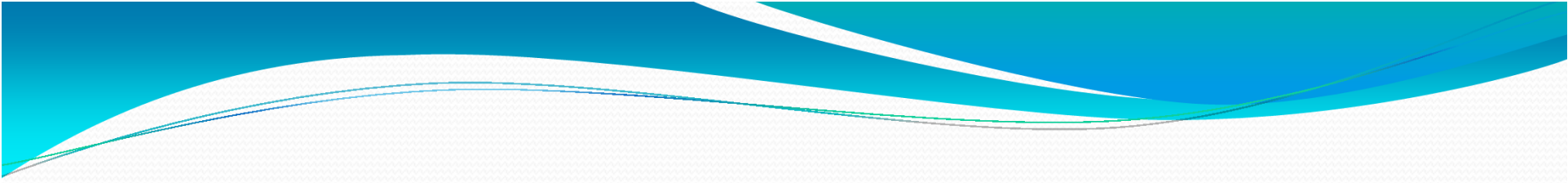


15-121

***Introduction to data structures -
Fast Sorting***

Ananda Gunawardena

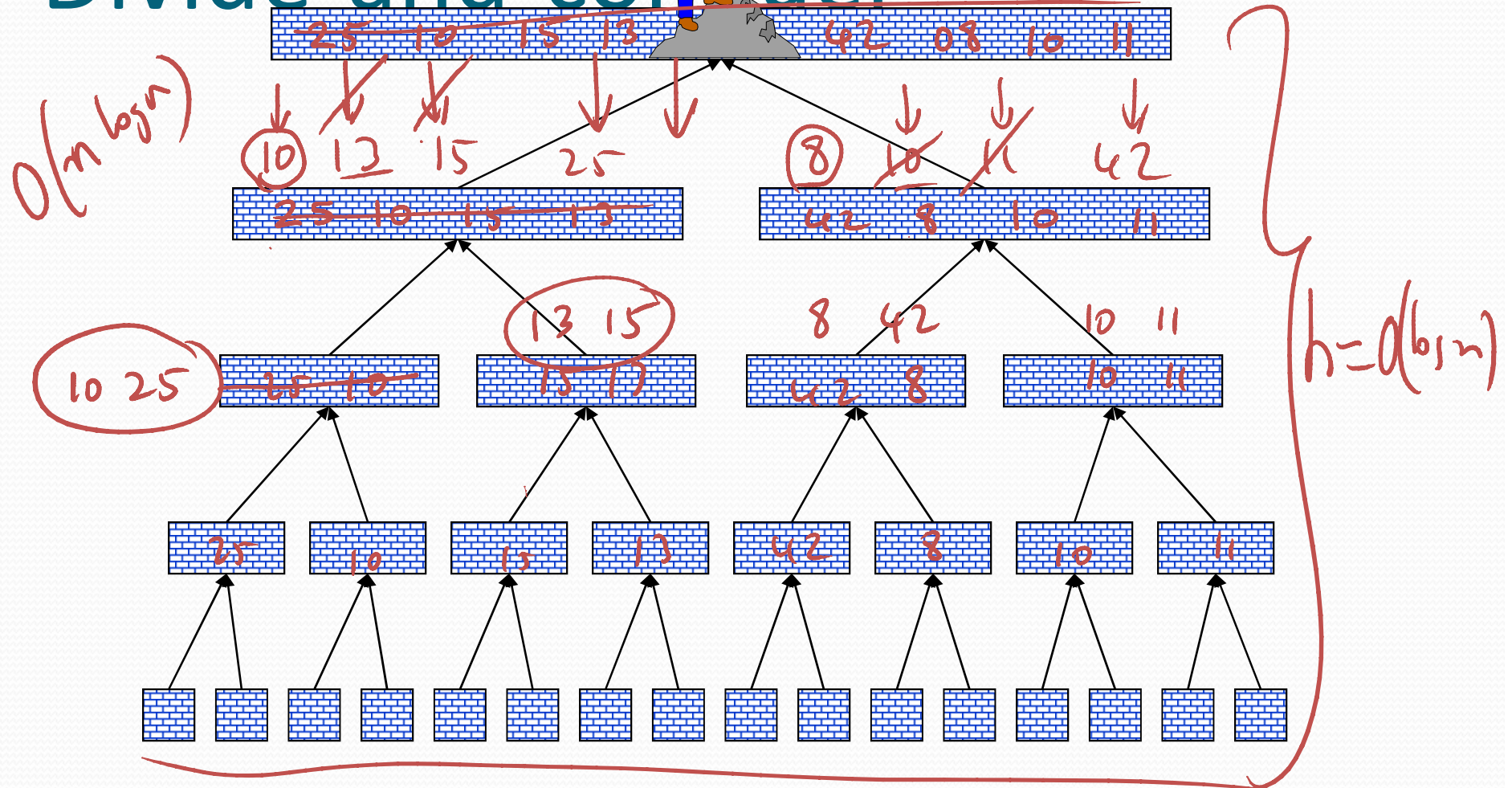


Merge Sort

8 10 10 11 13 15 25 42

Merge $\stackrel{?}{=} O(n)$

Divide-and-conquer



Merging Two Sorted Arrays

- All the work in merge sort is done at the merge step.
- Example

1	13	24	26
---	----	----	----

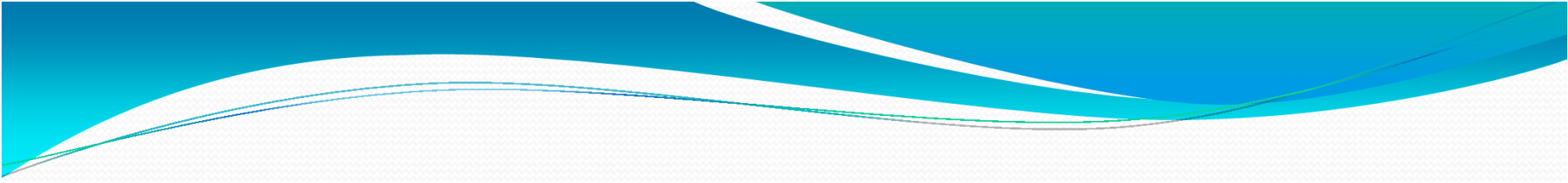
