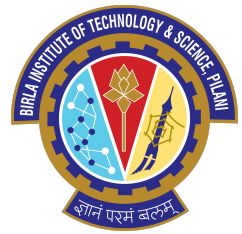
CS F364: Design & Analysis of Algorithms

Lecture-kt06: Sorting, Lower Bounds and Linear Time Solution



Dr. Kamlesh Tiwari,
Assistant Professor,
Department of Computer Science and Information Systems,
BITS Pilani, Rajasthan-333031 INDIA

Feb 2, 2017 (Campus @ BITS-Pilani Jan-May 2017)

Recap: Analysis of Randomized Quick Sort

```
RANDOMIZED-QUICKSORT(A, p, r)
1  if p < r
2      q = RANDOMIZED-PARTITION(A, p, r)
3      RANDOMIZED-QUICKSORT(A, p, q − 1)
4      RANDOMIZED-QUICKSORT(A, q + 1, r)
```

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  exchange A[r] with A[i]
3  return PARTITION(A, p, r)
```

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

Recap: Analysis of Randomized Quick Sort

```
RANDOMIZED-QUICKSORT(A, p, r)
1  if p < r
2      q = RANDOMIZED-PARTITION(A, p, r)
3      RANDOMIZED-QUICKSORT(A, p, q − 1)
4      RANDOMIZED-QUICKSORT(A, q + 1, r)
```

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  exchange A[r] with A[i]
3  return PARTITION(A, p, r)
```

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

Let

S_i: i^{th} sorted element

Recap: Analysis of Randomized Quick Sort

```
RANDOMIZED-QUICKSORT(A, p, r)
1  if p < r
2      q = RANDOMIZED-PARTITION(A, p, r)
3      RANDOMIZED-QUICKSORT(A, p, q − 1)
4      RANDOMIZED-QUICKSORT(A, q + 1, r)
```

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  exchange A[r] with A[i]
3  return PARTITION(A, p, r)
```

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

Let

S_i: i^{th} sorted element

X_{ij}: comparisons count S_i to S_j

Recap: Analysis of Randomized Quick Sort

```
RANDOMIZED-QUICKSORT(A, p, r)
1  if p < r
2      q = RANDOMIZED-PARTITION(A, p, r)
3      RANDOMIZED-QUICKSORT(A, p, q − 1)
4      RANDOMIZED-QUICKSORT(A, q + 1, r)
```

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  exchange A[r] with A[i]
3  return PARTITION(A, p, r)
```

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

Let

S_i: i^{th} sorted element

X_{ij}: comparisons count S_i to S_j

Expected comparisons

$$E[\sum_{i=1}^{n}\sum_{j>i} X_{ij}] = \sum_{i=1}^{n}\sum_{j>i} E[X_{ij}]$$

Recap: Analysis of Randomized Quick Sort

```
RANDOMIZED-QUICKSORT(A, p, r)
1  if p < r
2      q = RANDOMIZED-PARTITION(A, p, r)
3      RANDOMIZED-QUICKSORT(A, p, q − 1)
4      RANDOMIZED-QUICKSORT(A, q + 1, r)
```

