LINEAR-TIME SORTING

Hybrid Quicksort

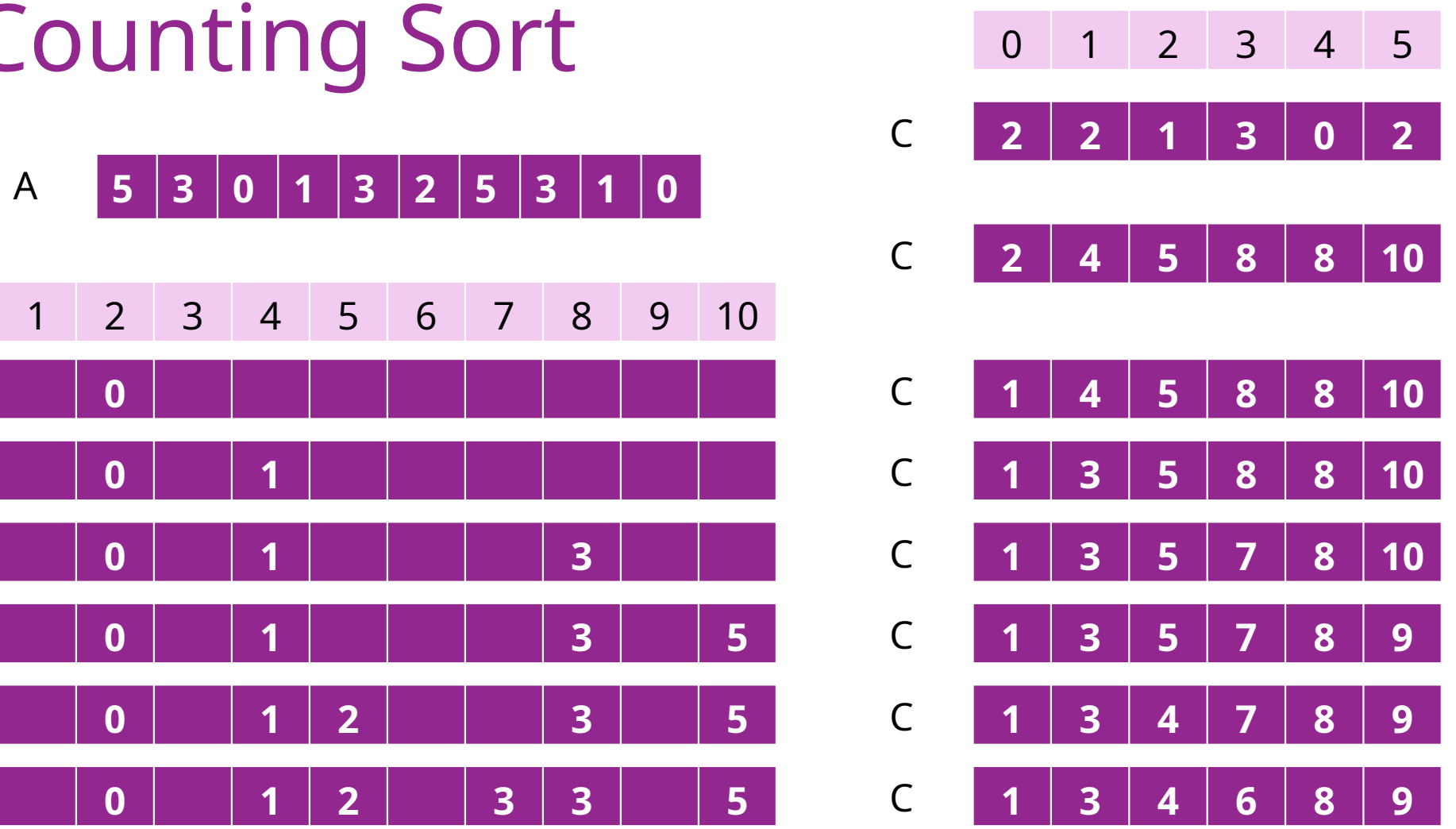
What is the time complexity of quick sort where if size smaller than S insertion sort is applied to the subset?