2	15	27	38
---	----	----	----

--	--	--	--	--	--	--	--

Why Merge sort is bad

→ Extra memory to merge

→



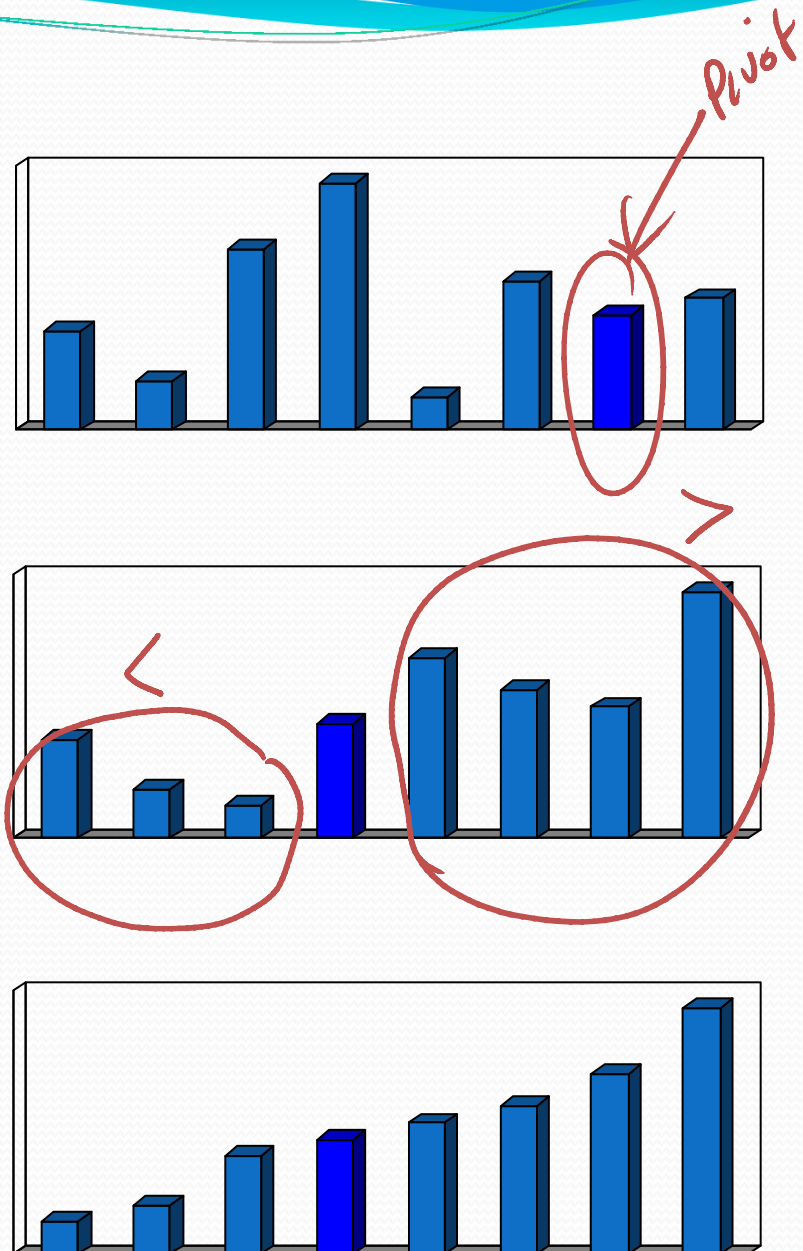
Quick Sort

Quicksort

- Quicksort was invented in 1960 by Tony Hoare.
- Quicksort has $O(N^2)$ worst-case performance, and on average $O(N \log N)$.
- *More importantly, it is the fastest known comparison-based sorting algorithm in practice.*

