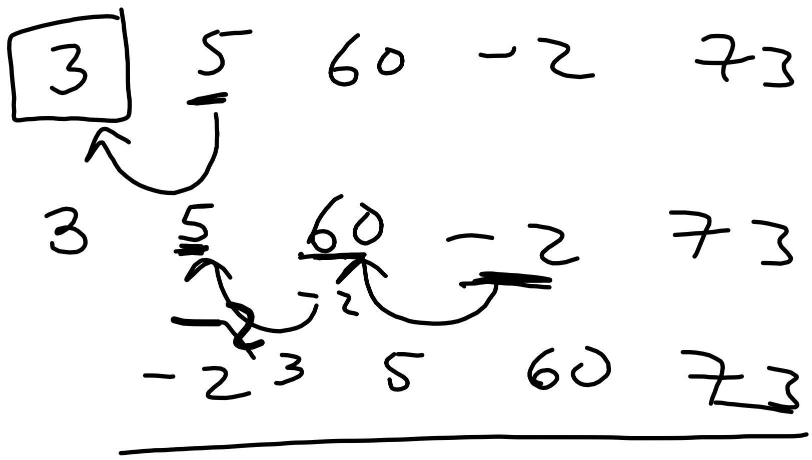
$\boxed{-5}$ 10 3 2 9

−5 10 $\underline{3}$ 2 9

−5 3 2 10 $\underline{2}$ 9

−5 2 3 $\underline{10}$ $\underline{9}$

−5 2 3 9 10



Insertion sort

Divides array into sorted & unsorted parts, & repeatedly selects an element from the ~~s~~ unsorted part & inserts it in its correct position in the sorted part

$\boxed{5}$ 3 -2 9 1 17

sorted ↑ unsorted part

sorted

3 5 -2 9 1 17

-2 3 5 9 1 17

-2 3 5 9 1 17

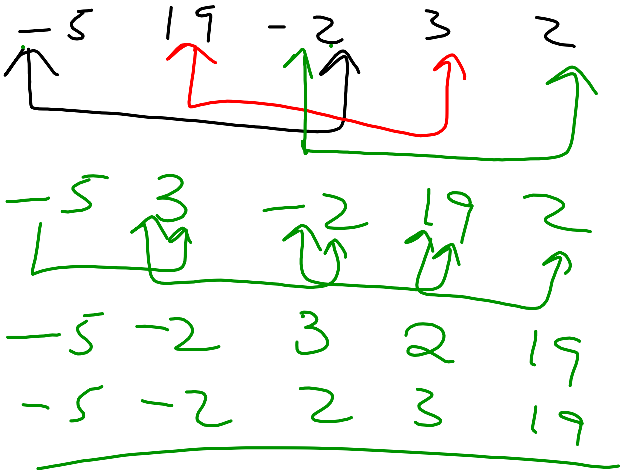
$\boxed{-2\ 1\ 3\ 5\ 9\ 17}$

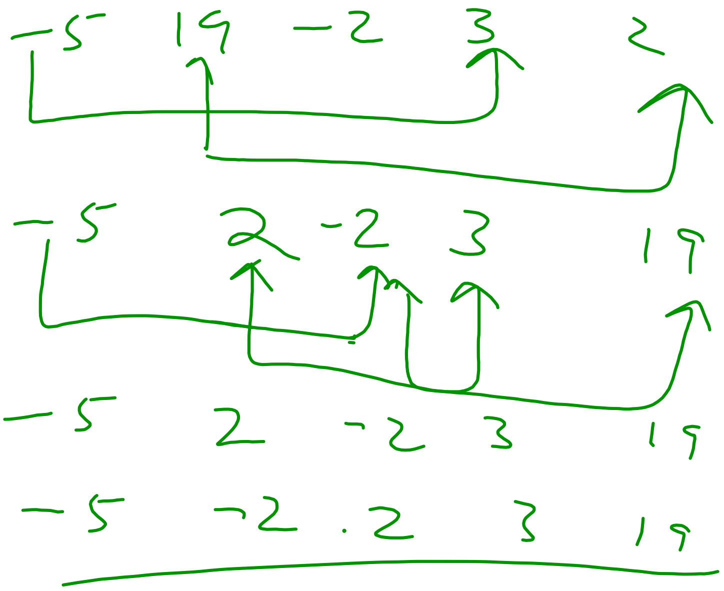
$O(n^2)$

Brute force $O(n^2)$

- Bubble sort
→ Insertion sort
→ Selection sort

Insertion sort





Shell sort

→ h sorting

divide the array into smalle arrays using a h-spacing and repeated decrease the spacing & finally carry out Insertion sorting