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  exchange A[r] with A[i]
3  return PARTITION(A, p, r)
```

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

Let
S_i: i^{th} sorted element
X_{ij}: comparisons count S_i to S_j

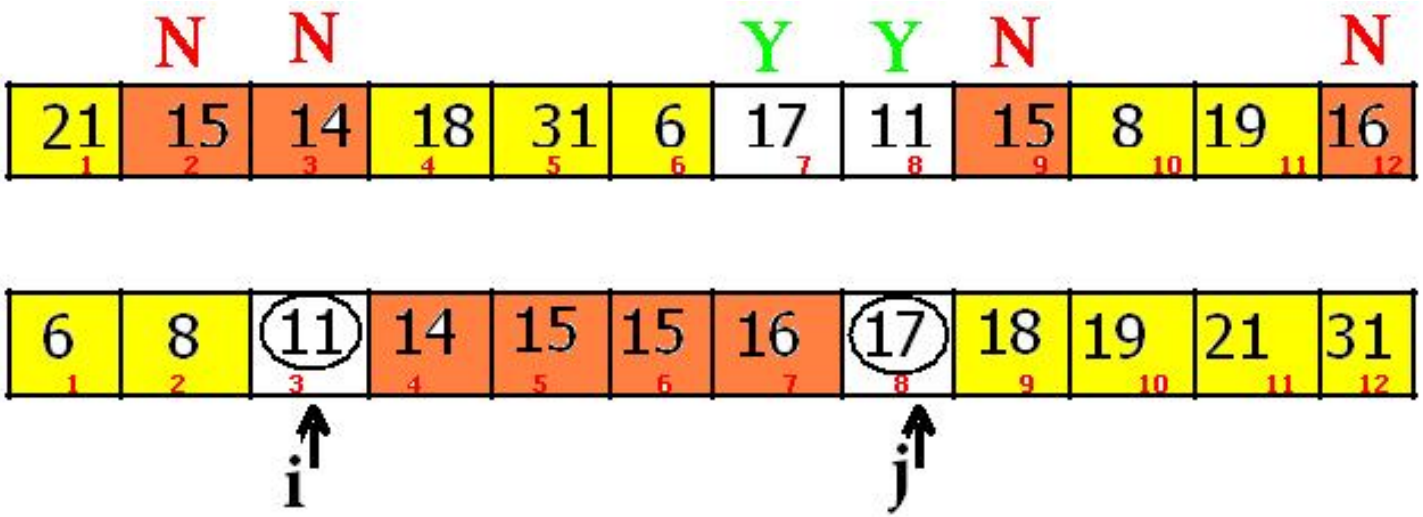
Expected comparisons

$$E[\sum_{i=1}^{n}\sum_{j>i} X_{ij}] = \sum_{i=1}^{n}\sum_{j>i} E[X_{ij}]$$

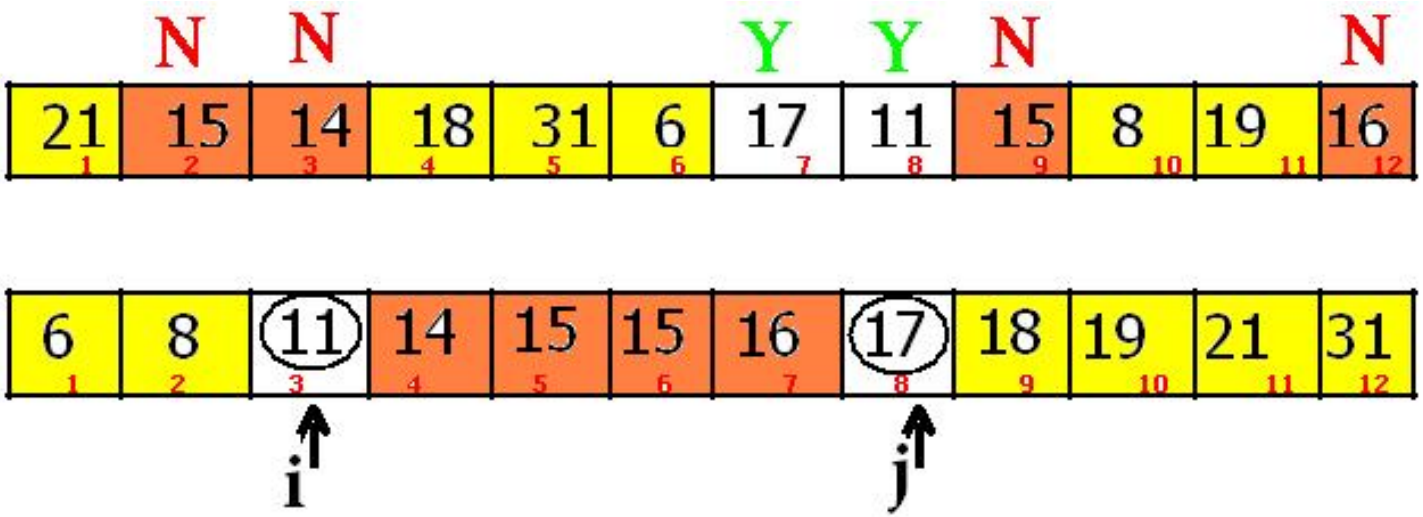
Let p_{ij} be probability of comparison between $S_i \leftrightarrow S_j$

$$E[X_{ij}] = p_{ij} \times 1 + (1 - p_{ij}) \times 0 = p_{ij}$$

Recap: Analysis of Randomized Quick Sort

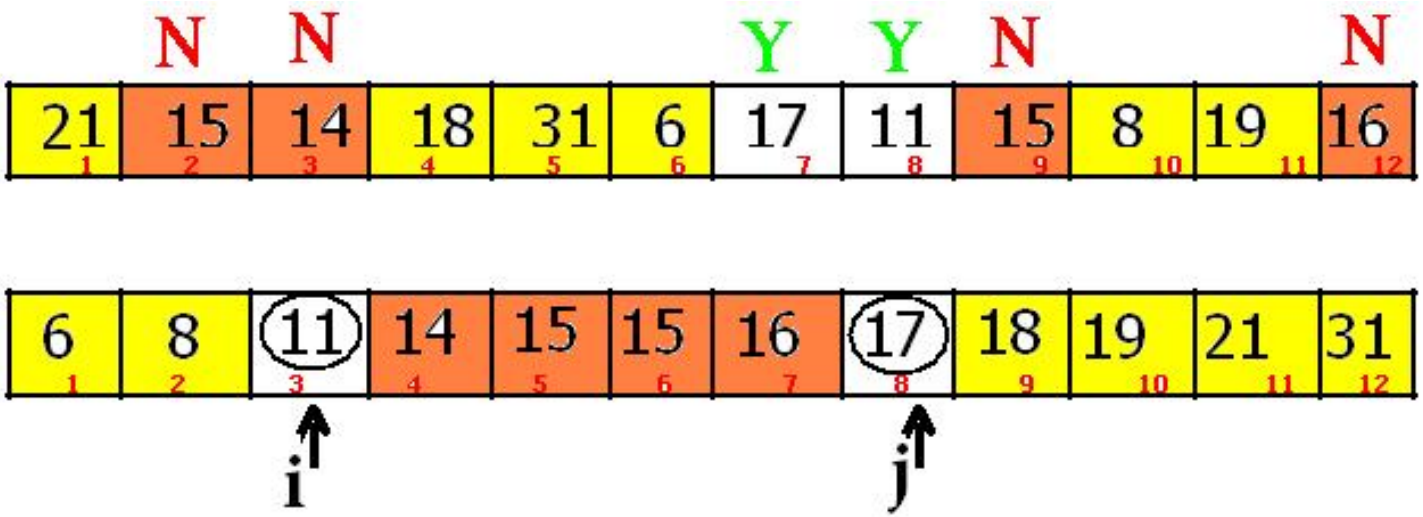


Recap: Analysis of Randomized Quick Sort



$p_{ij} = 2/(j - i + 1)$

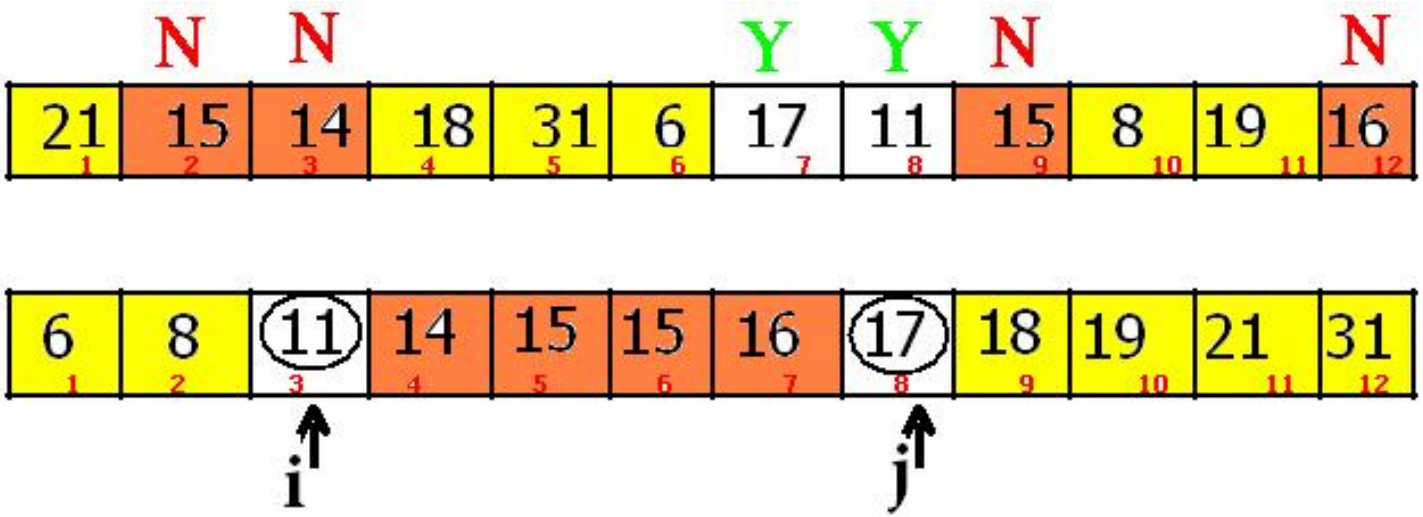
Recap: Analysis of Randomized Quick Sort



$p_{ij} = 2/(j - i + 1)$

$$\sum_{i=1}^{n} \sum_{j>i} E[X_{ij}] = \sum_{i=1}^{n} \sum_{j>i} p_{ij}$$