Hybrid Quicksort

What is the time complexity of quick sort where if size smaller than S insertion sort is applied to the subset?

- Parititioning to S sized subsets:
- Sorting each subset:
- Total cost:

Counting Sort



Counting Sort

A

5	3	0	1	3	2	5	3	1	0

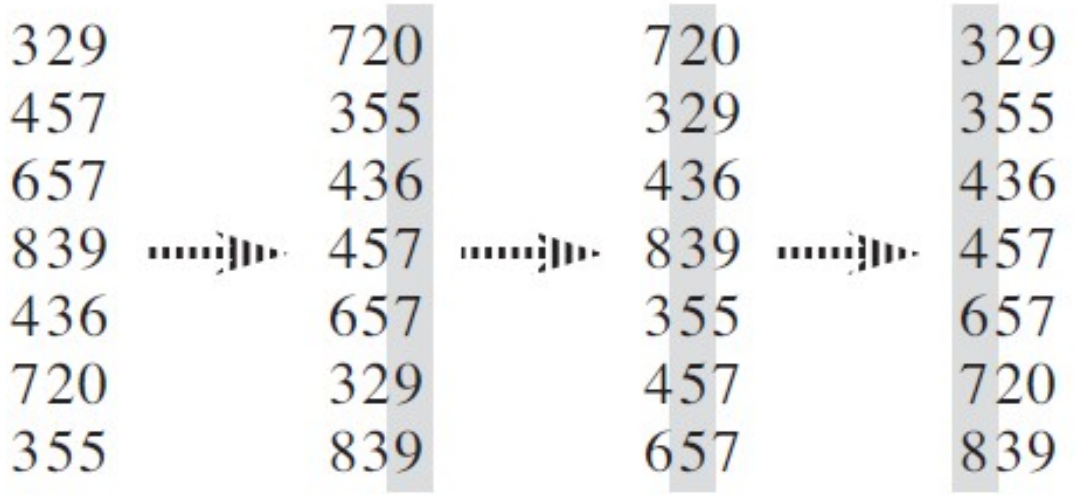
	1	2	3	4	5	6	7	8	9	10
B		0		1	2		3	3		5
B		0	1	1	2		3	3		5
B	0	0	1	1	2		3	3		5
B	0	0	1	1	2	3	3	3		5
B	0	0	1	1	2	3	3	3	5	5

