Design and Analysis of Data Structures and Algorithms :: Sorting part 1

อ.ดร.วรินทร์ วัฒนพรพรหม

Sorting

- การจัดเรียงข้อมูลเป็นขั้นตอนหนึ่งที่สำคัญในการประมวลผลข้อมูล
- ข้อมูลที่จัดเรียงแล้วช่วยให้เราทำงานกับข้อมูลนั้นได้ง่ายขึ้นเช่น

การแสดงผล

การคำนวณ

ประโยชน์ด้านการแสดงผล

Southern Bell

305 666-6637 615
JAN 07, 1984 R56 044
PAGE 5 OF 6

DETAIL OF ITEMIZED CALLS

NO.	DATE	TIME		PLACE	AREA-NUMBER	*	MIN	AMOUNT
**SOUTHERN BELL								
1.	DEC 12	506PM	TO	FLINT	MI 313 257-7836	E	1	.38
2.	DEC 12	1059PM	TO	FLUSHING	MI 313 659-2990	EM	56	10.06
3.	DEC 13	1243AM	TO	FLUSHING	MI 313 659-2990	N	8	1.48
4.	DEC 15	1158AM	TO	FLINT	MI 313 257-7836	D	1	.64
5.	DEC 16	558AM	TO	FLUSHING	MI 313 659-2990	N	6	1.13

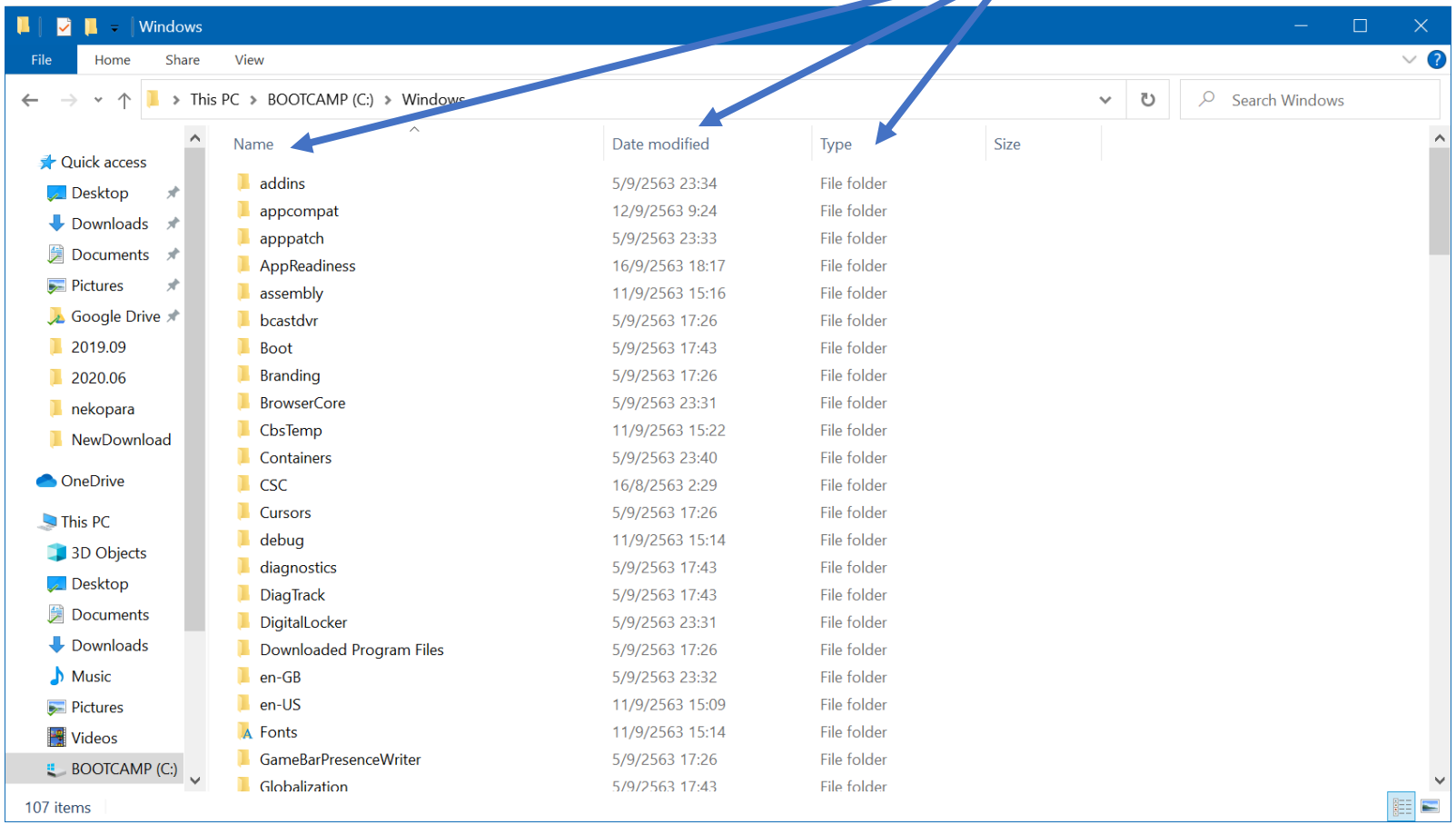
SOUTHERN BELL TOTAL CHARGE FOR ITEMIZED CALLS 13.69
TOTAL TAX: FED .41 STA .00
BILLING INQUIRIES CALL 263-5255