Quicksort idea

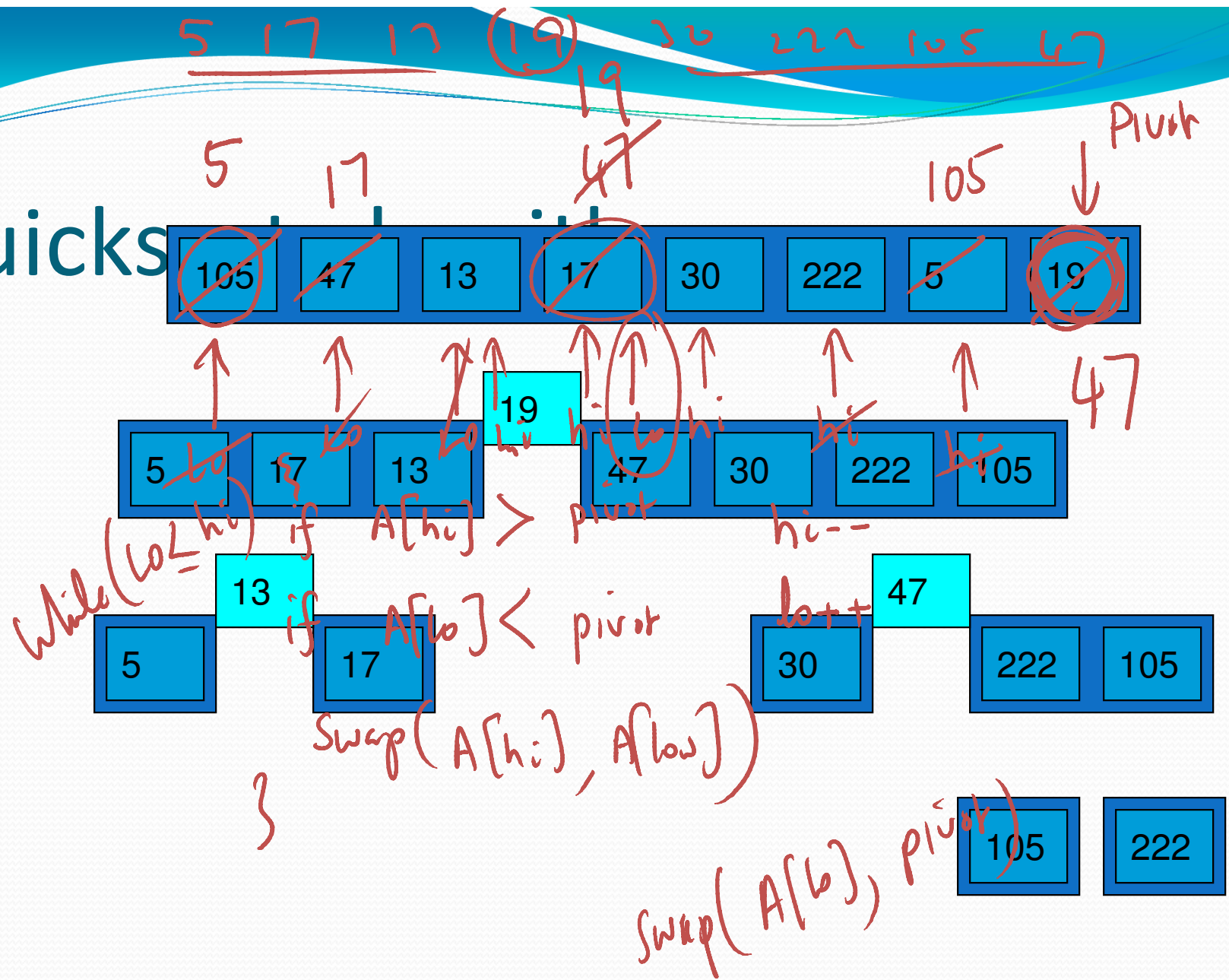
- Choose a pivot.
- Rearrange so that pivot is in the “right” spot.
- Recurse on each half and conquer!



Quicksort algorithm

- If array A has 1 (or 0) elements, then done.
- Choose a *pivot element* x from A.
- Divide A- $\{x\}$ into two arrays:
 - $B = \{y \in A \mid y \leq x\}$
 - $C = \{y \in A \mid y \geq x\}$
- Result is $B + \{x\} + C$.
- Recurse on arrays B and C

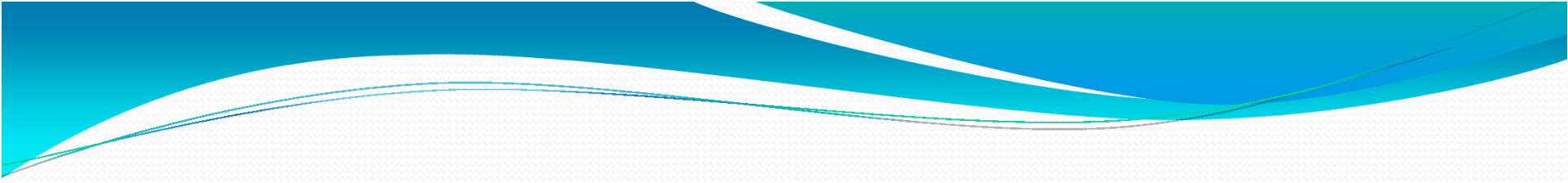
Quicksort



Place the pivot algorithm

- Assume we have an array of size n
- Pick a pivot x
- Place the pivot x in the last place of the array
- Have two pointers $i = 0, j = n-2$

```
while ( i <= j ) {  
    while (A[j] > pivot) j--;  
    while (A[i] < pivot) i--;  
    swap (A[i], A[j]);  
}  
swap (A[i], pivot);
```