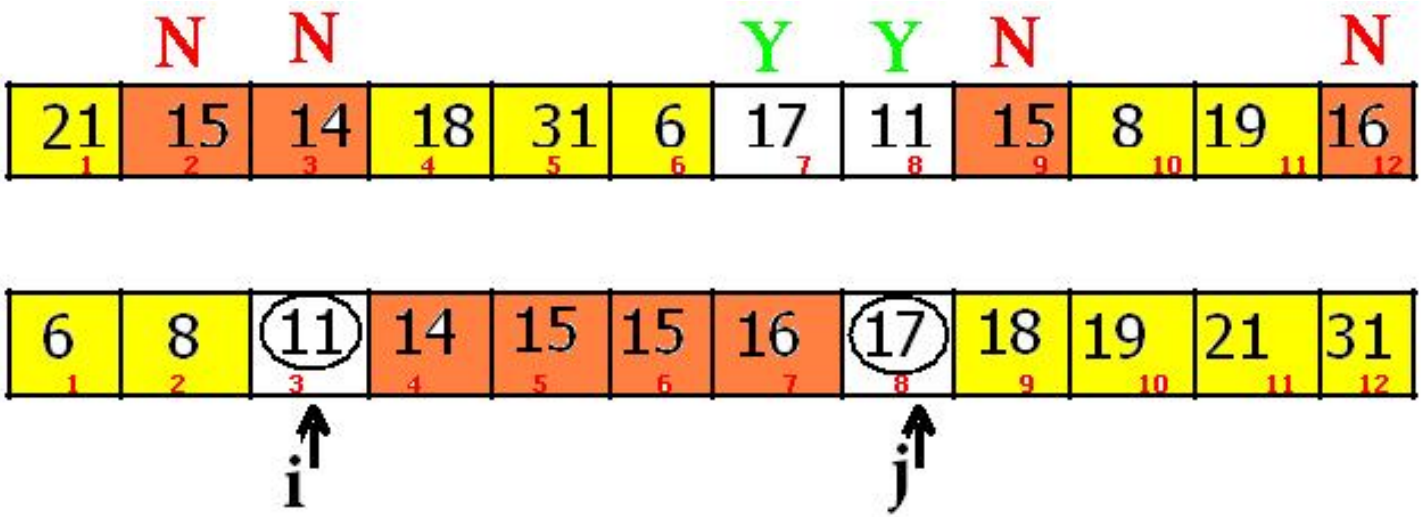
Recap: Analysis of Randomized Quick Sort



$p_{ij} = 2/(j - i + 1)$

$$\sum_{i=1}^{n} \sum_{j>i} E[X_{ij}] = \sum_{i=1}^{n} \sum_{j>i} p_{ij} = \sum_{i=1}^{n} \sum_{j>i} \frac{2}{j - i + 1}$$

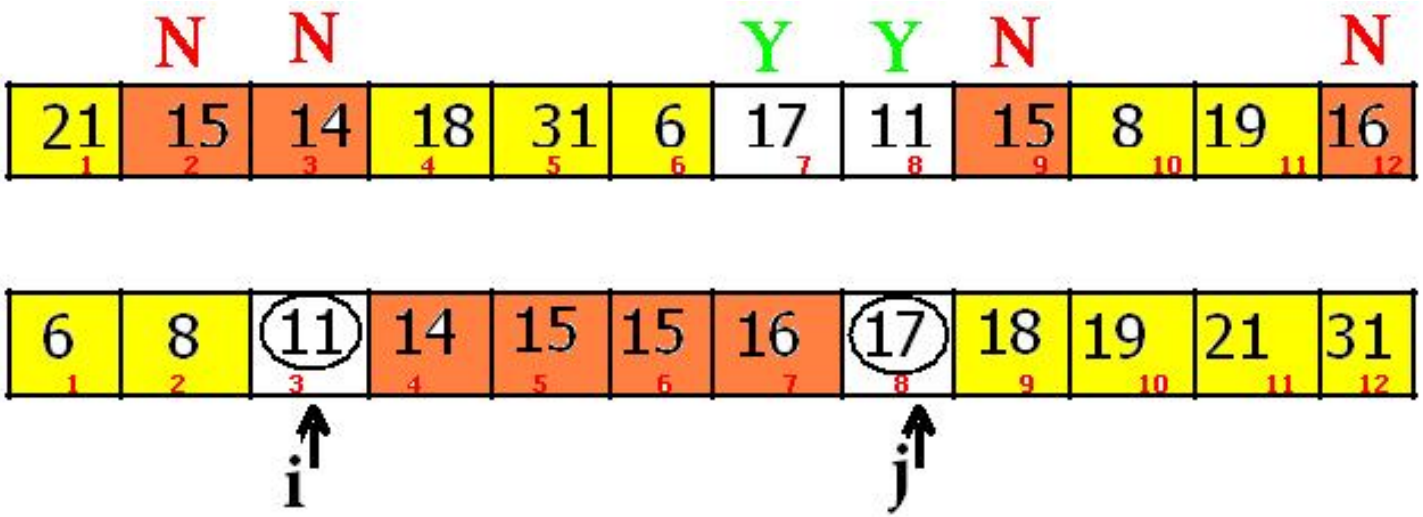
Recap: Analysis of Randomized Quick Sort



$p_{ij} = 2/(j - i + 1)$

$$\sum_{i=1}^{n} \sum_{j>i} E[X_{ij}] = \sum_{i=1}^{n} \sum_{j>i} p_{ij} = \sum_{i=1}^{n} \sum_{j>i} \frac{2}{j - i + 1} = 2 \sum_{i=1}^{n} \sum_{k=2}^{n-i+1} \frac{1}{k}$$

Recap: Analysis of Randomized Quick Sort

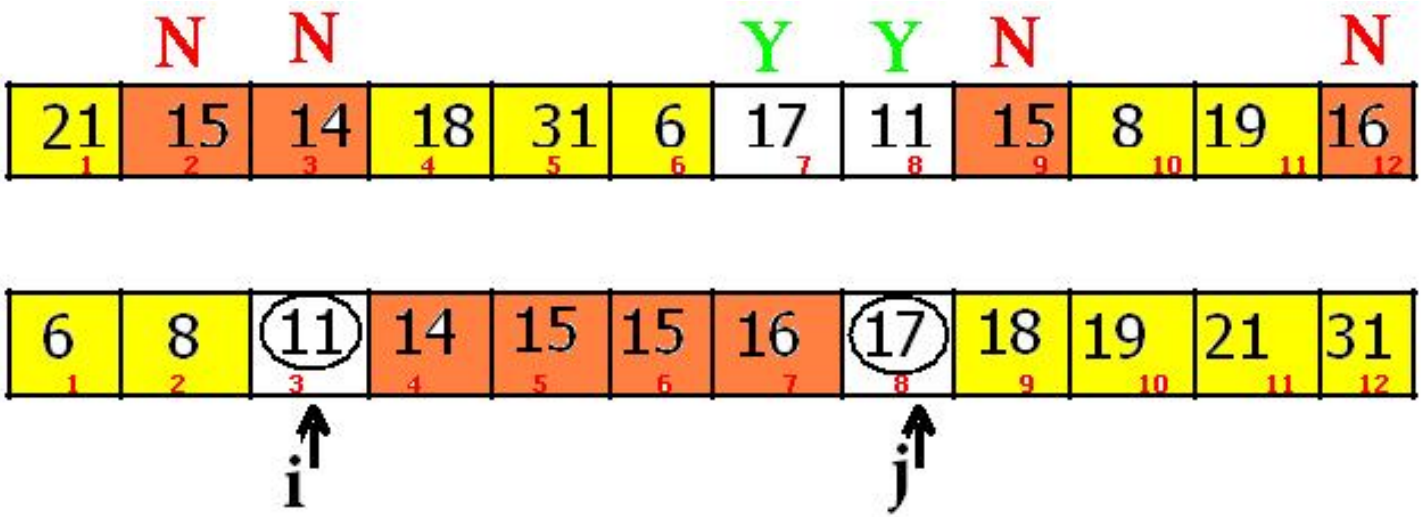


$p_{ij} = 2/(j - i + 1)$

$$\sum_{i=1}^{n} \sum_{j>i} E[X_{ij}] = \sum_{i=1}^{n} \sum_{j>i} p_{ij} = \sum_{i=1}^{n} \sum_{j>i} \frac{2}{j - i + 1} = 2 \sum_{i=1}^{n} \sum_{k=2}^{n-i+1} \frac{1}{k}$$

$$\leq 2 \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k}$$

Recap: Analysis of Randomized Quick Sort



