Input Enhancement

➪ Counting sorts

Premise: if we know how many are smaller, we know where in the sorted list to place a key.

0	1	2	3	4	5
62	31	84	96	19	47
3	1	4	5	0	2

Counts (colored) may directly index the sorted list (e.g., 19 in sorted position 0).

If the extra info was available, we could end up with linear time sorting!

How many elements are smaller?

- ✎ How fast can list be sorted?
- ✎ At what cost?

Comparison Counting

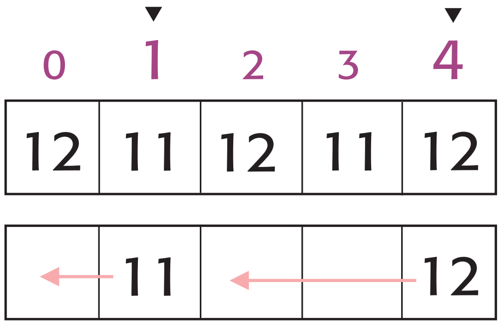
Exercise
Determine the efficiency. What would be a good basic operation to count (i.e, <u>cite as basic op for reporting efficiency</u>)?

Algorithm $ComparisonCountSort$
Input $A[0..n-1]$
Output Sorted elements of A in $S[0..n-1]$

for $i \leftarrow 0$ **to** $n-1$ **do** $Count[i] \leftarrow 0$
for $i \leftarrow 0$ **to** $n-2$ **do**
    for $j \leftarrow i+1$ **to** $n-1$ **do**
        if $A[i] < A[j]$ **then**
            $Count[j] \leftarrow Count[j] + 1$
        else $Count[i] \leftarrow Count[i] + 1$
for $i \leftarrow 0$ **to** $n-1$ **do** $S[Count[i]] \leftarrow A[i]$
return S

Efficiency?