Example

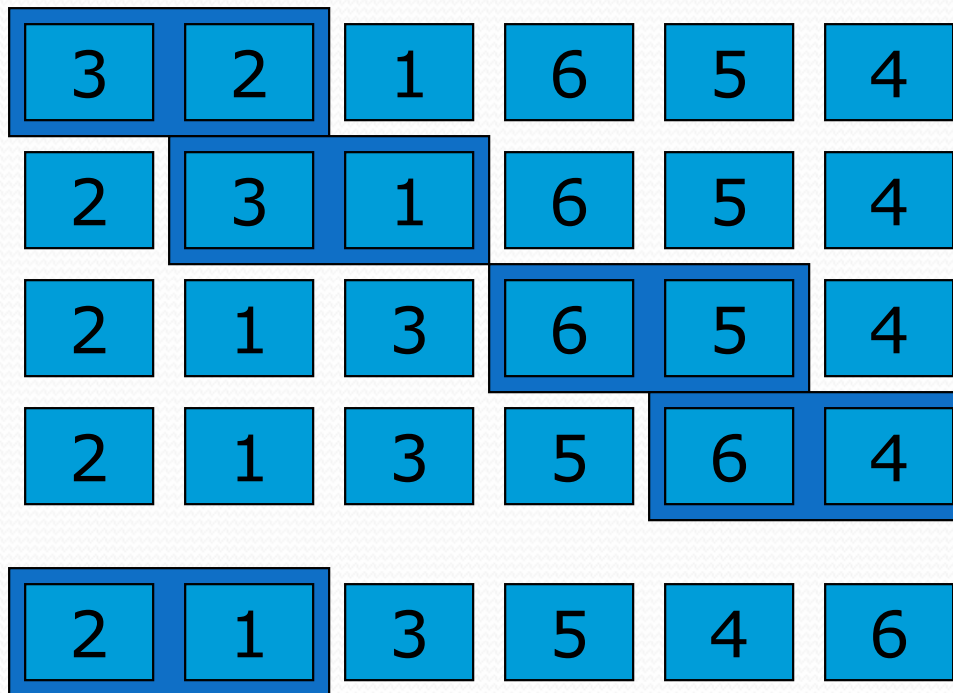
- Show how to place the pivot (choose middle element) in the right place
- 34 24 56 17 19 45 90 23 36

Survey of Sorting

Ananda Gunawardena

Naïve sorting algorithms

- Bubble sort: *scan for flips, until all are fixed*



Etc...

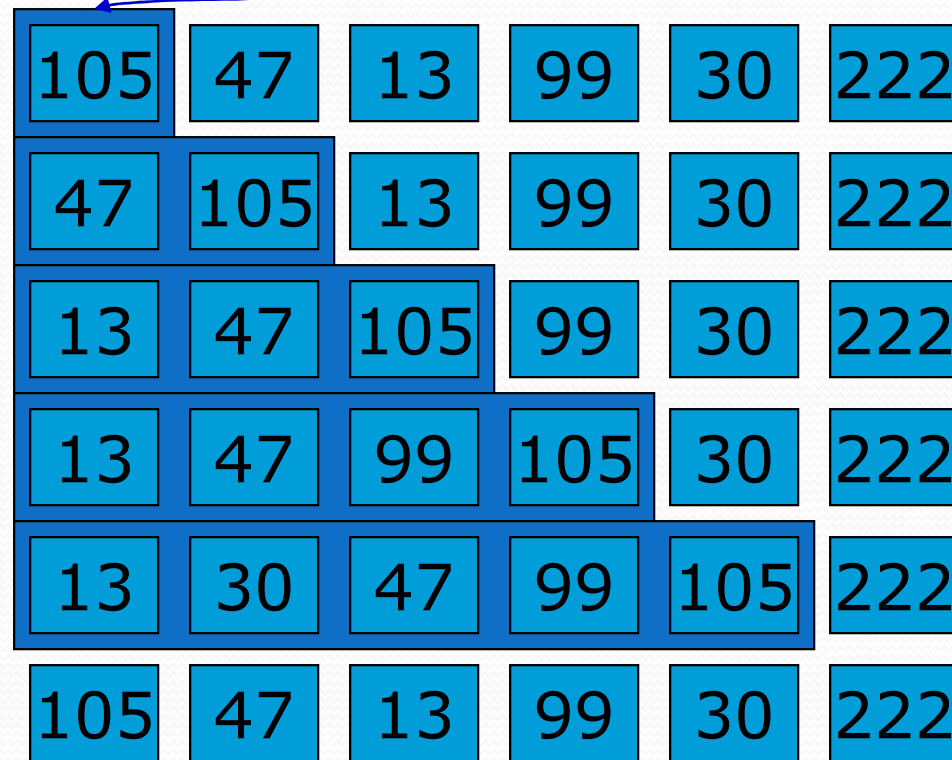
Naïve Sorting

```
for i=1 to n-1
  { for j=0 to n-i-1
    if (A[j].compareTo(A[j+1])>0)
      swap(A[j], A[j+1]);
    if (no swaps) break;
  }
```