$p_{ij} = 2/(j - i + 1)$

$$\sum_{i=1}^{n} \sum_{j>i} E[X_{ij}] = \sum_{i=1}^{n} \sum_{j>i} p_{ij} = \sum_{i=1}^{n} \sum_{j>i} \frac{2}{j - i + 1} = 2 \sum_{i=1}^{n} \sum_{k=2}^{n-i+1} \frac{1}{k}$$

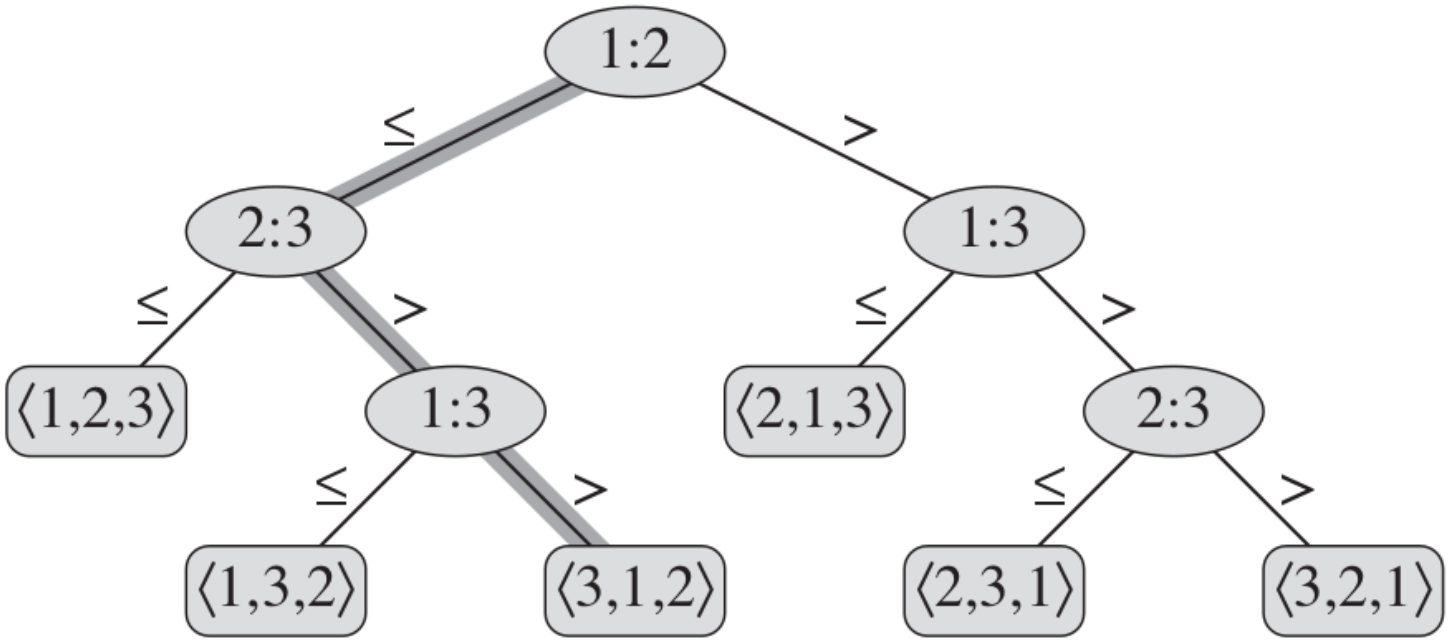
$$\leq 2 \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k} = 2nH_n = O(n \ln n)$$

Decision Tree Model

Sort three items a_1, a_2, a_3

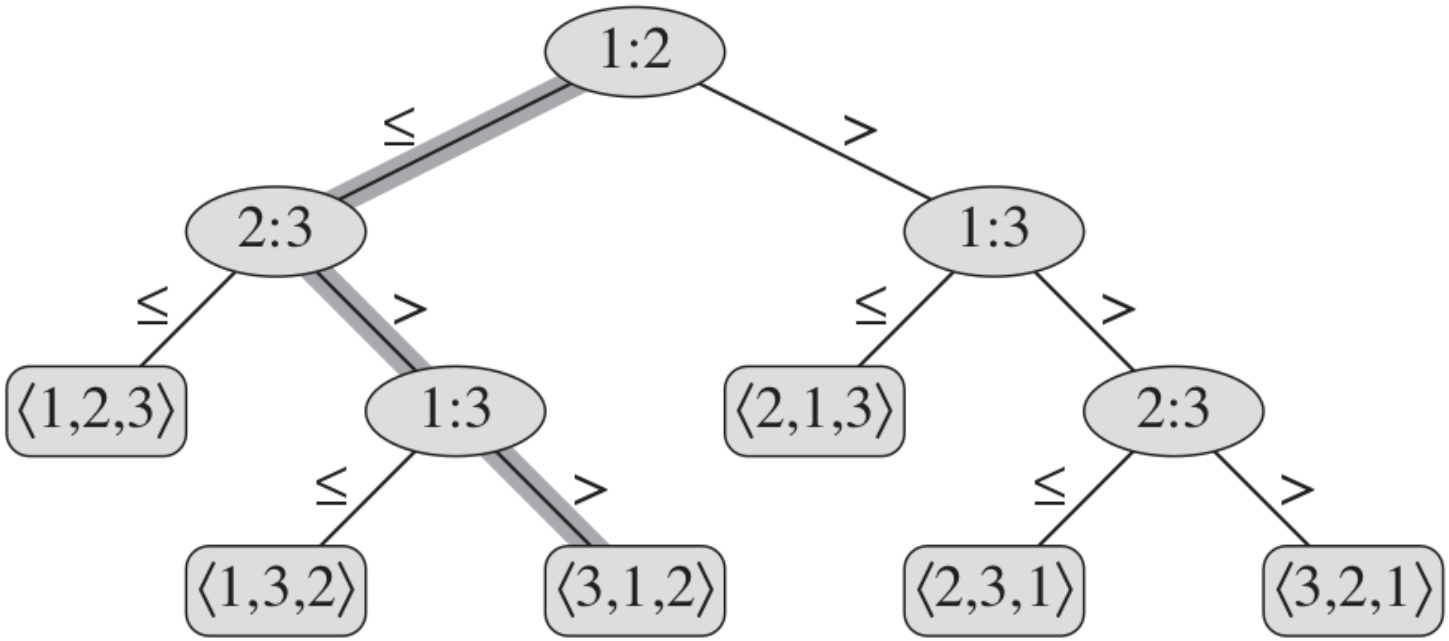
Decision Tree Model

Sort three items a_1, a_2, a_3



Decision Tree Model

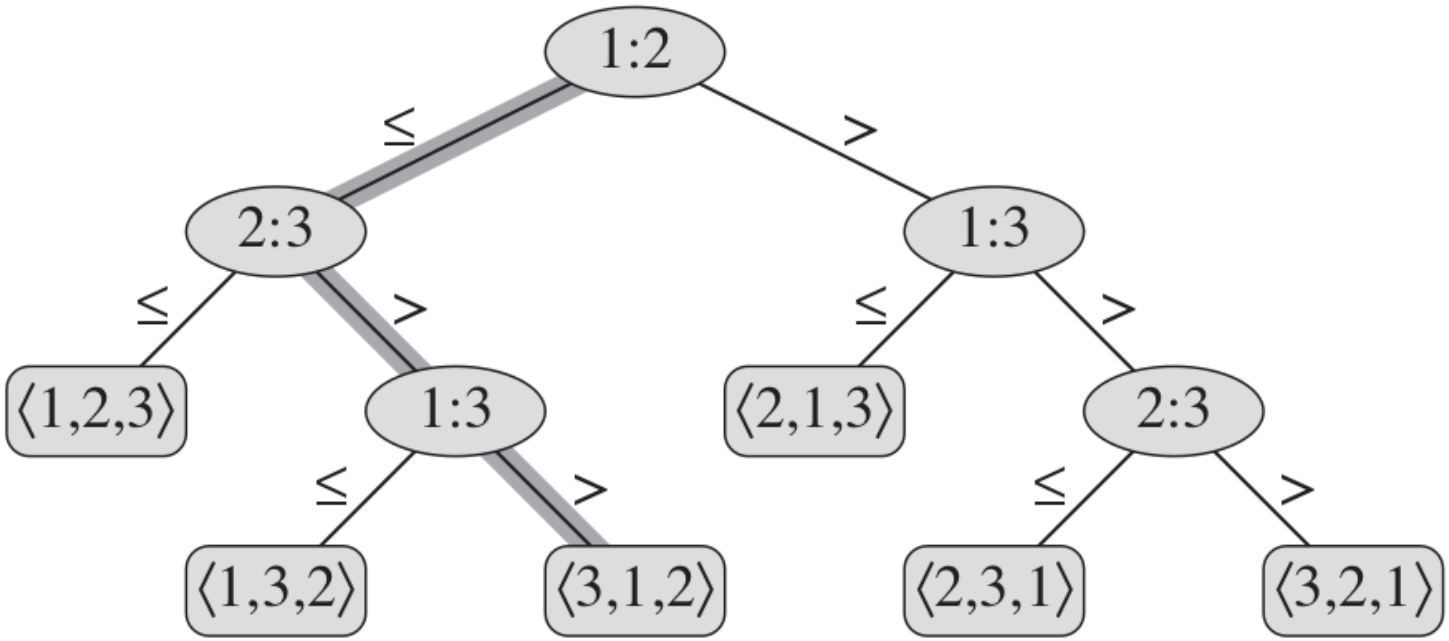
Sort three items a_1, a_2, a_3



Is it always to be a binary tree?

Decision Tree Model

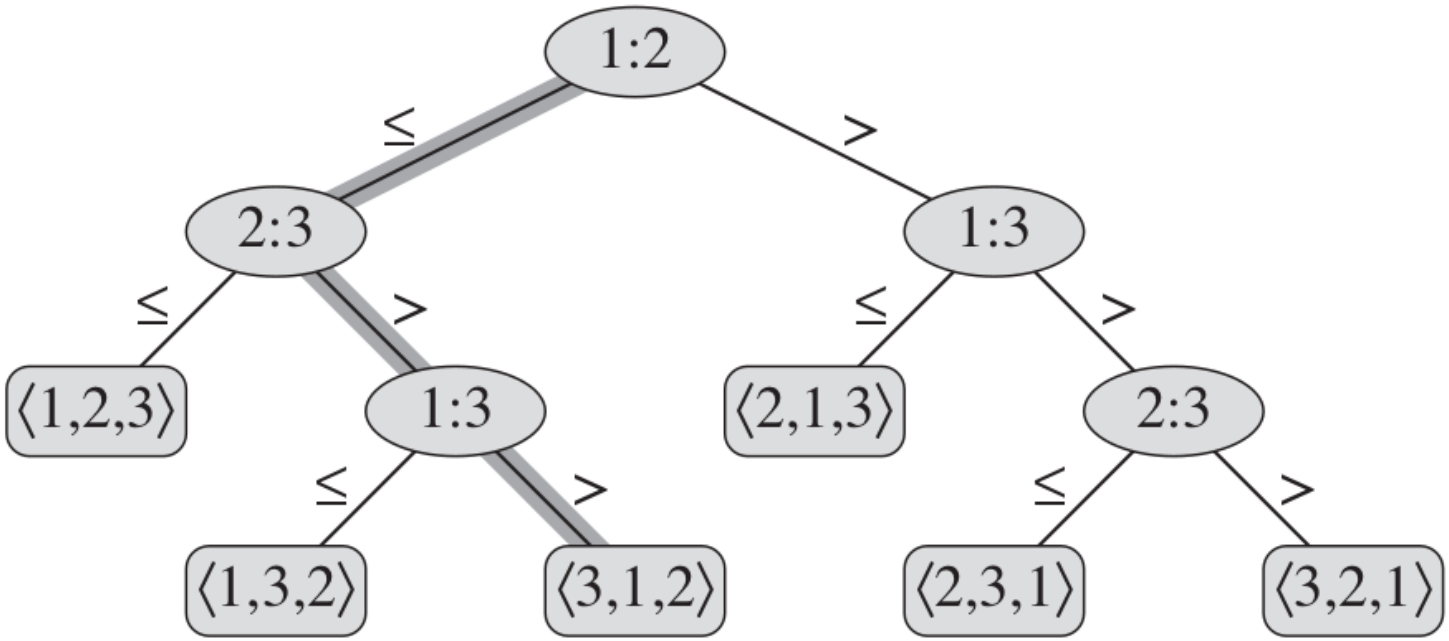
Sort three items a_1, a_2, a_3



Is it always to be a binary tree?
What is worst case time taken by this algorithm?

Decision Tree Model

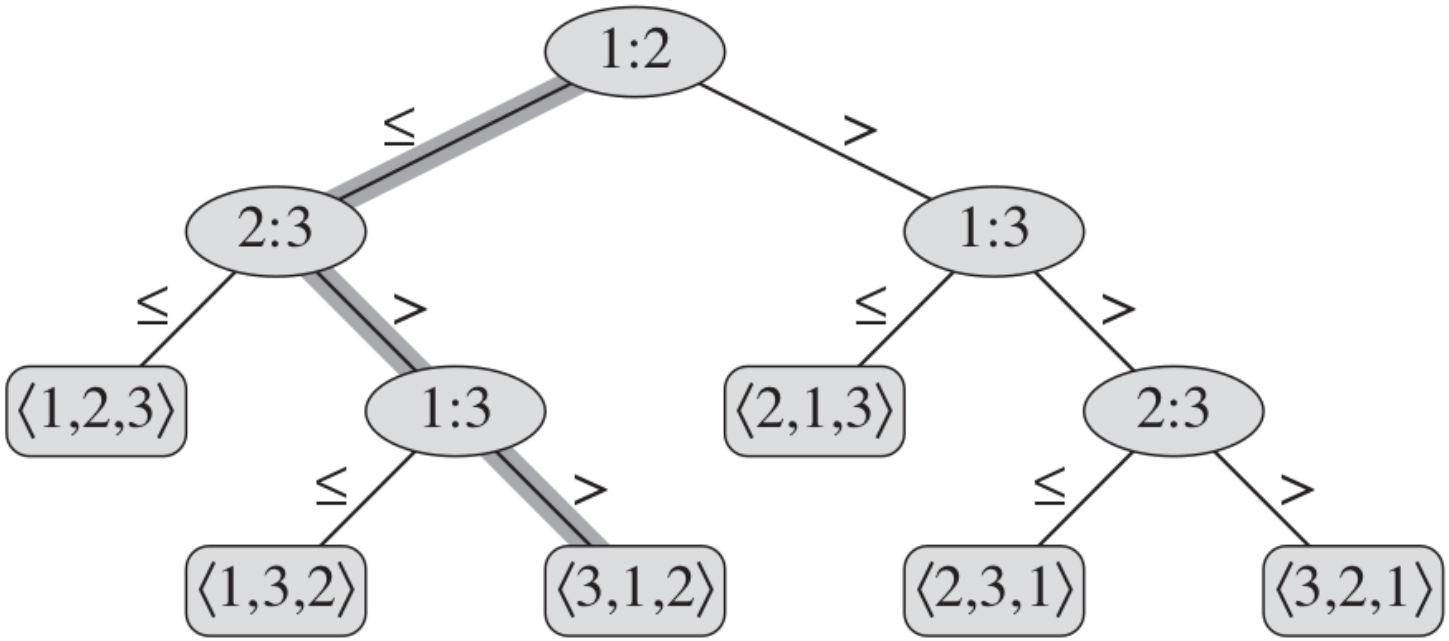
Sort three items a_1, a_2, a_3



Is it always to be a binary tree?