	0	1	2	3	4	5
C	1	3	4	6	8	9
C	1	2	4	6	8	9
C	0	2	4	6	8	9
C	0	2	4	5	8	9
C	0	2	4	5	8	8

Counting Sort

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k                                   Θ(k)
 3      C[i] = 0
 4  for j = 1 to A.length                            Θ(n)
 5      C[A[j]] = C[A[j]] + 1
 6  // C[i] now contains the number of elements equal to i.
 7  for i = 1 to k                                   Θ(k)
 8      C[i] = C[i] + C[i − 1]
 9  // C[i] now contains the number of elements less than or equal to i.
10  for j = A.length downto 1                        Θ(n)
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] − 1
```

Radix Sort



What would be if we start from MSD-to-LSD rather than LSD-to-MSD?

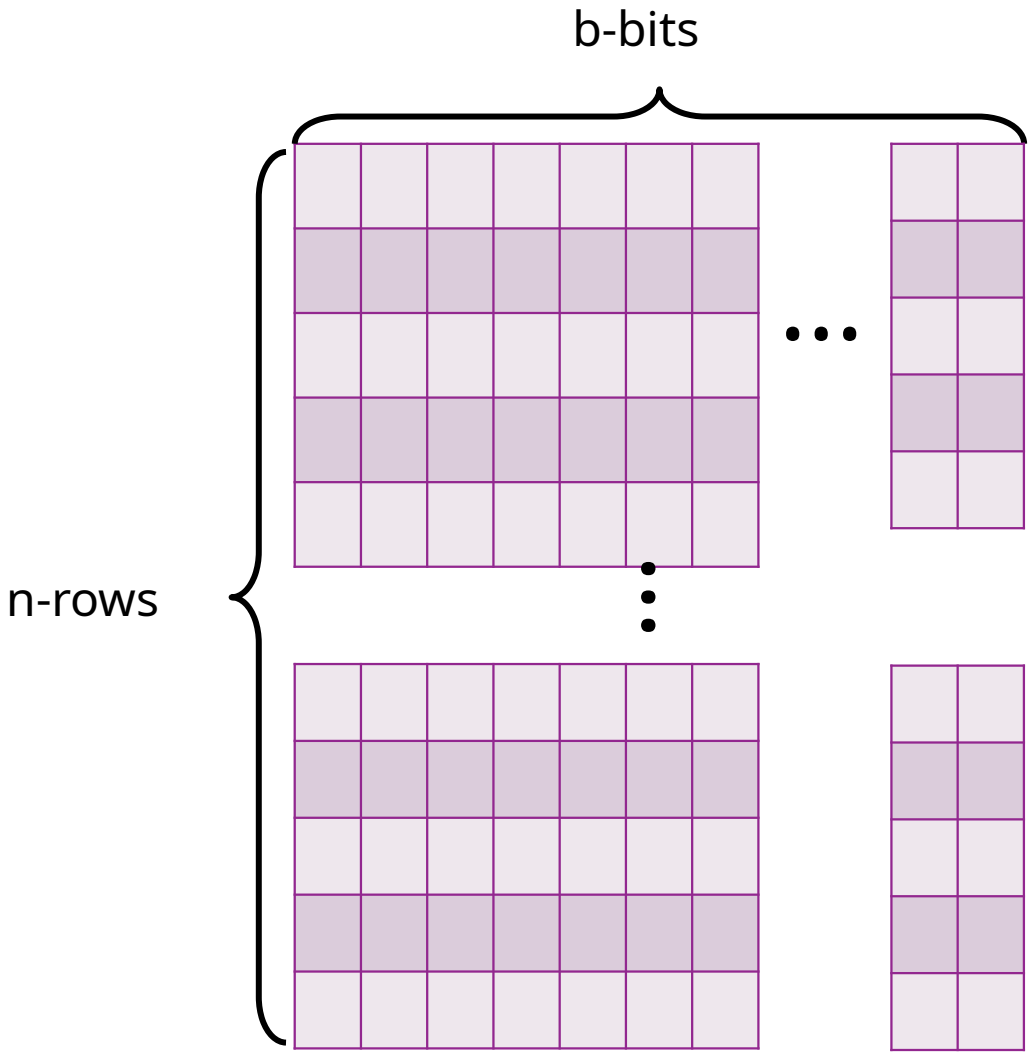
RADIX-SORT(A, d)

1 **for** $i = 1$ **to** d

2 use a stable sort to sort array A on digit i $\Theta(n+k)$