- What happens if
 - All keys are equal?
 - Keys are sorted in reverse order?
 - Keys are sorted?
 - keys are randomly distributed?
- Exercise: Count the number of operations in bubble sort and find a Big O analysis for bubble sort

Insertion sort

Sorted subarray



105	47	13	99	30	222
47	105	13	99	30	222
13	47	105	99	30	222
13	47	99	105	30	222
13	30	47	99	105	222
105	47	13	99	30	222

Insertion sort

- **Algorithm**

```
for i = 1 to n-1 do
    insert a[i] in the proper place
    in a[0:i-1]
```

- **Correctness**

- Note: after i steps, the sub-array $A[0:i]$ is sorted

How fast is insertion sort?

To insert $a[i]$ into $a[0:i-1]$, **slide** all elements larger than $a[i]$ to the right.

```
tmp = a[i];  
for (j = i; j > 0 && a[j-1] > tmp; j--)  
    a[j] = a[j-1];  
a[j] = tmp;
```

of slides = $O(\text{\#inversions})$

very fast if array is nearly sorted to begin with

Selection sort

- Algorithm

```
for i = n-1 to 1 do
```

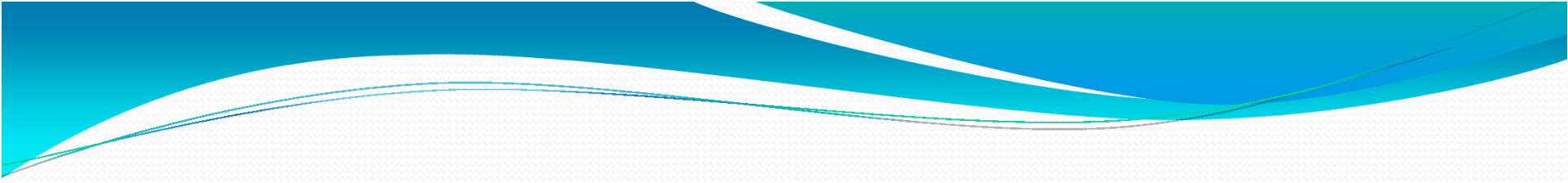
```
    Find the largest entry in the  
    in the subarray A[0:i]
```

```
    Swap with A[i]
```

What is the runtime complexity of selection sort?

