Sorting Algorithms

Sorting

- Given a sequence of n numbers $\langle a_1,\ldots,a_n \rangle$, reorder it into $\langle b_1,\ldots,b_n \rangle$ where $b_1 \leq \ldots \leq b_n$.
- Input: $\langle a_1,\ldots,a_n \rangle$
- Output: $\langle b_1,\ldots,b_n \rangle$
- Sample Input: $\langle 0,3,5,7,1,2,1,2,1 \rangle$
- Sample Output: $\langle 0,1,1,1,2,2,3,5,7 \rangle$

Sorting

Algorithm	Worst-case	Average-case
Insertion Sort	$\Theta(n^2)$	$\Theta(n^2)$
Merge Sort	$\Theta(n\log n)$	$\Theta(n\log n)$
Heap Sort	$O(n\log n)$	——
Quick Sort	$\Theta(n^2)$	$\Theta(n\log n)$ (Expected)
Counting Sort	$\Theta(n+k)$	$\Theta(n+k)$
Radix Sort	$\Theta(d(n+k))$	$\Theta(d(n+k))$
Bucket Sort	$\Theta(n^2)$	$\Theta(n)$ (Expected)

Non-comparison sort

Comparison sort

Quick Sort

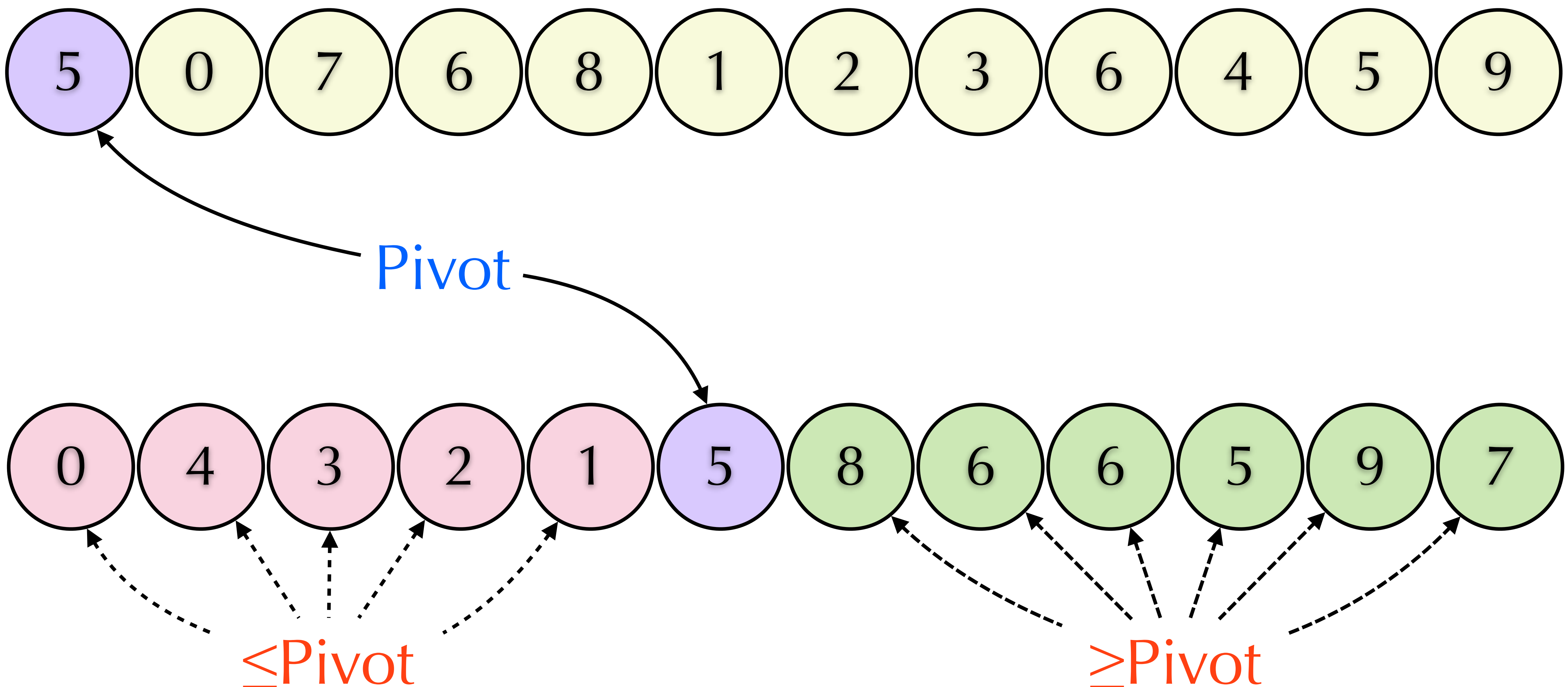
Sort A[1],...,A[n]

- Termination: It is sorted when $n=1$.
- Divide: Reorder A and find m such that
  - For $i<m$, $A[i] \le A[m]$.
  - For $i>m$, $A[i] \ge A[m]$.

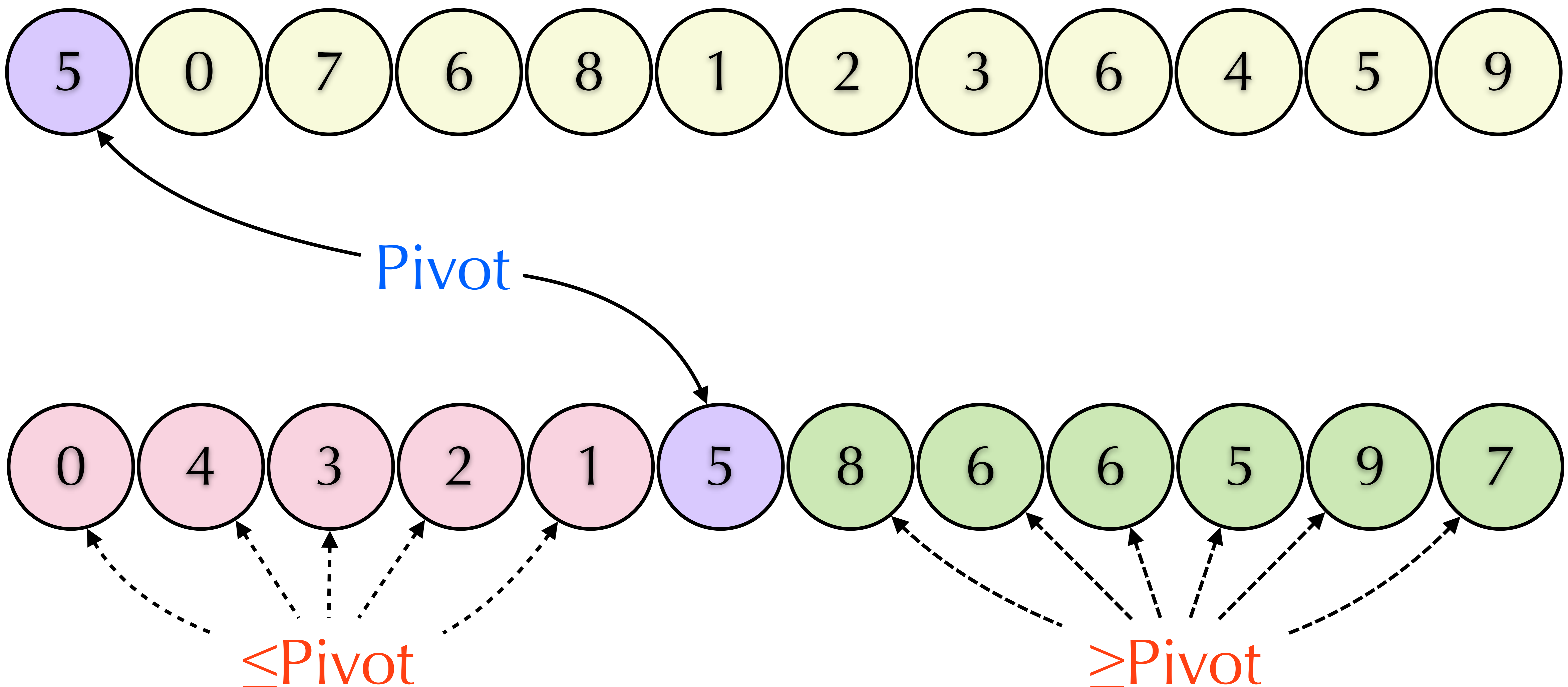
Partition

- Conquer: Sort A[1..m−1] and A[m+1..n].
- Combine: No need.
- Time: $T(n)=T(m-1)+T(n-m)+\Theta(n)$