What is worst case time taken by this algorithm? $O(height)$

Decision Tree Model

Sort three items a_1, a_2, a_3



Is it always to be a binary tree?
What is worst case time taken by this algorithm? $O(height)$
How many leaves would be there with 4 items?

Lower Bound

Any comparison sort algorithm requires $\Omega(n \log n)$ comparison in the worst case.

Lower Bound

Any comparison sort algorithm requires $\Omega(n \log n)$ comparison in the worst case.

- There are $n!$ permutations of n items. Each should be a leaf

Lower Bound

Any comparison sort algorithm requires $\Omega(n \log n)$ comparison in the worst case.

- There are $n!$ permutations of n items. Each should be a leaf
- Binary tree of height h has at most 2^h leaves

Lower Bound

Any comparison sort algorithm requires $\Omega(n \log n)$ comparison in the worst case.

- There are $n!$ permutations of n items. Each should be a leaf
- Binary tree of height h has at most 2^h leaves

$$n! \leq 2^h$$

Lower Bound

Any comparison sort algorithm requires $\Omega(n \log n)$ comparison in the worst case.

- There are $n!$ permutations of n items. Each should be a leaf
- Binary tree of height h has at most 2^h leaves

$$n! \leq 2^h$$

$$h \geq \log(n!)$$

Lower Bound

Any comparison sort algorithm requires $\Omega(n \log n)$ comparison in the worst case.

- There are $n!$ permutations of n items. Each should be a leaf
- Binary tree of height h has at most 2^h leaves

$$n! \leq 2^h$$

$$h \geq \log(n!)$$

- Stirling's approximation of $n!$ is $(n/e)^n$

Lower Bound

Any comparison sort algorithm requires $\Omega(n \log n)$ comparison in the worst case.

- There are $n!$ permutations of n items. Each should be a leaf
- Binary tree of height h has at most 2^h leaves

$$n! \leq 2^h$$

$$h \geq \log(n!)$$

- Stirling's approximation of $n!$ is $(n/e)^n$