Sorting Comparison

- Discuss the pros and cons of each of the naïve sorting algorithms



Advanced Sorting

Quick Sort

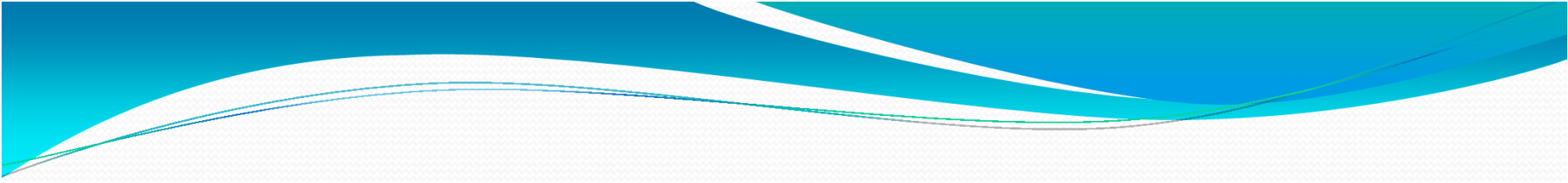
- Fastest algorithm in practice
- Algorithm
 - Find a pivot
 - Move all elements smaller than pivot to left
 - Move all elements bigger than pivot to right
 - Recursively sort each half
 - $O(n \log n)$ algorithm

Merge Sort

- Divide the array into two equal halves
- Divide each half recursively until each array is of size 1
- Merge two (sorted) arrays of size 1
- Complete the process recursively

Heap Sort

- Build a max heap
- Delete Max (attach to end of array) until heap is empty
- Resulting array is sorted
- Complexity