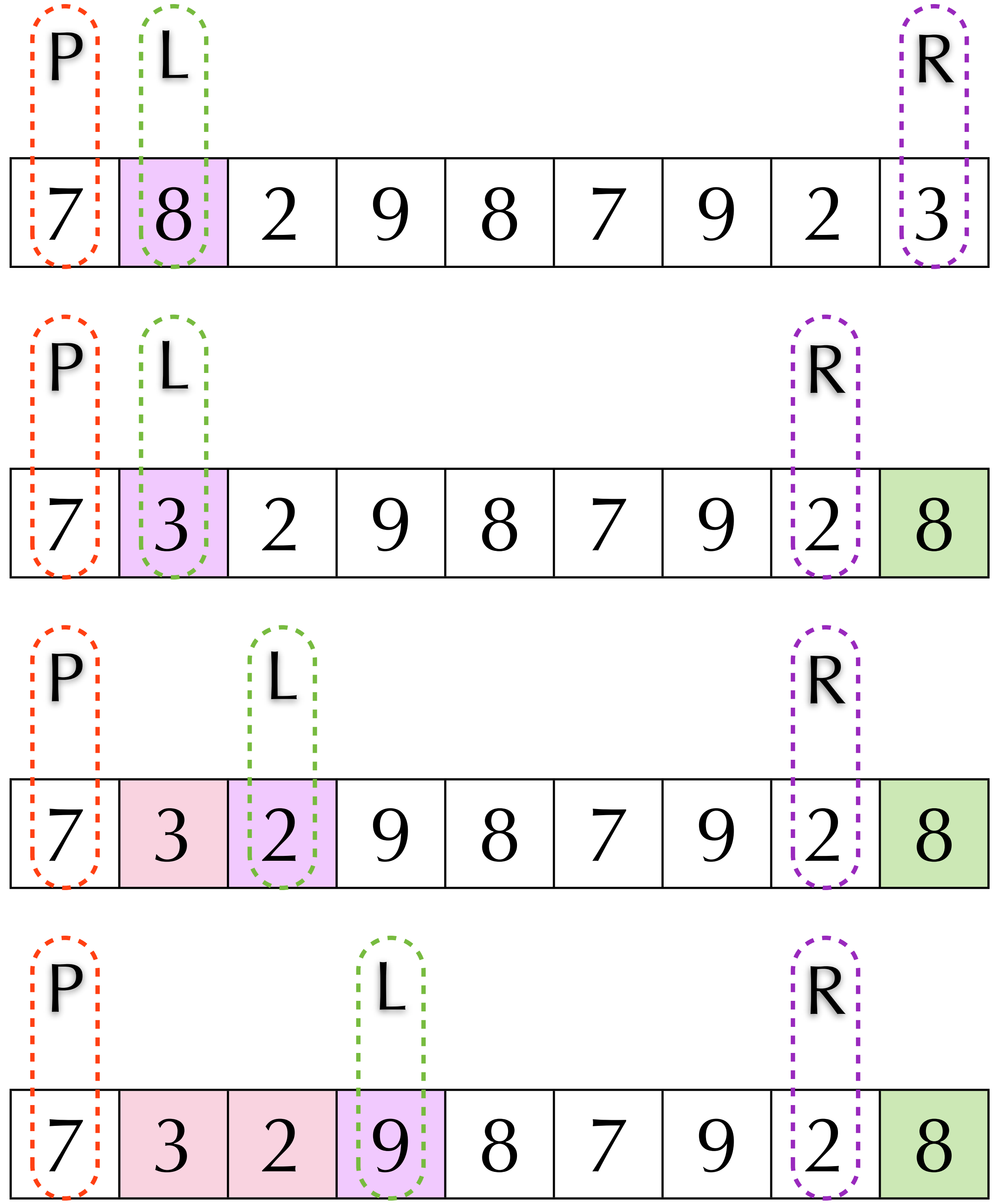
Partition



Partition



Partition

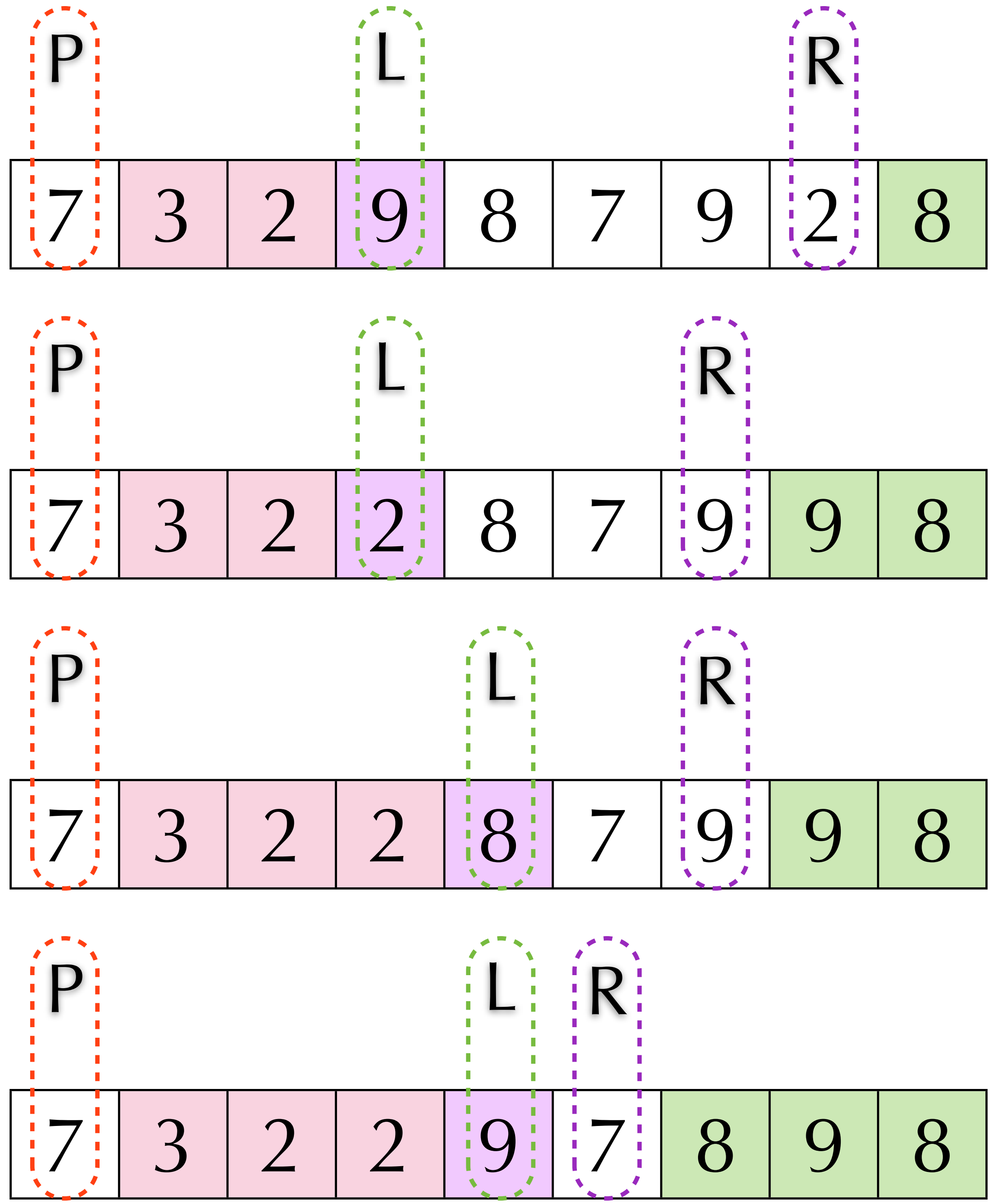


If A[P]<A[L]:
swap(A[L],A[R])
R=R–1

If A[P]≥A[L]:
L=L+1

If A[P]≥A[L]:
L=L+1

Partition

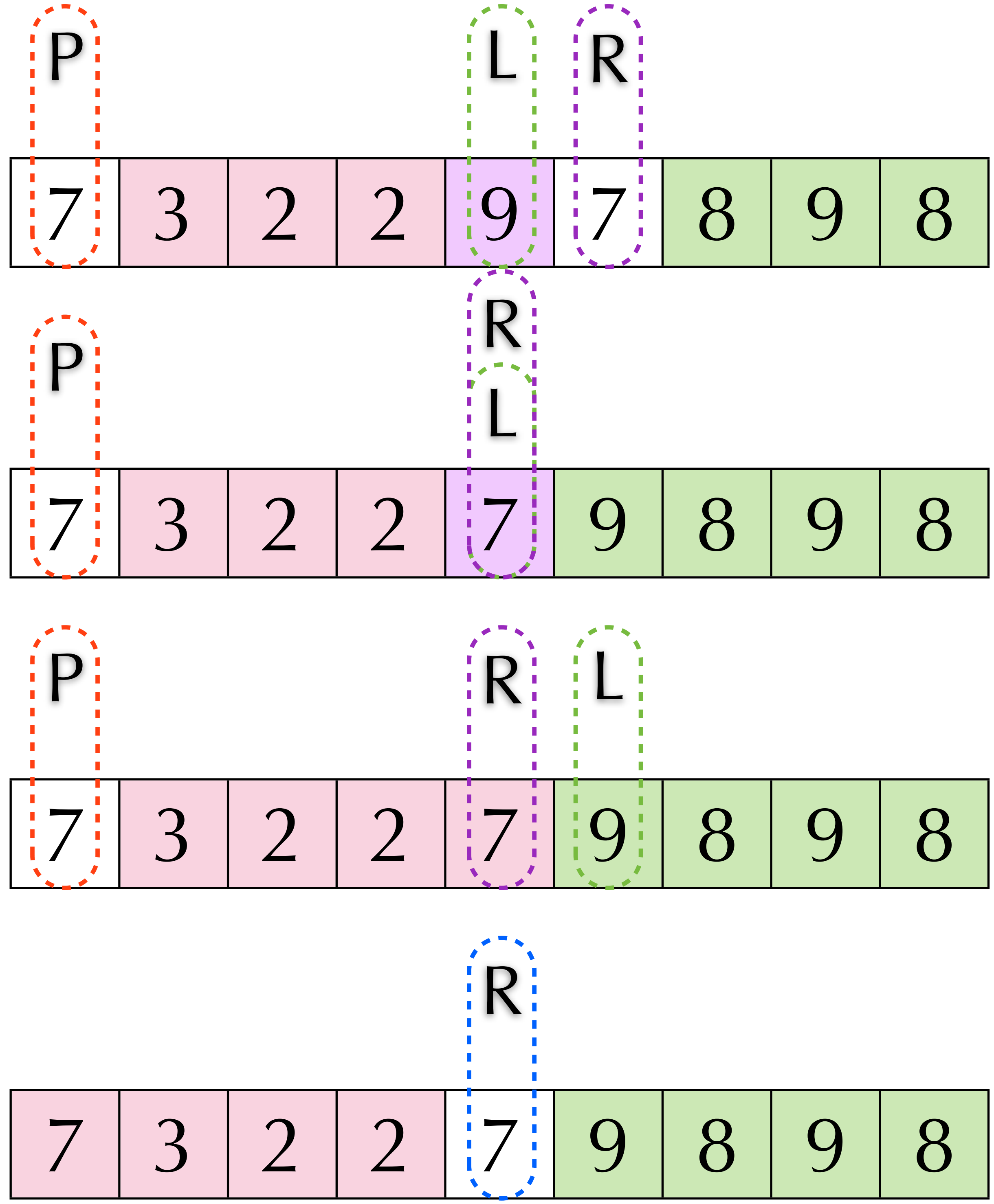


If A[P]<A[L]:
swap(A[L],A[R])
R=R–1

If A[P]≥A[L]:
L=L+1

If A[P]<A[L]:
swap(A[L],A[R])
R=R–1

Partition

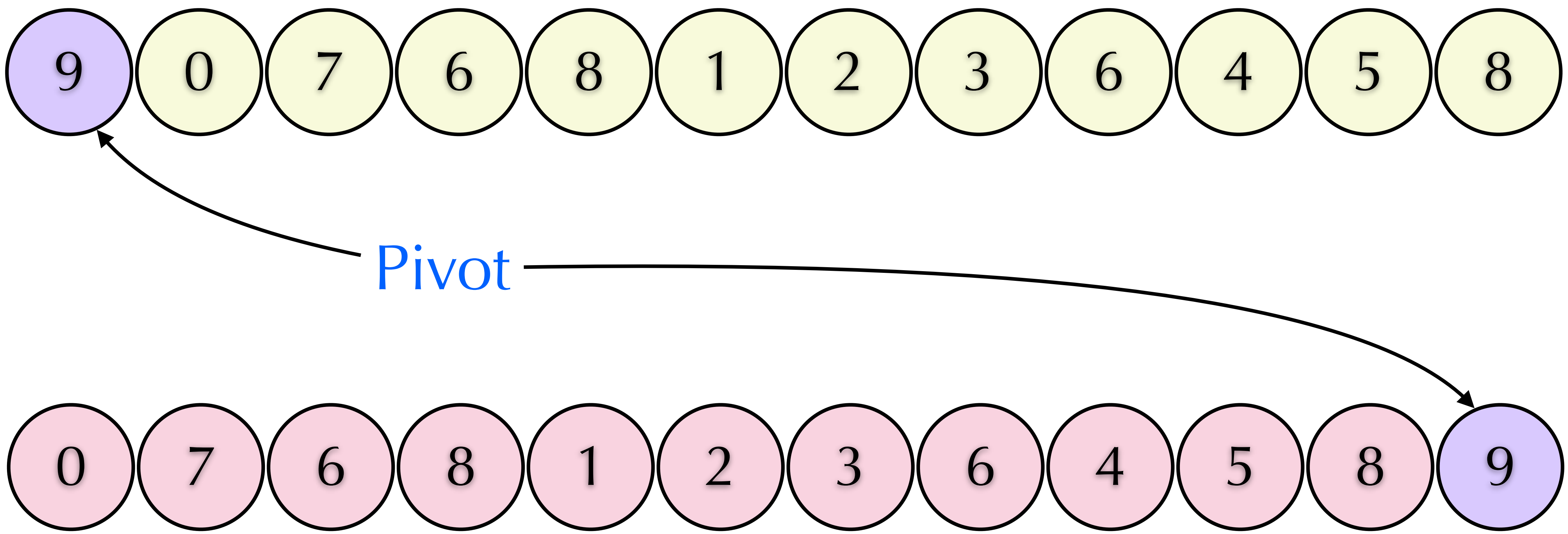


If A[P]<A[L]:
swap(A[L],A[R])