$$h \geq n\log(n) - n\log(e)$$

Lower Bound

Any comparison sort algorithm requires $\Omega(n \log n)$ comparison in the worst case.

- There are $n!$ permutations of n items. Each should be a leaf
- Binary tree of height h has at most 2^h leaves

$$n! \leq 2^h$$

$$h \geq \log(n!)$$

- Stirling's approximation of $n!$ is $(n/e)^n$

$$h \geq n\log(n) - n\log(e)$$

$$h = \Omega(n\log(n))$$

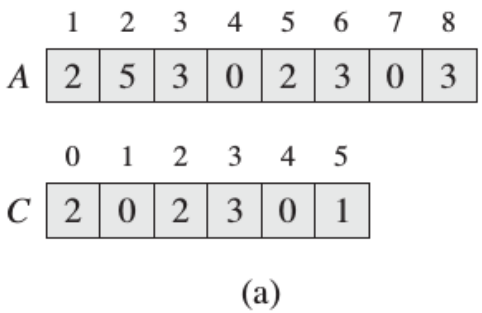
Still Wish $O(n)$? use Counting Sort

COUNTING-SORT(A, B, k)

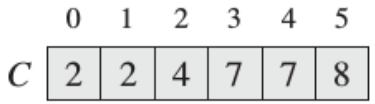
```
1   let C[0..k] be a new array
2   for i = 0 to k
3       C[i] = 0
4   for j = 1 to A.length
5       C[A[j]] = C[A[j]] + 1
6   // C[i] now contains the number of elements equal to i.
7   for i = 1 to k
8       C[i] = C[i] + C[i − 1]
9   // C[i] now contains the number of elements less than or equal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