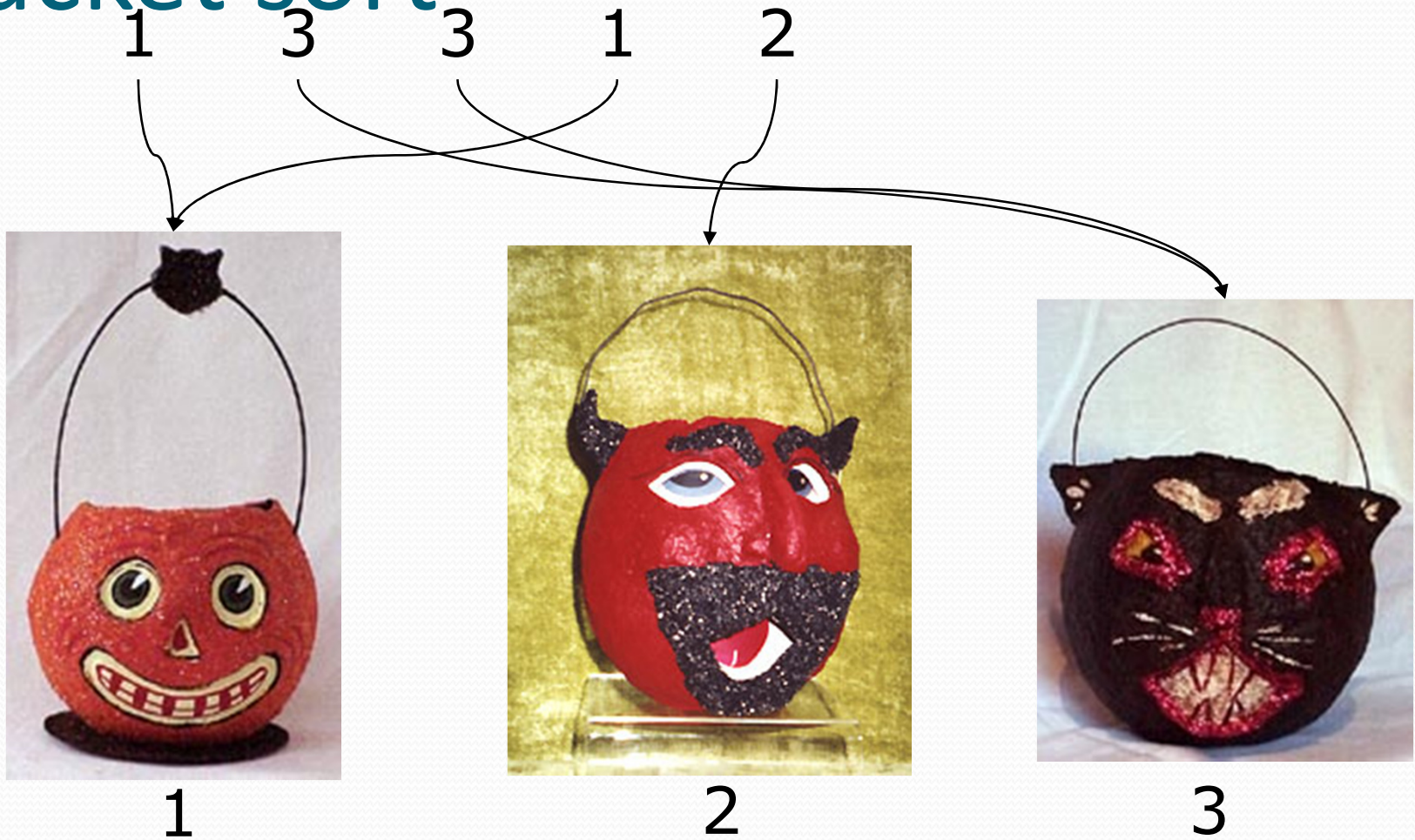


Bucket Sort

Bucket sort

- In addition to comparing pairs of elements, we require these additional restrictions:
 - all elements are non-negative integers
 - all elements are less than a predetermined maximum value
- Elements are usually keys paired with other data

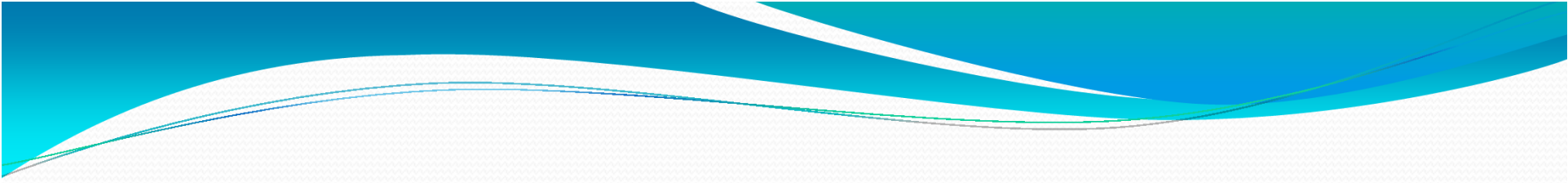
Bucket sort





Bucket sort characteristics

- Runs in $O(N)$ time.
- Easy to implement each bucket as a linked list.
- Is *stable*:
 - If two elements (A,B) are equal with respect to sorting, and they appear in the input in order (A,B), then they remain in the same order in the output.



Radix Sort



Radix sort

- If your integers are in a larger range then do bucket sort on **each digit**
- Start by sorting with the **low-order digit** using a **STABLE** bucket sort.