R=R–1

If A[P]≥A[L]:
L=L+1

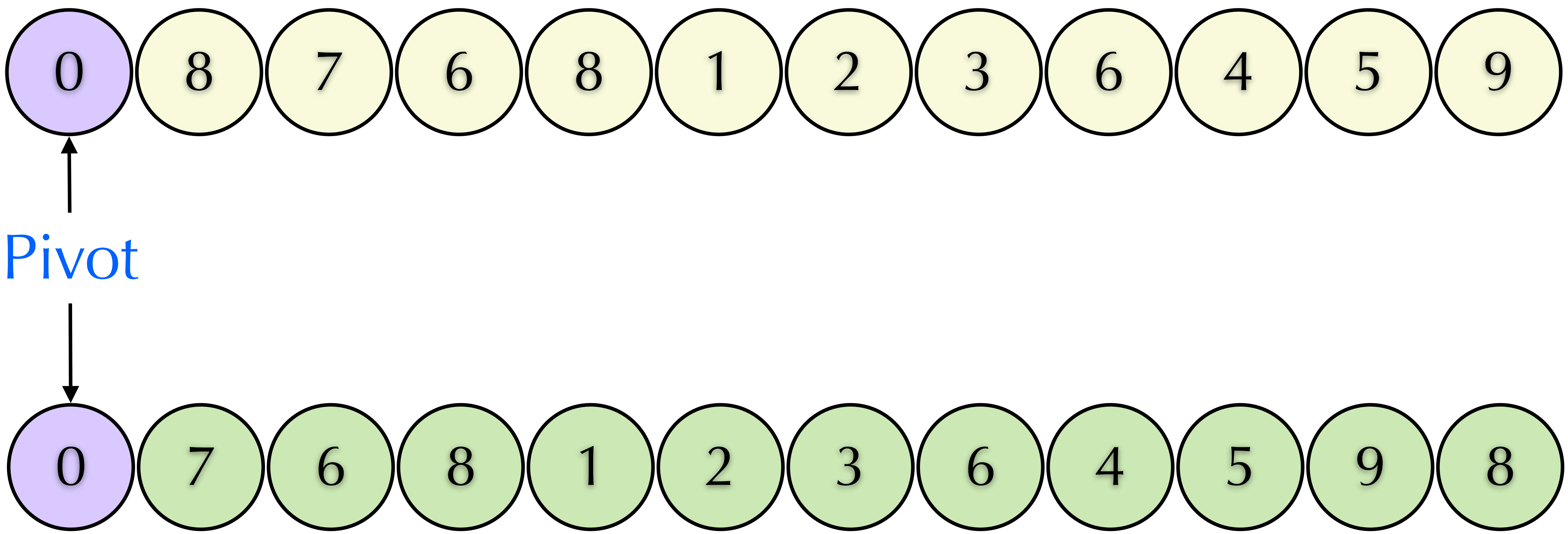
If R<L:
swap(A[P],A[R])
return R

Partition: Bad Case 1



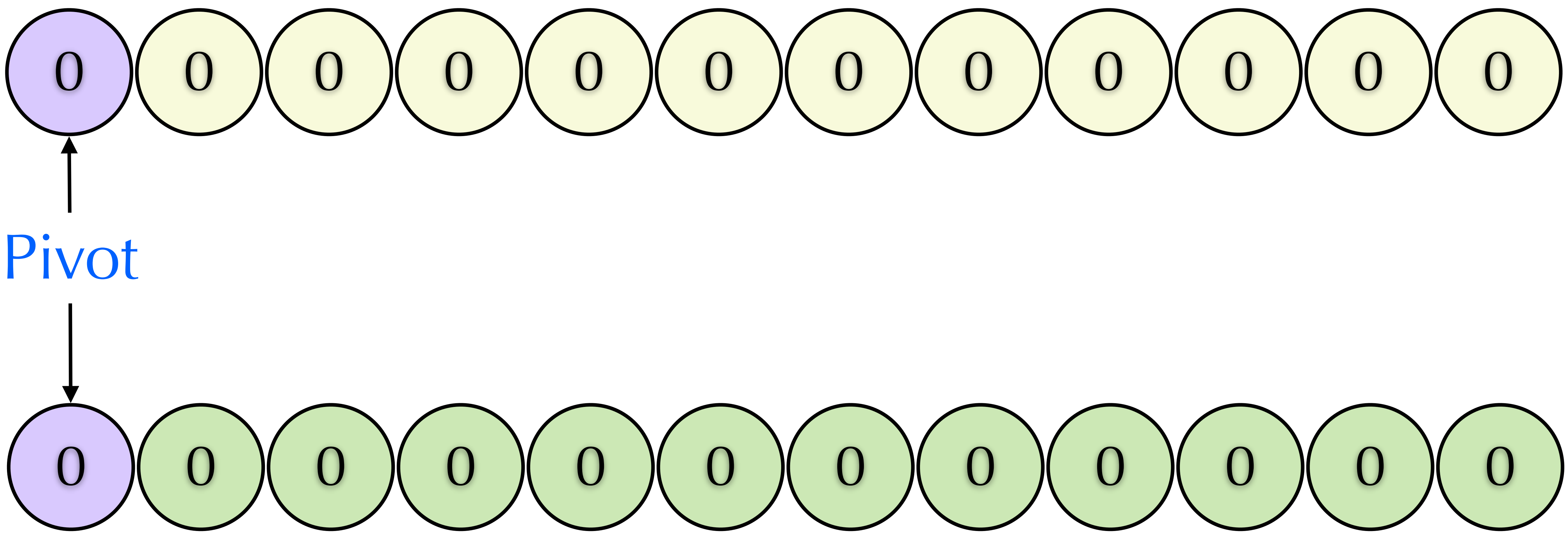
Actually, this is the worst case. (Note: proof is needed.)

Partition: Bad Case 2



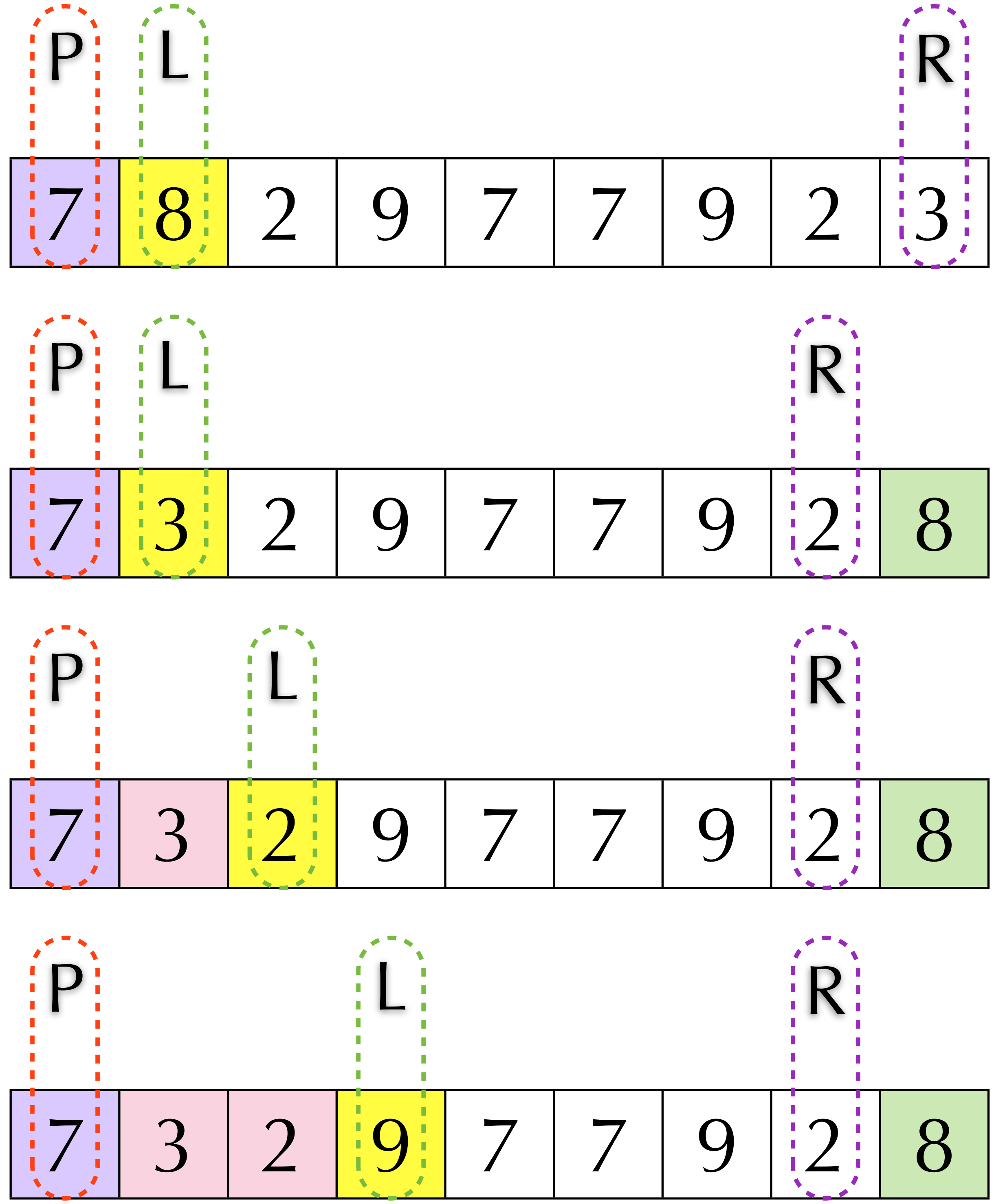
Actually, this is the worst, too. (Note: proof is needed.)

Partition: Bad Case 3



Actually, this is the worst, too. (Note: proof is needed.)