Sort: 2, 5, 3, 0, 2, 3, 0, 3

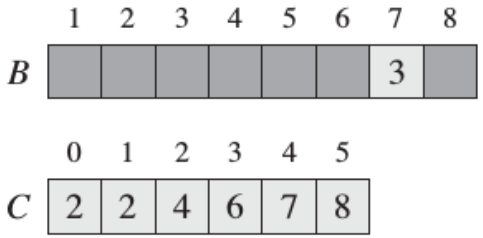
Counting Sort in action



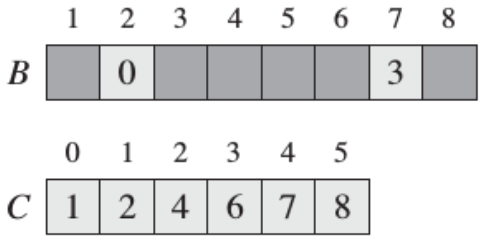
(a)



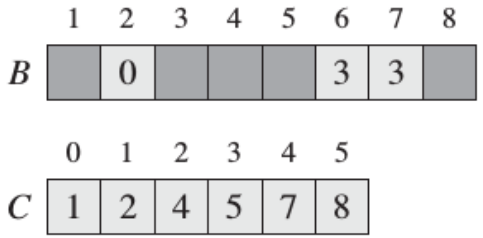
(b)



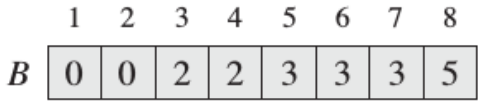
(c)



(d)



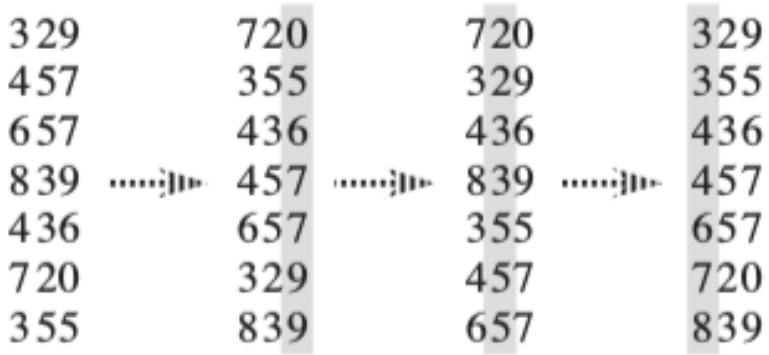
(e)



(f)

Radix Sort

Use a stable sorting algorithm to sort array A on digit (1 to d)

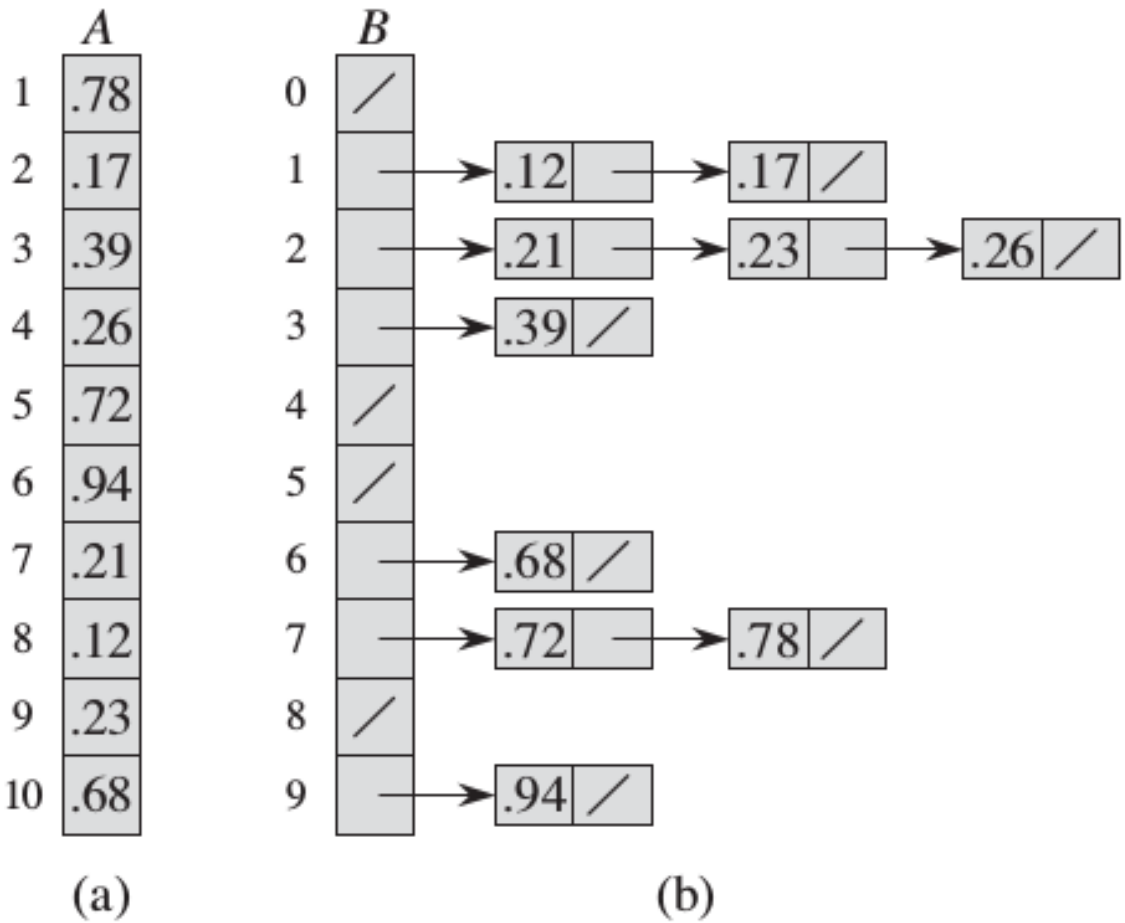


Bucket Sort

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


A
1 .78
2 .17
3 .39
4 .26
5 .72
6 .94
7 .21
8 .12
9 .23
10 .68
(a)
B
0
1 .12 .17
2 .21 .23 .26
3 .39
4
5
6 .68
7 .72 .78
8
9 .94
(b)

Thank You!

Thank you very much for your attention!

Queries ?