- Then, do the next-lowest, and so on

Radix sort

- Example:

2	0	1	0	0	1	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	1	0	0	1	1
5	1	0	1	1	0	0	1	0	1	0	1	0
1	0	0	1	1	1	1	0	1	0	1	1	1
7	1	1	1	1	0	1	1	0	1	0	0	0
3	0	1	1	0	0	1	1	1	1	0	1	1
4	1	0	0	1	1	1	1	1	1	1	0	1
6	1	1	0	0	1	1	1	1	1	1	1	1

Each sorting step must be stable.



Radix sort characteristics

- Each sorting step can be performed via bucket sort, and is thus $O(N)$.
- If the numbers are all b bits long, then there are b sorting steps.
- Hence, radix sort is $O(bN)$.

What about non-binary?

- Radix sort can be used for decimal numbers and alphanumeric strings.

0	3	2
2	2	4
0	1	6
0	1	5
0	3	1
1	6	9
1	2	3
2	5	2

0	3	1
0	3	2
2	5	2
1	2	3
2	2	4
0	1	5
0	1	6
1	6	9

0	1	5
0	1	6
1	2	3
2	2	4
0	3	1
0	3	2
2	5	2
1	6	9

0	1	5
0	1	6
0	3	1
0	3	2
1	2	3
1	6	9
2	2	4
2	5	2