Modified Partition

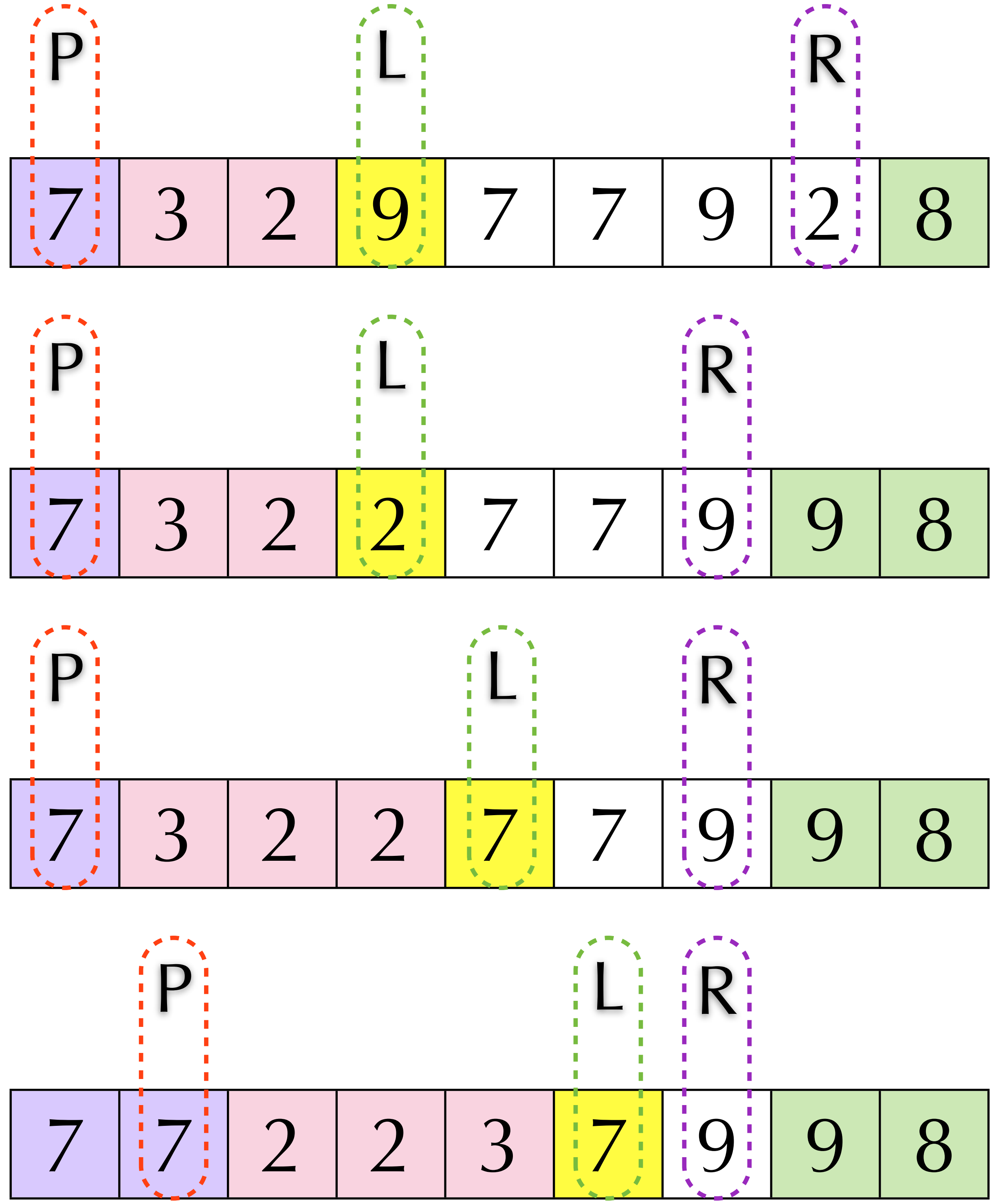


If A[P]<A[L]:
swap(A[L],A[R])
R=R–1

If A[P]>A[L]:
L=L+1

If A[P]>A[L]:
L=L+1

Modified Partition

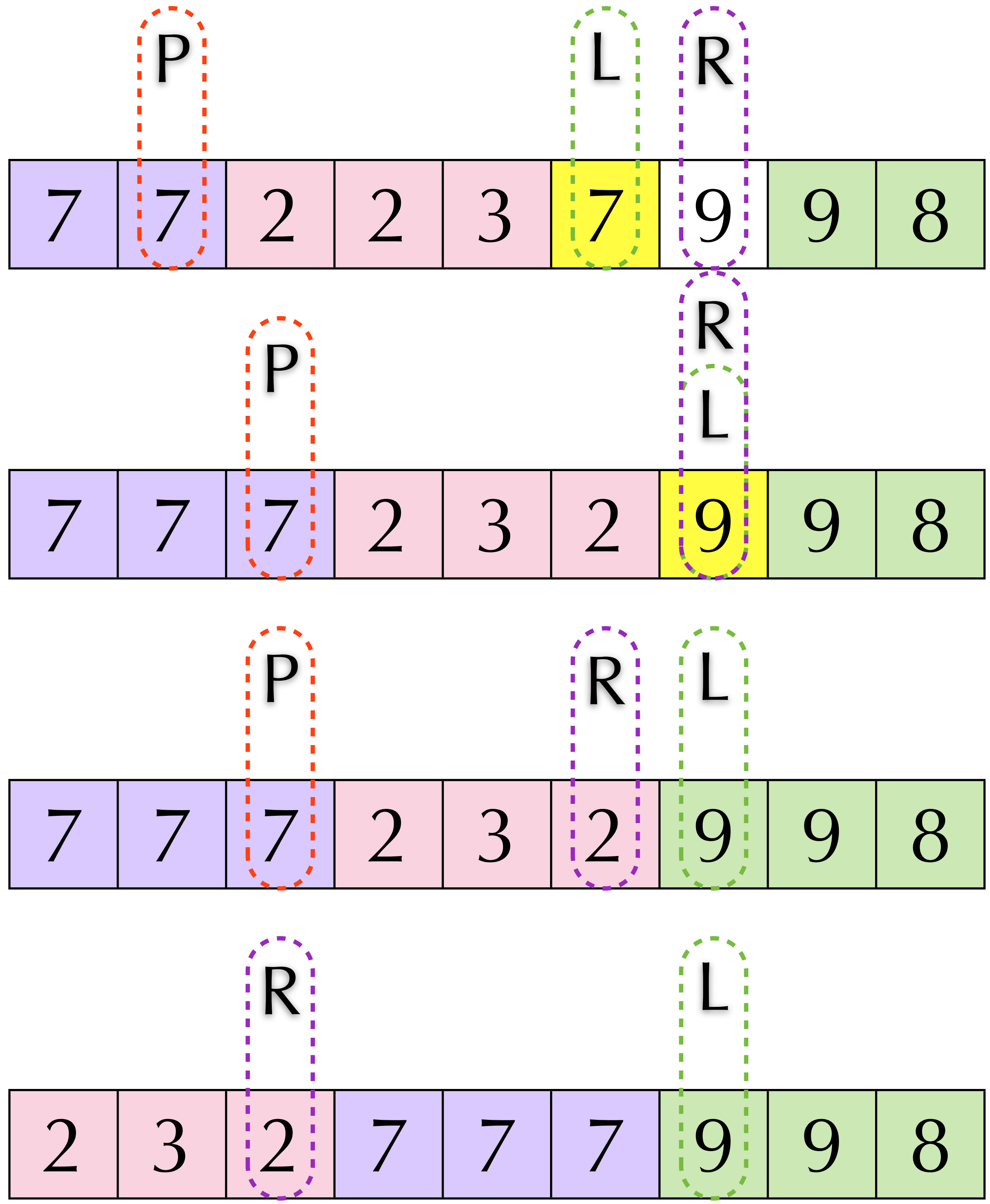


If A[P]<A[L]:
swap(A[L],A[R])
R=R–1

If A[P]>A[L]:
L=L+1

If A[P]=A[L]:
P=P+1
swap(A[P],A[L])
L=L+1

Modified Partition



If A[P]=A[L]:
P=P+1
swap(A[P],A[L])
L=L+1

If A[P]<A[L]:
swap(A[L],A[R])
R=R–1

If R<L:
while(P>0)
  swap(A[P],A[R])
  P=P–1, R=R–1

qsort(A[1..R])
qsort(A[L..n])

Quick Sort

- Worst case:
  - $T(n)=\max_{1\le m\le n}(T(m-1)+T(n-m))+\Theta(n)$
  - $T(n)=?$
- Average case:
  - What is average? The input sequence is uniformly randomly sampled.
  - $T(n)=(2/n)(T(1)+\ldots+T(n-1))+\Theta(n)$
  - $T(n)=?$

Worst-Case

- $T(n) \geq T(n-1)+\Theta(n) \geq T(n-2)+\Theta(n-1+n)$
  $\geq T(1)+\Theta(2+\ldots+n)=\Theta(n^2)$ .... $T(n)=\Omega(n^2)$
- Guess $T(n)=O(n^2)$: Assume $T(n) \leq cn^2$
- $T(n) \leq c(\max_{1 \leq m \leq n}((m-1)^2+(n-m)^2))+\Theta(n)$
  $\leq c(\max_{1 \leq m \leq n}(2m^2-2m-2nm+n^2+1))+c'n$
  $=c(n-1)^2+c'n$
  $=c(n^2-2n+1)+c'n$
  $=cn^2-(2cn-c'n-c) \leq cn^2$ ... take $c \geq c'$, $n \geq 1$
- $T(n)=O(n^2)$. Conclusion: $T(n)=\Theta(n^2)$

Average Case

- It is sufficient to show that: $F(n)=(2/n)(F(1)+\ldots+F(n-1))+n=\Theta(n\log n)$
- $nF(n)=2F(1)+2F(2)+\ldots+2F(n-1)+n^2$
- $(n-1)F(n-1)=2F(1)+\ldots+2F(n-2)+n^2-2n+1$
- $nF(n)=(n+1)F(n-1)+2n-1$
- $$\begin{aligned} F(n) &= \tfrac{n+1}{n}F(n-1)+\Theta(1) \\ &= \tfrac{n+1}{n}\left(\tfrac{n}{n-1}F(n-2)+\Theta(1)\right)+\Theta(1) \\ &= \tfrac{n+1}{n-1}F(n-2)+\left(\tfrac{n+1}{n}+1\right)\Theta(1) \\ &= \tfrac{n+1}{n-3}F(n-3)+\left(\tfrac{n+1}{n-1}+\tfrac{n+1}{n}+1\right)\Theta(1) \\ &= (n+1)F(1)+\Theta\left(\sum_{k=1}^{n+1}\tfrac{n+1}{k}\right) \end{aligned}$$

$\Theta(n\log n)$

Note on Quick Sort

- This analysis works only for distinct keys
- The partition method and the complexity analysis in the textbook are different from the slides. At least not identical.
- Please read the textbook.

Comparison Sort: Lower Bound

- The output $\langle b_1,\ldots,b_n \rangle$ of any sorting algorithm is sorted.
- Sorting algorithm: determine a one-to-one and onto mapping function $f:\{1,\ldots,n\} \to \{1,\ldots,n\}$ such that $a_{f(i)}=b_i$.
- There are $n!$ one-to-one and onto functions (permutations).
- At the beginning, every one is a possible candidate of the answer.

Comparison Sort: Lower Bound

- A comparison has 3 possible results:
  - <, =, and >.
- Suppose we need m comparisons to determine f. Then, we have $n!/3^m \leq 1$.
- $3^m \geq n!$ implies $m = \Omega(n\log n)$.
- $o(n\log n)$-time sort: Non-comparison sort

Counting Sort

- Works only if there are not many kinds of values. (Non-comparison based sorting)
- Suppose there are k kinds of values.
  - values: $v_1 < v_2 < \ldots < v_k$ (sometimes simply $v_i = i$)
  - Prepare k queues $Q_1, \ldots, Q_k$.
  - For each a_j of value v_i: enqueue a_j into Q_i.
  - For $i=1$ to k: Repeat dequeueing Q_i until empty. Append the dequeued value to the output.
- It can be stable!

Counting Sort

A

7	8	2	9	8	7	9	2	3

B

Queue Array

0	1	2	3	4	5	6	7	8	9

Counting Sort

A

7	8	2	9	8	7	9	2	3

B

Queue Array

0	1	2	3	4	5	6	7	8	9
							7		

Counting Sort

A

7	8	2	9	8	7	9	2	3

B

Queue Array

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

Counting Sort

A

7	8	2	9	8	7	9	2	3

B

Queue Array

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

Counting Sort

A

7	8	2	9	8	7	9	2	3

B

Queue Array

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

Counting Sort

A

7	8	2	9	8	7	9	2	3

B

Queue Array

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

Counting Sort

A

7	8	2	9	8	7	9	2	3

B

Queue Array

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

Counting Sort

A

7	8	2	9	8	7	9	2	3

B

Queue Array

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

Counting Sort

A

7	8	2	9	8	7	9	2	3

B

Queue Array

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

Counting Sort

A

7	8	2	9	8	7	9	2	3

B

Queue Array

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

Counting Sort

A

7	8	2	9	8	7	9	2	3

B

2	2	3	7	7	8	8	9	9

Queue Array

0	1	2	3	4	5	6	7	8	9

Time Complexity

- Both enqueue and dequeue take $\Theta(1)$.
- Each of $a_1,\ldots,a_n$ is enqueued once and dequeued once. $\Theta(n)$ in total
- We have to check whether Q_i is empty for c_i+1 times if there are c_i number equal to v_i. $\Theta(n+k)$ in total
- We can conclude it takes $\Theta(n+k)$ time.

Counting Sort

- Queue array takes a lot extra space.
- Let $c_{ount}[i]$ be the numbers of a_j equal to v_i.
- Let $c_{umulated\_count}[i]=\Sigma_{i'\leq i}c_{ount}[i']$.
- For j=n downto 1

  If $a_j=v_i$

  $B[c_{umulated\_count}[i]]=a_j$

  $c_{umulated\_count}[i]=c_{umulated\_count}[i]-1$

Counting Sort

A

7	8	2	9	8	7	9	2	3

Count

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

B

Cumulated_count

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

Counting Sort

A

7	8	2	9	8	7	9	2	3

Count

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

B

		3						

Cumulated_count

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

Counting Sort

A

7	8	2	9	8	7	9	2	3

Count

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

B

	2	3						

Cumulated_count

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

Counting Sort

A

7	8	2	9	8	7	9	2	3

Count

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

B

	2	3						9

Cumulated_count

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

Counting Sort

A

7	8	2	9	8	7	9	2	3

Count

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

B

	2	3		7				9

Cumulated_count

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

Counting Sort

A

7	8	2	9	8	7	9	2	3

Count

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

B

	2	3		7		8		9

Cumulated_count

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

Counting Sort

A

7	8	2	9	8	7	9	2	3

Count

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

B

	2	3		7		8	9	9

Cumulated_count

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

Counting Sort

A

7	8	2	9	8	7	9	2	3

Count

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

B

2	2	3		7		8	9	9

Cumulated_count

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

Counting Sort

A

7	8	2	9	8	7	9	2	3

Count

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

B

2	2	3		7	8	8	9	9

Cumulated_count

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

Counting Sort

A

7	8	2	9	8	7	9	2	3

Count

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

B

2	2	3	7	7	8	8	9	9

Cumulated_count

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

Counting Sort

A

7	8	2	9	8	7	9	2	3

Count

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

B

2	2	3	7	7	8	8	9	9

Cumulated_count

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

Counting Sort

A

7	8	2	9	8	7	9	2	3

Count

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

B

2	2	3	7	7	8	8	9	9

Cumulated_count

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

Time Complexity

- Compute c_{ount} and $c_{umulated\_count}$: $\Theta(k)$
- Each of $a_1,\ldots,a_n$ can be put into correct position in $\Theta(1)$. $\Theta(n)$ in total
- We can conclude it takes $\Theta(n+k)$ time.
- Stable: Homework

Radix Sort

MSD: Most Significant Digit

LSD: Least Significant Digit

- Suppose all keys are d digits number based on k.
- We can sort n numbers by d stable sorts
  - Sort them according to LSD
  - Sort them according to 2^{nd}-LSD.
  - …
  - Sort them according to 2^{nd}-MSD.
  - Sort them according to MSD.
- Note: Counting sort is stable.