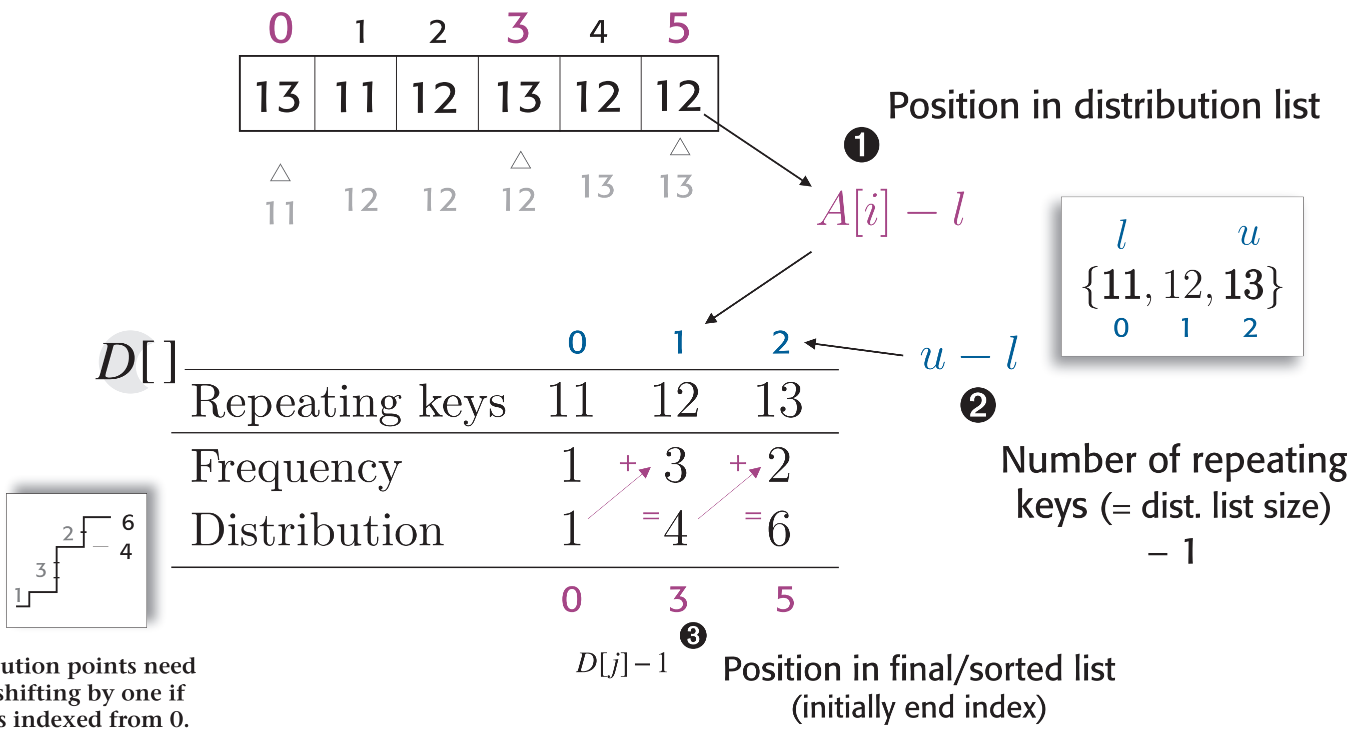
Input Enhancement
Sorting Repeated Keys



It is easy to build the sorted array *if* we can efficiently calculate key frequencies and distribution (i.e., gather some statistical info about the keys).

- ⇨ **How many? (= key frequency)**
- ⇨ **End index (= distribution)**

Input Enhancement
Distribution Counting



$D[\,]$	0	1	2
Repeating keys	11	12	13
Frequency	1	+ 3	+ 2
Distribution	1	= 4	= 6
	0	3	5

Distribution points need down shifting by one if array is indexed from 0.

Distribution Counting Algorithm

Exercise
Determine efficiency if keys are distinct (not repeating).

Algorithm *DistributionCounting*
Input $A[0..n-1]$, l, u (where $l \le A[i] \le u$ for $0 \le i \le n-1$)
Output Sorted elements of A in $S[0..n-1]$

Programming 101, figure it out.

```
1: for j ← 0 to u − l do D[j] ← 0          ❷
2: for i ← 0 to n − 1 do D[A[i] − l] ++     ❶
3: for j ← 1 to u − l do
4:      D[j] ← D[j − 1] + D[j]

5: for i ← n − 1 downto 0 do
6:      j ← A[i] − l
7:      S[D[j] − 1] ← A[i]                  ❸
8:      D[j] −−

9: return S
```

Exercise
Trace steps 5-8 for the example from previous slide. Write a descriptive comment for each step. *An example last slide.*

Efficiency?

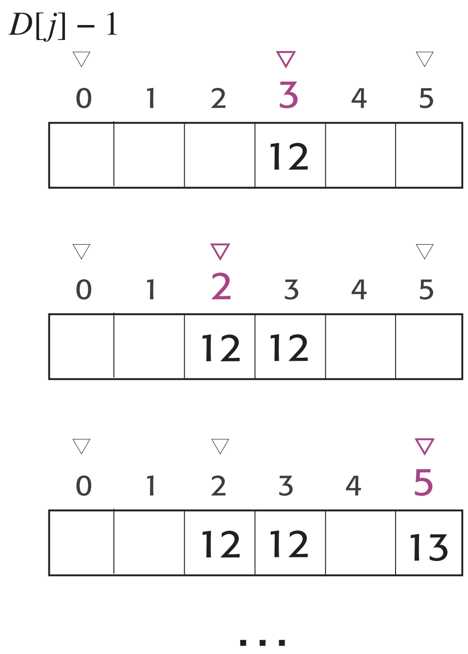
Distribution Counting
Example Walkthrough

```
for ...
    j ← A[i] − l
    S[D[j] − 1] ← A[i]
    D[j] −−
```

13	11	12	13	12	12
0	1	2	3	4	5

← i

	0	1	2
Repeating keys	11	12	13
Frequencies	1	3	2
Distribution	1	4	6
	1	3	6
	1	2	6
	1	2	5
		…	



Exercise
Repeat the trace for 13, 10, 15, 13, 15, 15. Suggest a fix for memory waste. Correct the descriptive label for parameter 2 (previous slides).

Trading Space for Time

➪ **Preconditioning**
➪ **?Preprocessing**

➪ Input enhancement

How different from presorting?

➪ Prestructuring: faster access

Quiz
Which style of trading space for time is used by each application?

➪ Important applications

String matching, hash tables, B-trees

. . .
▷ determine entry in distribution table
▷ insert key (initially last sorted appearance)
▷ update next sorted position (if encountered later)