Radix Sort

- For a given n b-bit number, how we should divide numbers to r-bits?
- What would be complexity if we choose r=b or r=1? What would be the optimal value for r?



Radix Sort

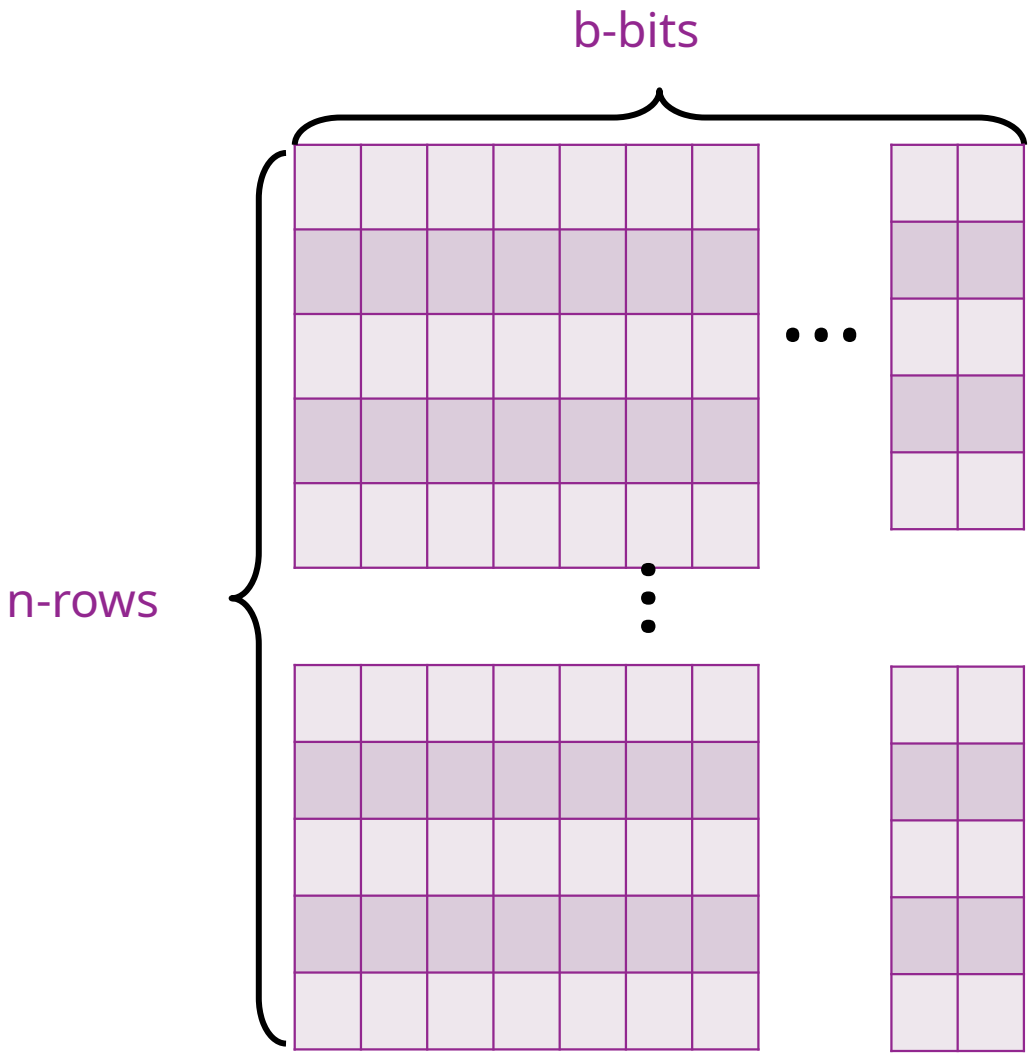
General formula for comlexity becomes;

(

For r=b ()

For r=1 (n)

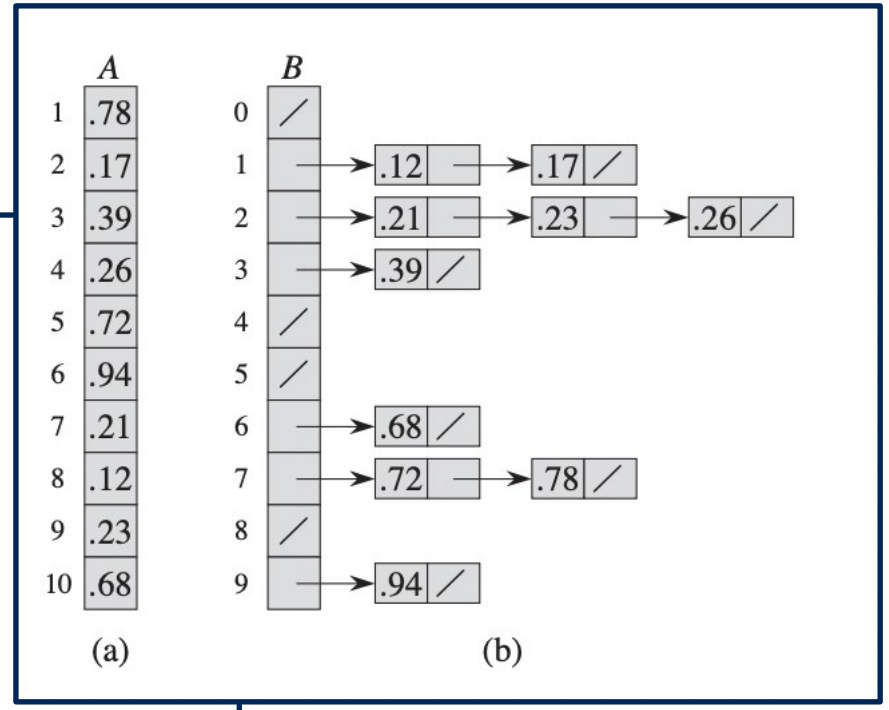
For r≈log(n) ()



Bucket Sort

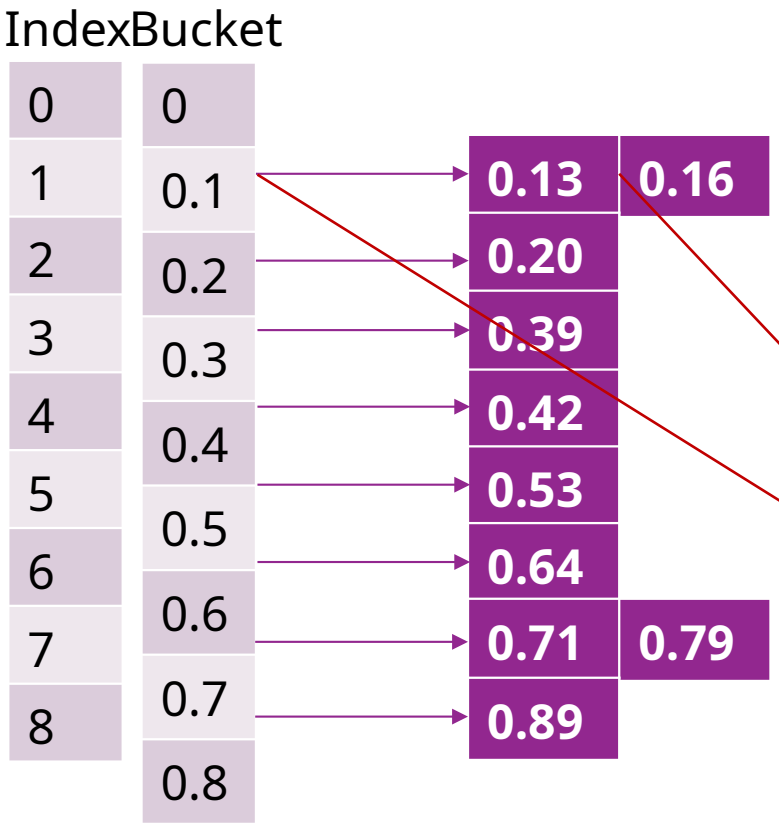
- Assumptions: Uniform distribution over [0, 1)

```
BUCKET-SORT(A)
1  let B[0..n − 1] be a new array
2  n = A.length
3  for i = 0 to n − 1
4       make B[i] an empty list
5  for i = 1 to n
6       insert A[i] into list B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i] with insertion sort
9  concatenate the lists B[0], B[1], . . . , B[n − 1] together in order
```



Bucket Sort

- Illustration of the BUCKET-SORT operation on the array
  - {0.79; 0.13; 0.16; 0.64; 0.39; 0.20; 0.89; 0.53; 0.71; 0.42}



- Step 1: Form sublists
- Step 2: Sort and combine sublists with insertion sort

Index of 0.13:
I = Floor(0.13*10)
I = Floor(1.3)
I = 1

Bucket Sort

Explain why the worst-case running time for bucket sort is . What simple change to the algorithm preserves its linear average-case running time and makes its worst-case running time ?

Bucket Sort

Explain why the worst-case running time for bucket sort is . What simple change to the algorithm preserves its linear average-case running time and makes its worst-case running time ?

- Worst Case: Bucket which contains all n values of the array.
- Since insertion sort has worst case running time , so does Bucket sort.
- We can avoid this by using merge sort to sort each bucket instead, which has worst case running time .

Can we do better?