Radix Sort

A

17	28	32	29	28	17	19	52	43

Queue Array

0	1	2	3	4	5	6	7	8	9

Radix Sort

LSD: Least Significant Digit

A

17	28	32	29	28	17	19	52	43

Queue Array

0	1	2	3	4	5	6	7	8	9
		32	43				17	28	29
		52					17	28	19

Radix Sort

A

17	28	32	29	28	17	19	52	43

32	52	43	17	17	28	28	29	19

Queue Array

0	1	2	3	4	5	6	7	8	9

Radix Sort

MSD: Most Significant Digit

A

17	28	32	29	28	17	19	52	43

32	52	43	17	17	28	28	29	19

Queue Array

0	1	2	3	4	5	6	7	8	9
	17	28	32	43	52				
	17	28							
	19	29							

Radix Sort

A

17	28	32	29	28	17	19	52	43

32	52	43	17	17	28	28	29	19

17	17	19	28	28	29	32	43	52

Queue Array

0	1	2	3	4	5	6	7	8	9

Time Complexity

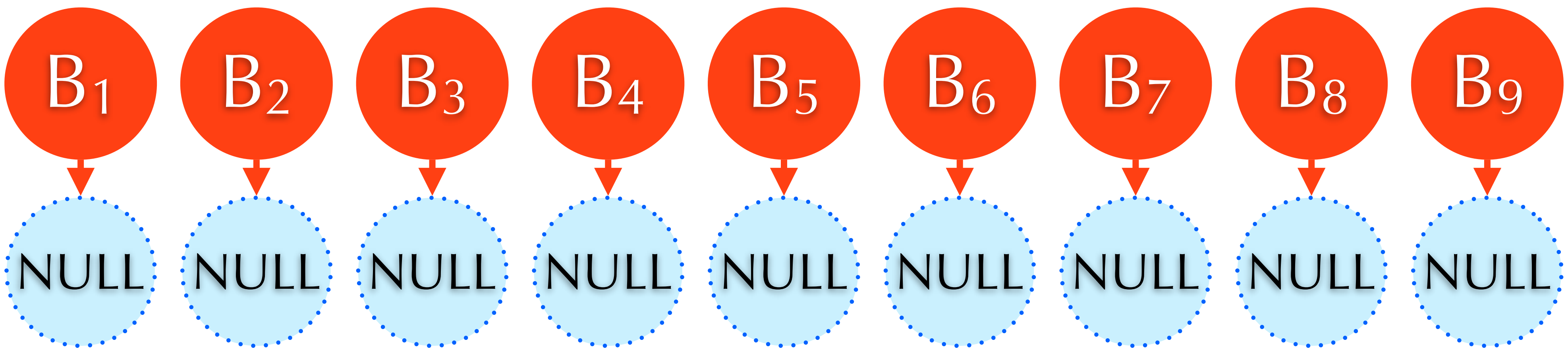
- Radix sort = d times counting sort
  - base-k
- $T(n)=d\Theta(n+k)=\Theta(d(n+k))$
- $T(n)=O(n)$ if $d=O(1)$ and $k=O(1)$.
- Question 1: How to sort integers $a_1,\ldots,a_n\in[0,n^{64})$ in $O(n)$ time?
- Question 2: Is radix sort faster than quick sort?

Bucket Sort

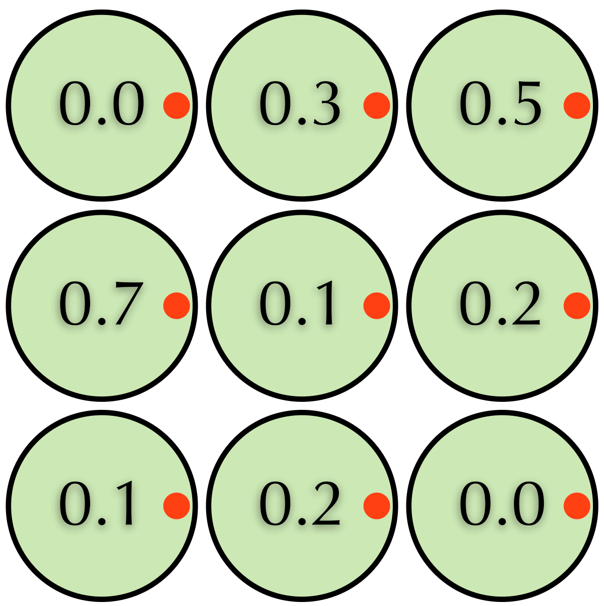
- Assume $a_1,\ldots,a_n \in [0,1)$ (Normalization)
- Prepare n buckets (sorted list) $B_1,\ldots,B_n$.
- For i = 1 to n do

```
For i = 1 to n do
    j=⌊nai⌋+1
    insert ai into Bj
loop
Concatenate B1,B2,…,Bn into ⟨b1,…,bn⟩
```

Bucket Sort

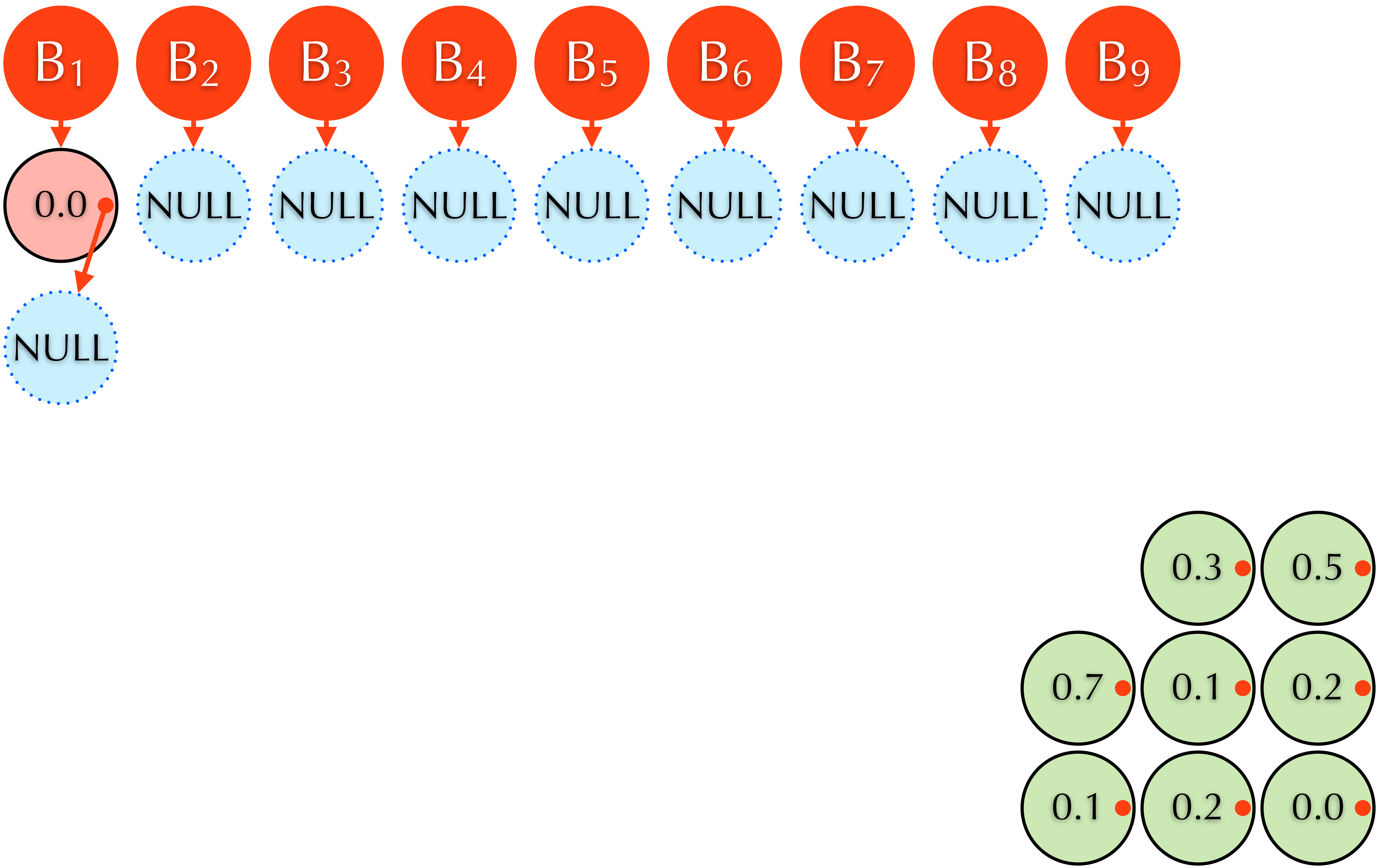


B1
B2
B3
B4
B5
B6
B7
B8
B9
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL

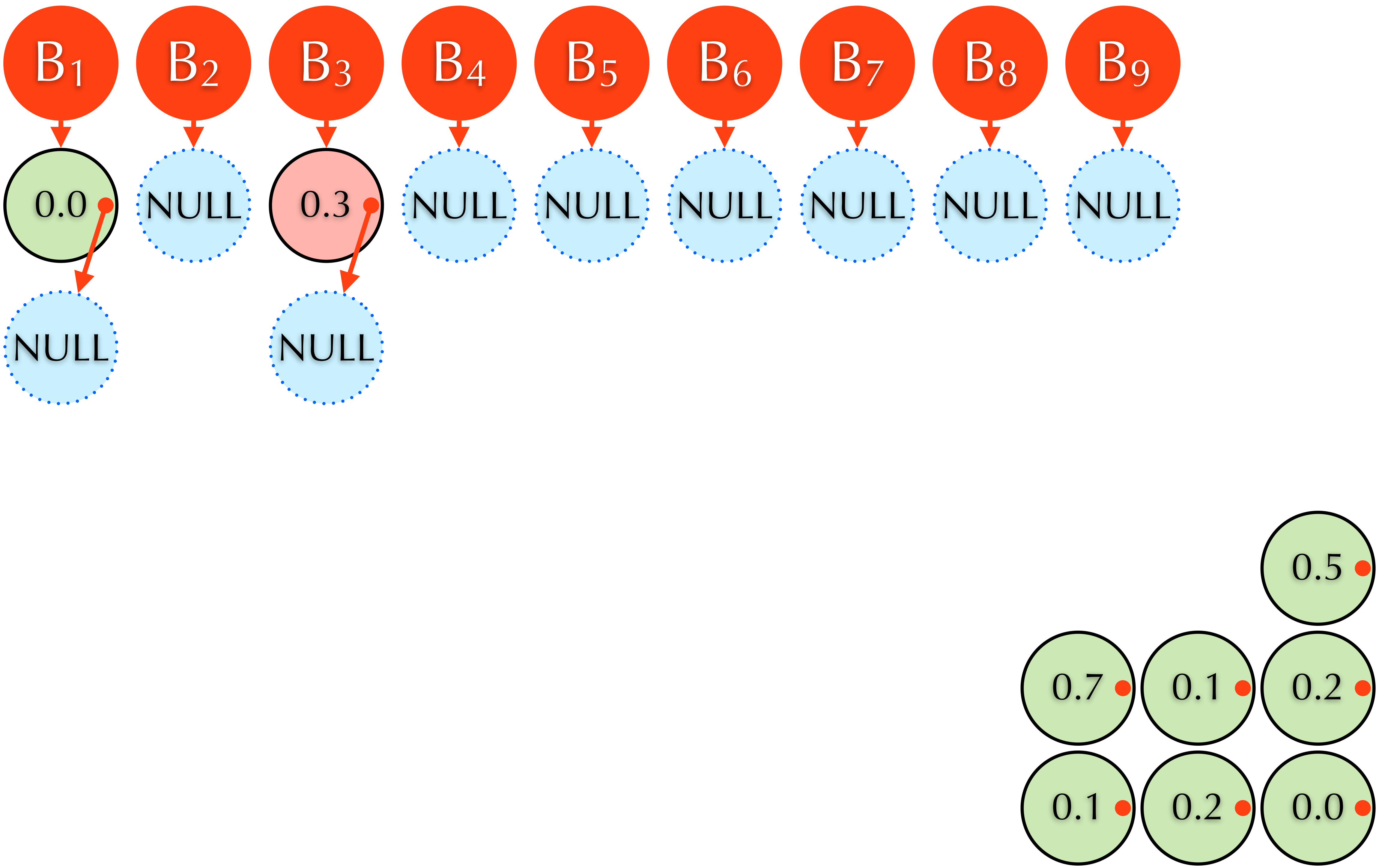


0.0
0.3
0.5
0.7
0.1
0.2
0.1
0.2
0.0

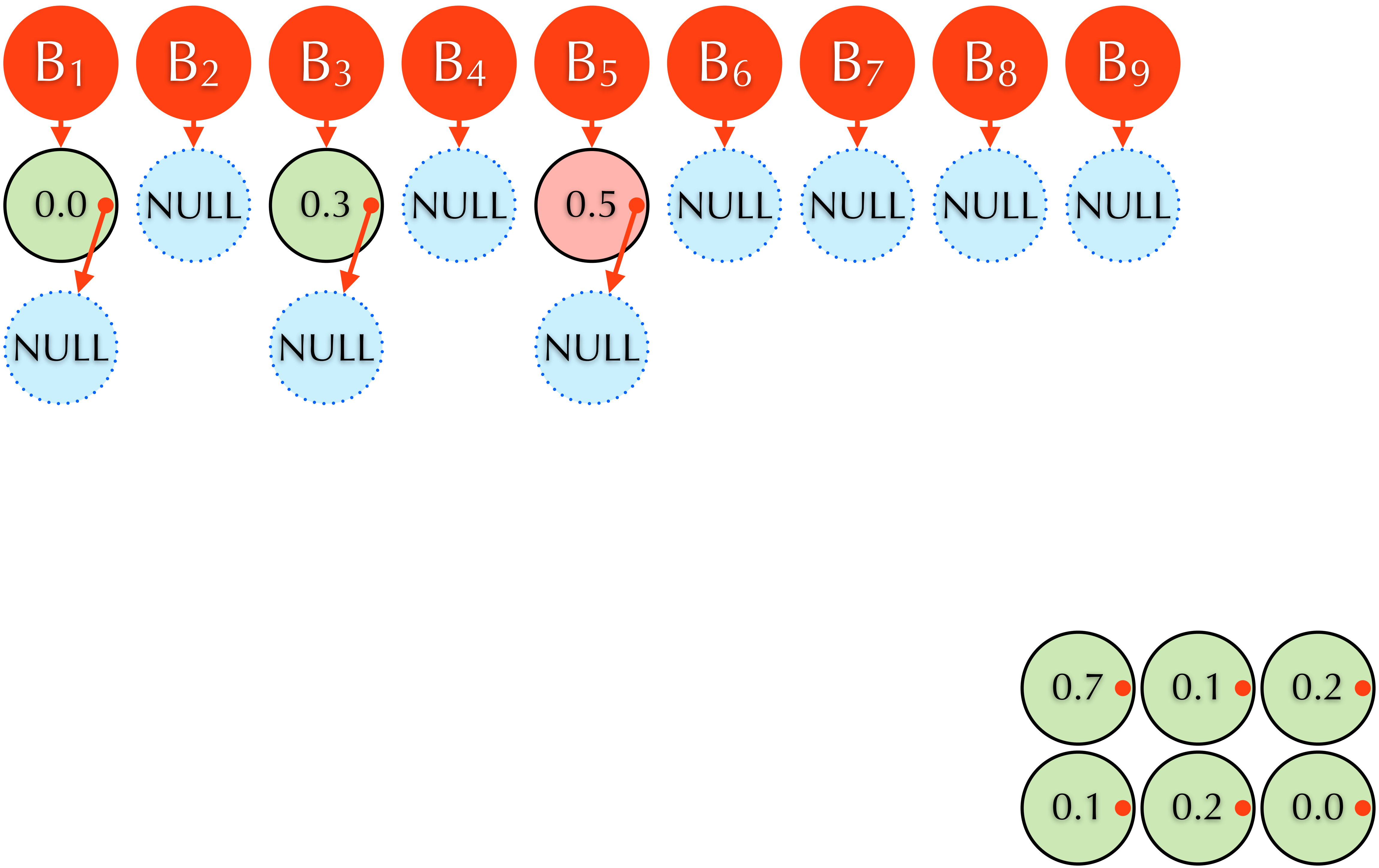
Bucket Sort


B1
B2
B3
B4
B5
B6
B7
B8
B9
0.0
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
0.3
0.5
0.7
0.1
0.2
0.1
0.2
0.0

Bucket Sort

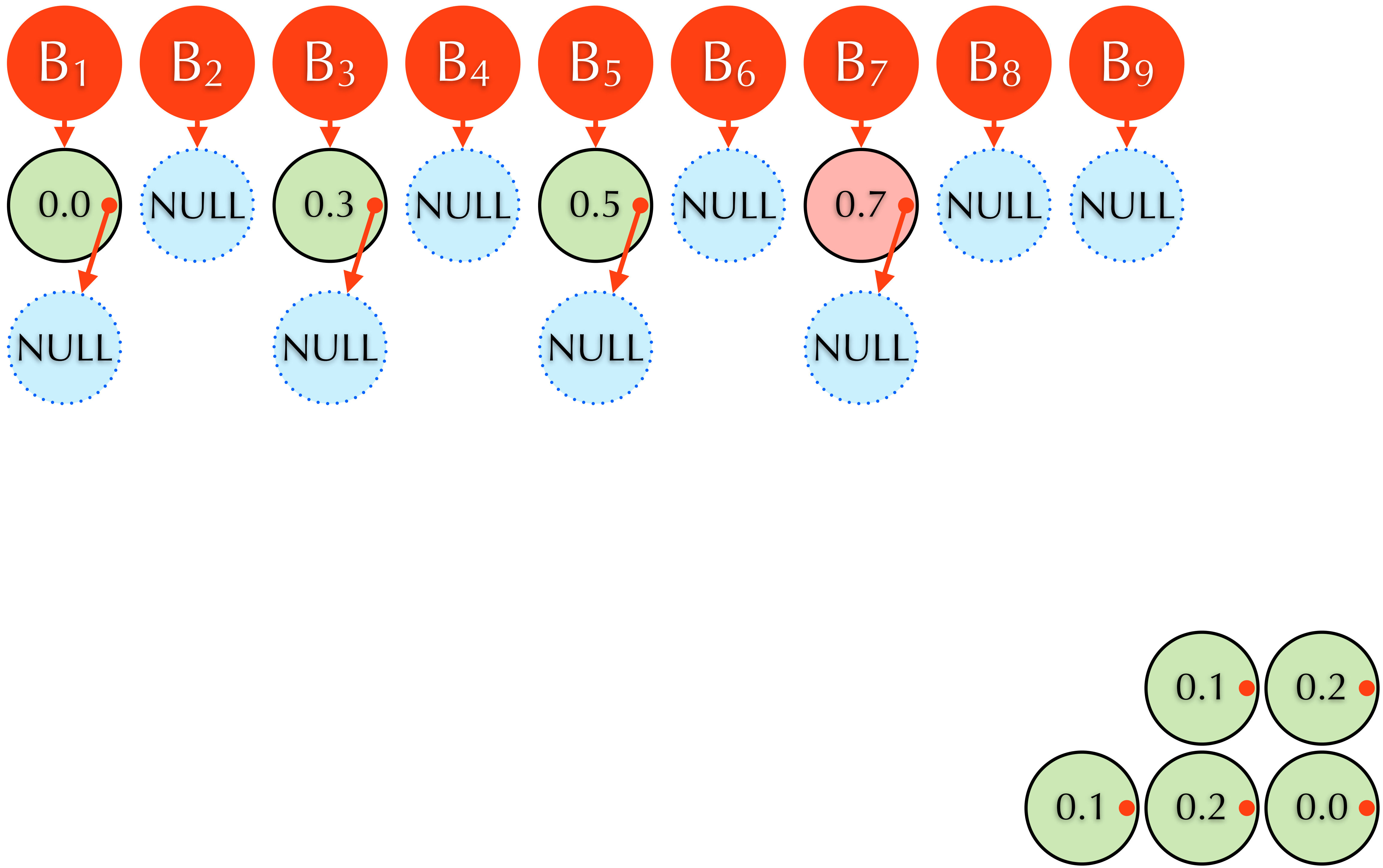

B1
B2
B3
B4
B5
B6
B7
B8
B9
0.0
NULL
0.3
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
0.5
0.7
0.1
0.2
0.1
0.2
0.0

Bucket Sort



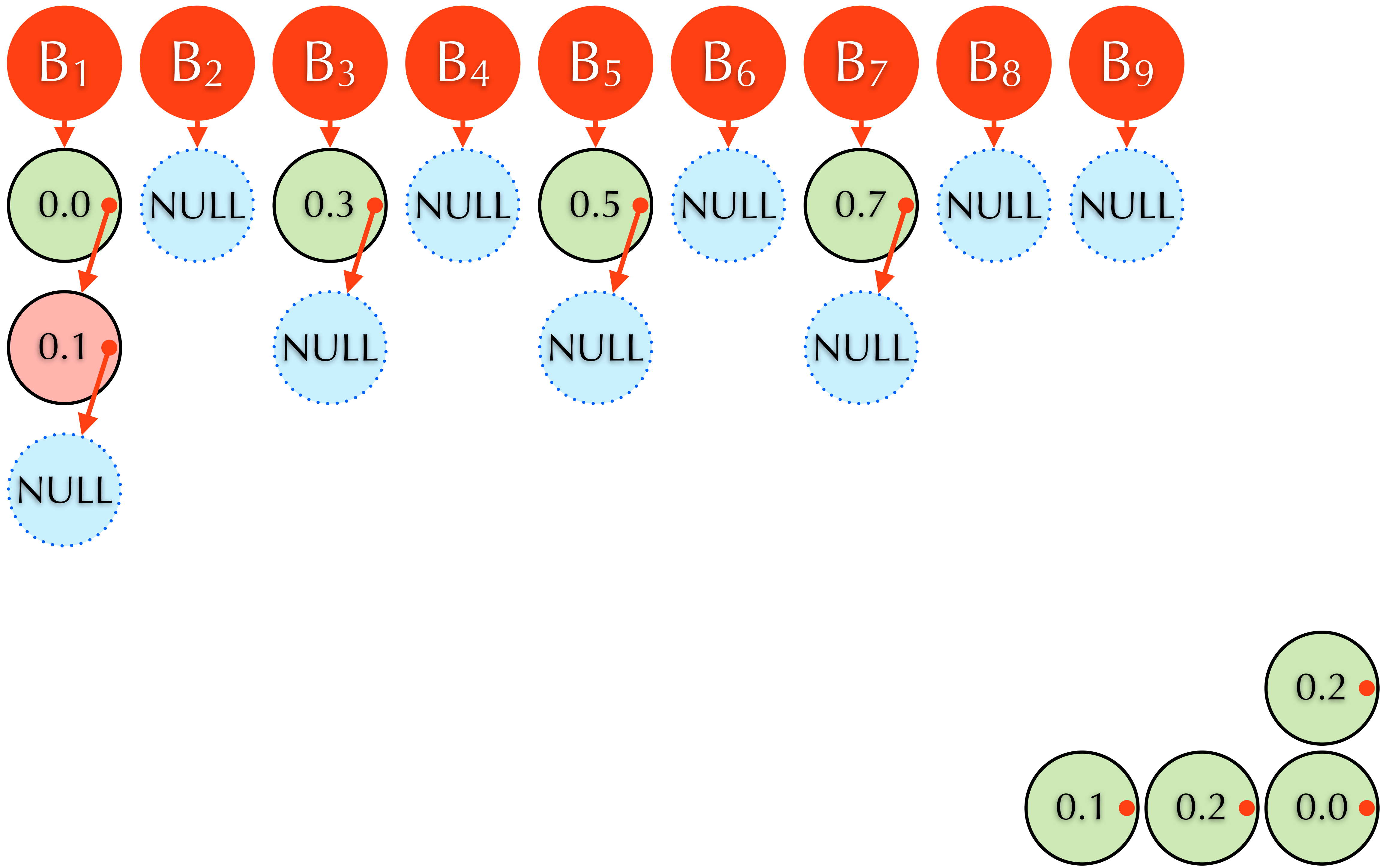
B1
B2
B3
B4
B5
B6
B7
B8
B9
0.0
NULL
0.3
NULL
0.5
NULL
NULL
NULL
NULL
NULL
NULL
NULL
0.7
0.1
0.2
0.1
0.2
0.0

Bucket Sort

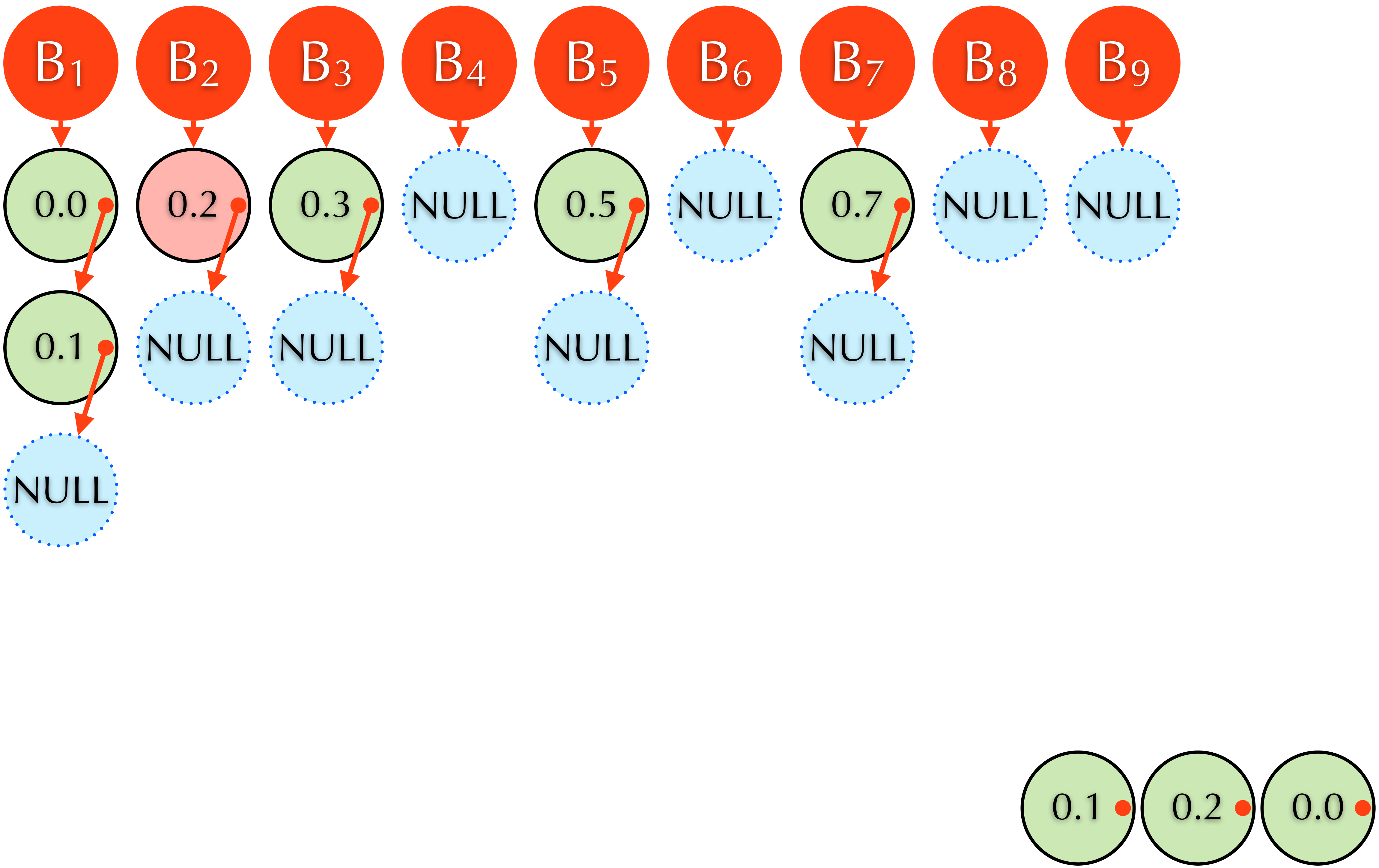


B1
B2
B3
B4
B5
B6
B7
B8
B9
0.0
NULL
0.3
NULL
0.5
NULL
0.7
NULL
NULL
NULL
NULL
NULL
NULL
0.1
0.2
0.1
0.2
0.0

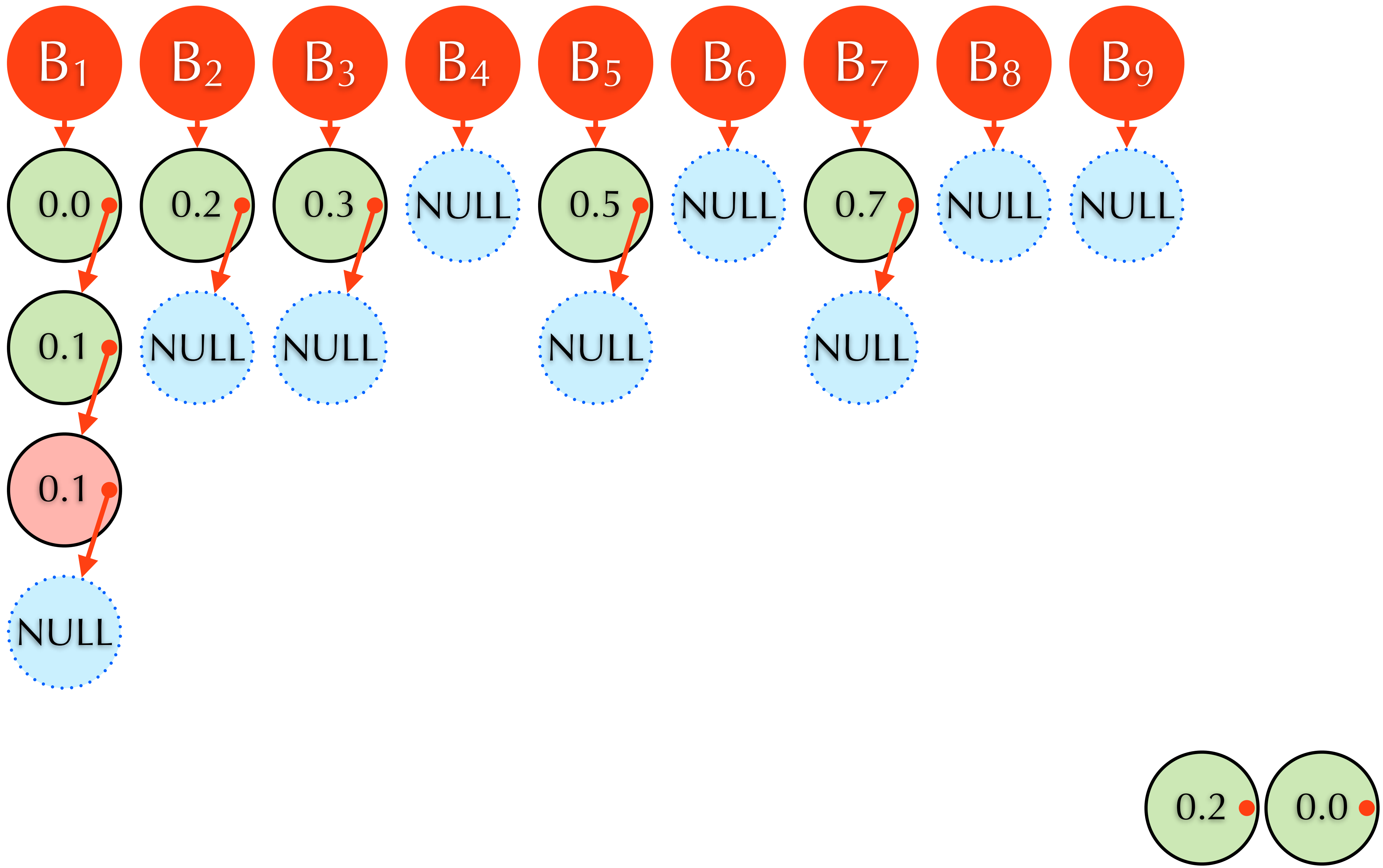
Bucket Sort


B1
B2
B3
B4
B5
B6
B7
B8
B9
0.0
NULL
0.3
NULL
0.5
NULL
0.7
NULL
NULL
0.1
NULL
NULL
NULL
NULL
0.2
0.1
0.2
0.0

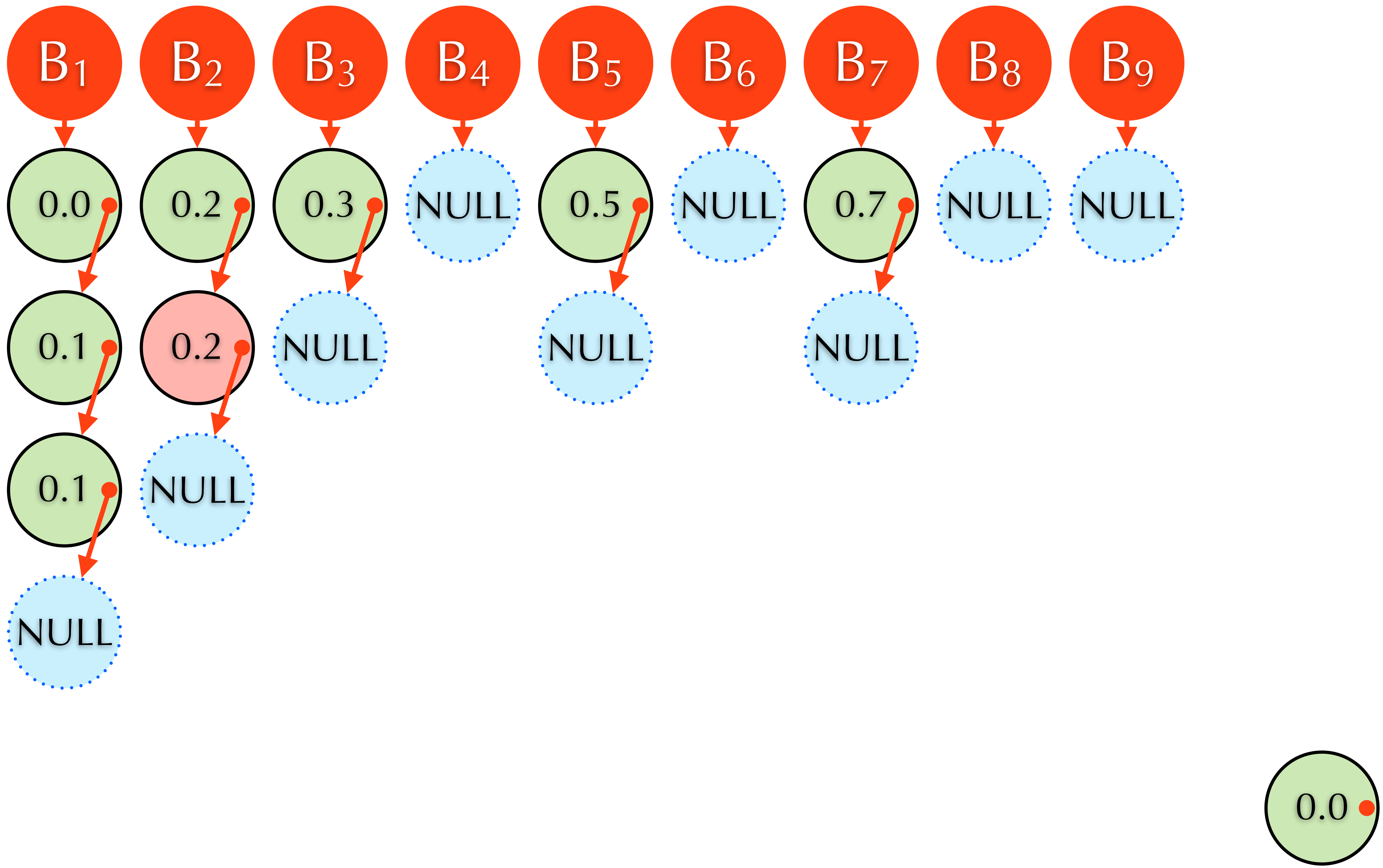
Bucket Sort


B1
B2
B3
B4
B5
B6
B7
B8
B9
0.0
0.2
0.3
NULL
0.5
NULL
0.7
NULL
NULL
0.1
NULL
NULL
NULL
NULL
NULL
0.1
0.2
0.0

Bucket Sort

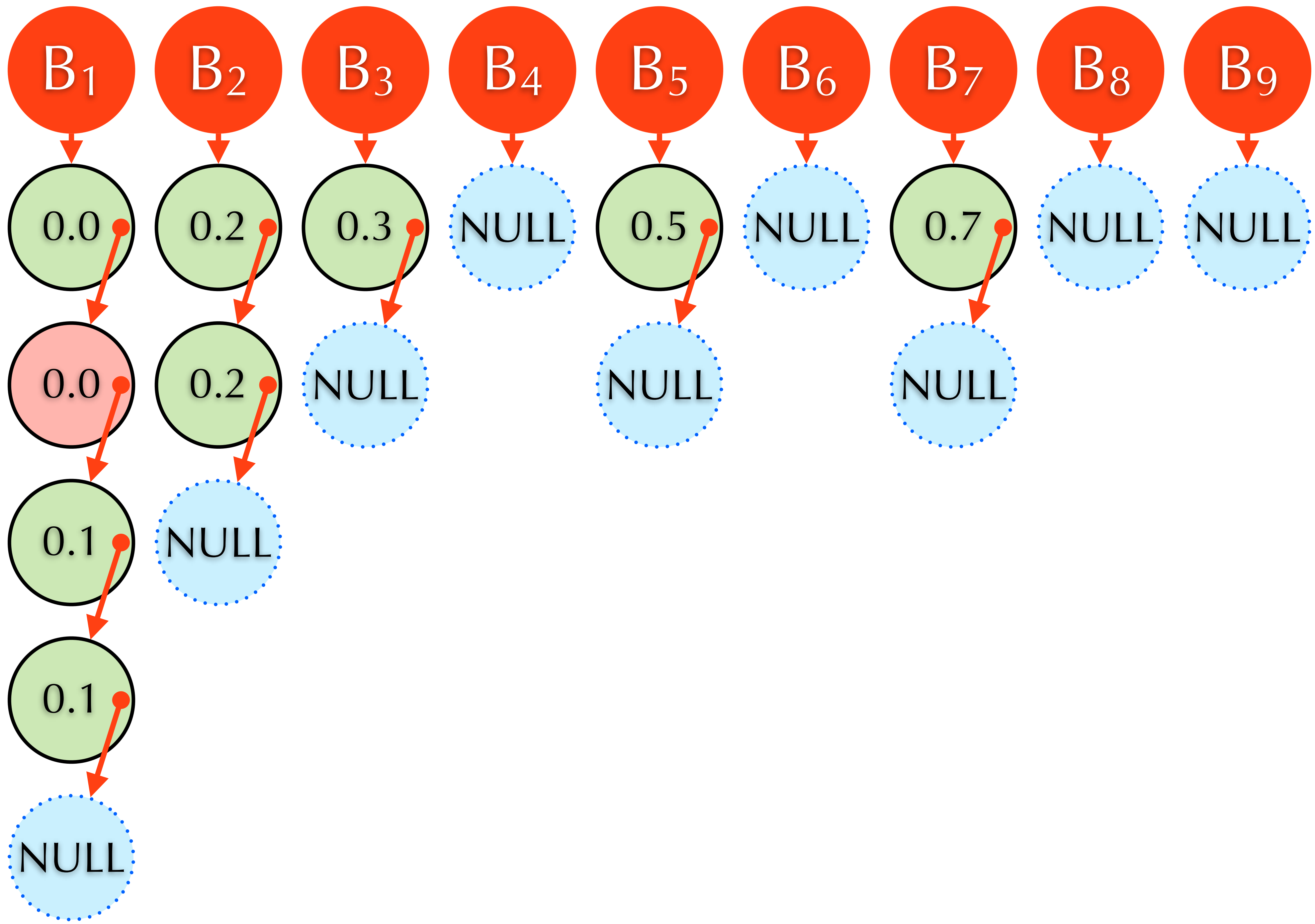

B1
B2
B3
B4
B5
B6
B7
B8
B9
0.0
0.2
0.3
NULL
0.5
NULL
0.7
NULL
NULL
0.1
NULL
NULL
NULL
NULL
0.1
NULL
0.2
0.0

Bucket Sort



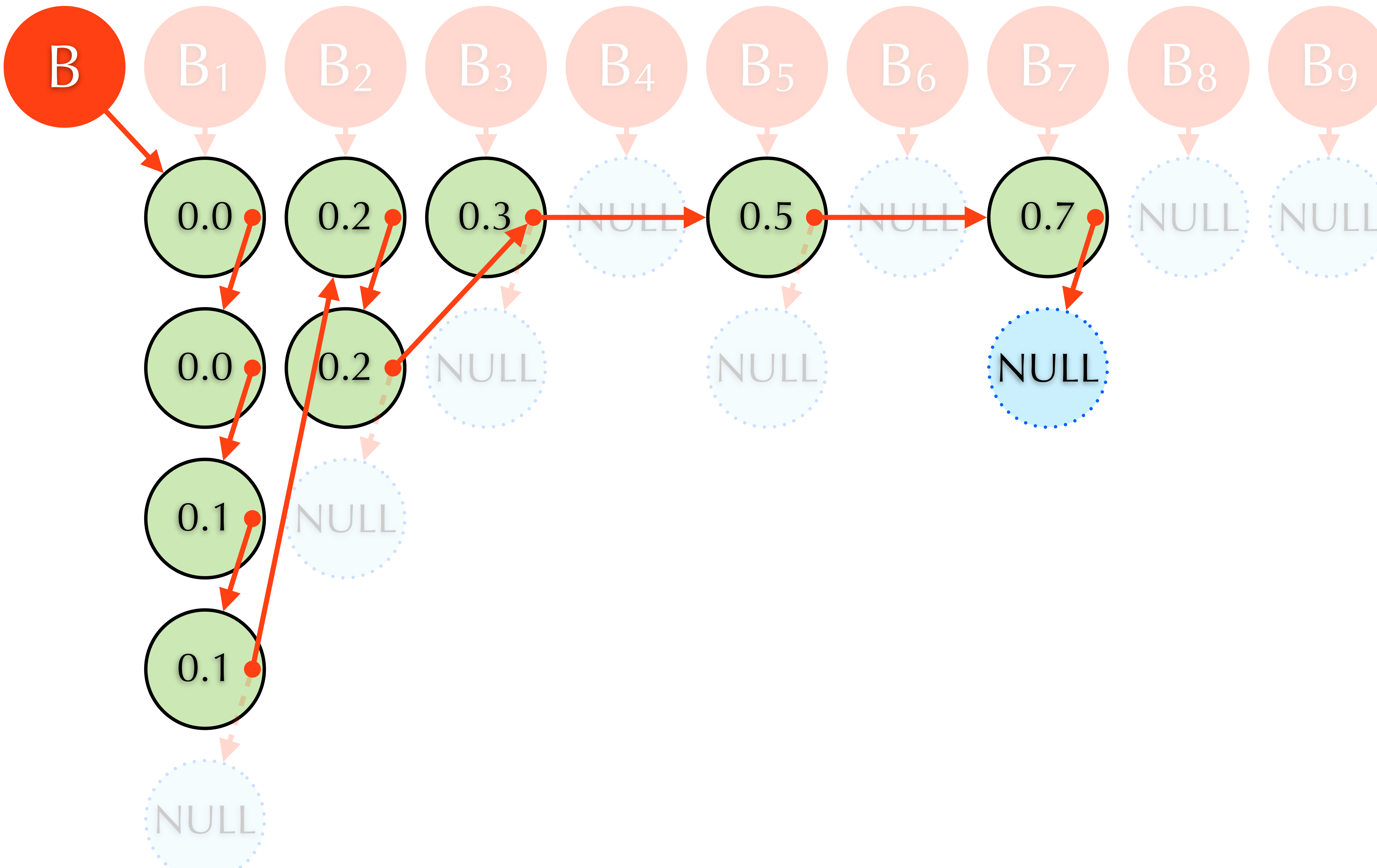
B1
B2
B3
B4
B5
B6
B7
B8
B9
0.0
0.2
0.3
NULL
0.5
NULL
0.7
NULL
NULL
0.1
0.2
NULL
NULL
NULL
0.1
NULL
NULL
0.0

Bucket Sort



B1
B2
B3
B4
B5
B6
B7
B8
B9
0.0
0.2
0.3
NULL
0.5
NULL
0.7
NULL
NULL
0.0
0.2
NULL
NULL
NULL
0.1
NULL
0.1
NULL

Bucket Sort



B
B1
B2
B3
B4
B5
B6
B7
B8
B9
0.0
0.2
0.3
NULL
0.5
NULL
0.7
NULL
NULL
0.0
0.2
NULL
NULL
NULL
0.1
NULL
0.1
NULL

Time Complexity

- $|B_i|$: size of B_i (number of elements in B_i)
- $T(n)=\Theta(n)+\Sigma_{1\leq i\leq n}\Theta(|B_i|^2)=\Sigma_{1\leq i\leq n}\Theta(|B_i|^2)$
- Worst case: all numbers are in one bucket
  - $T(n)=\Theta(n^2)$
- Average case:
  - $a_1,\ldots,a_n$ are independently uniformly randomly sampled from $[0,1)$. $\boxed{\Pr[a_j \in B_i]=1/n}$
  - $T(n)=E[\Sigma_{1\leq i\leq n}\Theta(|B_i|^2)]$ ... Expectation!
    $=\Theta(E[\Sigma_{1\leq i\leq n}|B_i|^2])$

Average Case

- Goal: $E[\Sigma_{1\leq i\leq n}|B_i|^2]=\Theta(n)$
- Let $X_{i,j}$ be the random variable indicating whether $a_j \in B_i$. I.e., $X_{i,j}=1$ if $a_j \in B_i$ and $X_{i,j}=0$ if $a_j \notin B_i$. $E[X_{i,j}]=Pr[a_j \in B_i]=1/n$
- $|B_i|=\Sigma_{1\leq j\leq n}X_{i,j}$
- $|B_i|^2=(\Sigma_{1\leq j\leq n}X_{i,j})(\Sigma_{1\leq j\leq n}X_{i,j})$
  $=\Sigma_{1\leq j\leq n}\Sigma_{1\leq k\leq n}X_{i,j}X_{i,k}$
  $=\Sigma_{1\leq j\leq n}X_{i,j}X_{i,j}+2\Sigma_{1\leq k<j\leq n}X_{i,j}X_{i,k}$
  $=\Sigma_{1\leq j\leq n}X_{i,j}+2\Sigma_{1\leq k<j\leq n}X_{i,j}X_{i,k}$

Average Case

- $E[|B_i|^2]=E[\Sigma_{1\leq j\leq n}X_{i,j}]+2E[\Sigma_{1\leq k<j\leq n}X_{i,j}X_{i,k}]$
- $E[\Sigma_{1\leq j\leq n}X_{i,j}]=n\times(1/n)=1$
- $2E[\Sigma_{1\leq k<j\leq n}X_{i,j}X_{i,k}]$
  $=n(n-1)E[X_{i,j}X_{i,k}]$ .... for $k<j$
  $\leq n^2E[X_{i,j}]E[X_{i,k}]$ ... $X_{i,j}$ & $X_{i,k}$ are independent
  $=n^2(1/n)(1/n)=1$
- $1\leq E[|B_i|^2]\leq 2$
- $n\leq E[\Sigma_{1\leq i\leq n}|B_i|^2]\leq 2n$ ...... $T(n)=\Theta(n)$