จัดเรียงตามวันที่

Z1L
PLEASE RETURN ENCLOSED CARD WITH YOUR PAYMENT
CONTINUED

ประโยชน์ด้านการค้นหา

จัดเรียงตาม Name, Date, Type



ประโยชน์ด้านการคำนวณ

- การหาค่า Median (ค่าข้อมูล ณ. ตำแหน่งกลางของชุดข้อมูล)

20 25 45 46 49 55 65 73 80 92 101

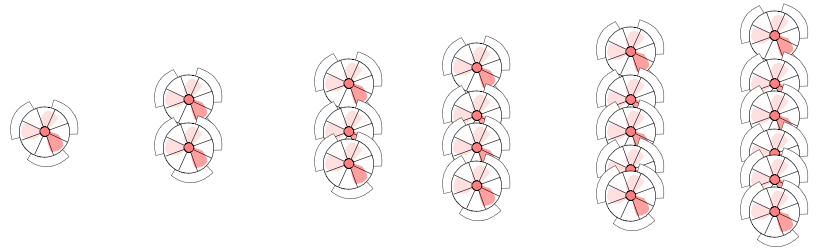
- การหาค่า Maximum

20 25 45 46 49 55 65 73 80 92 101

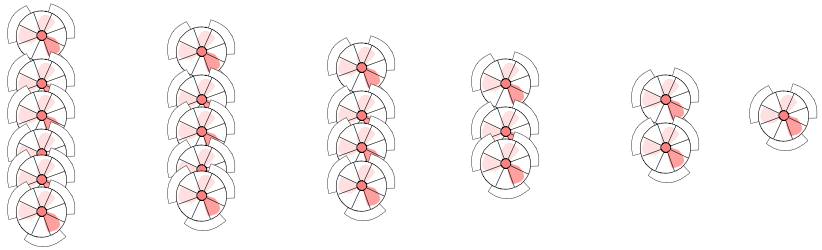
- หากข้อมูลไม่ได้จัดเรียงต้องใช้เวลา $O(n)$ เพื่อหาค่า Max
- แต่เมื่อข้อมูลจัดเรียงเรียบร้อยแล้วใช้เวลา $O(1)$

Ascending VS. Descending Order

- เราสามารถจัดเรียงข้อมูลได้ทุกประเภท int, float, boolean, character, string ฯลฯ
- ข้อมูลที่จัดเรียงต้องมี Key ที่ใช้เป็นตัววัดลำดับข้อมูลเช่น จำนวน วันที่ ชื่อ เป็นต้น
- การจัดเรียงจากน้อยไปหามาก (Ascending Order)
    - เช่นเรียงตัวเลข, วันที่ในรายการสมุดบัญชีธนาคาร



- การจัดเรียงมากไปหาน้อย (Descending Order)
    - เช่น เรียง E-mail โดยเอาวันที่ล่าสุดขึ้นมาก่อน



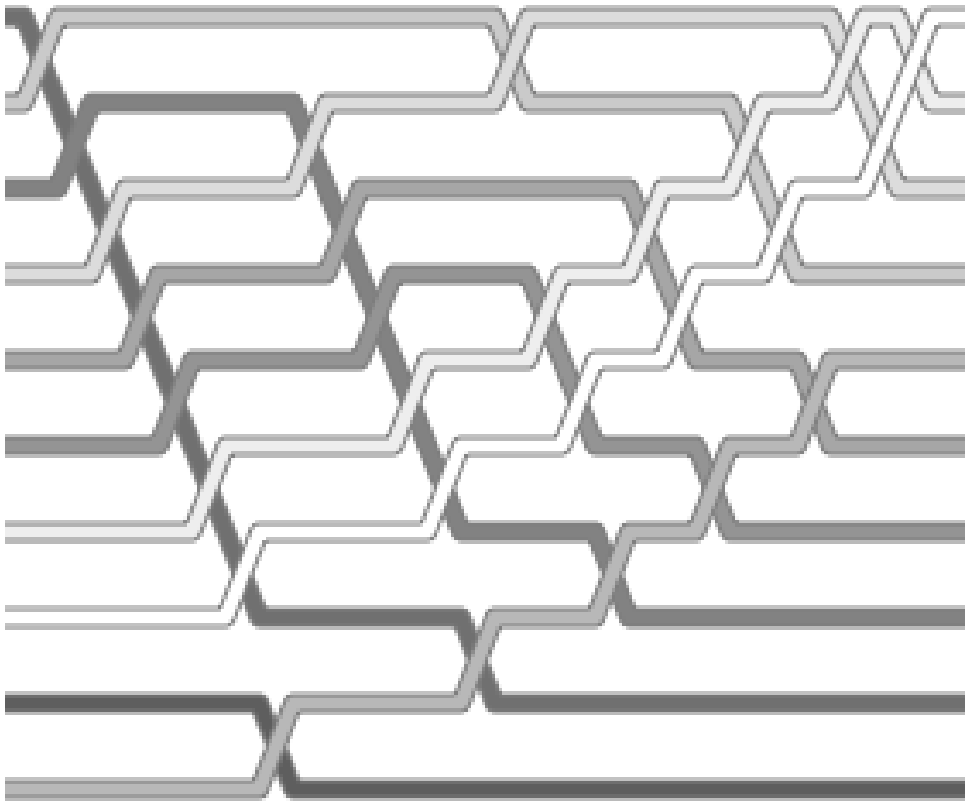
ความหมาย

- การจัดเรียงลำดับ (Sorting) หมายถึงการจัดเรียงข้อมูล ให้เรียงลำดับตามเงื่อนไขที่กำหนดไว้ (มากไปน้อย หรือ น้อยไปมาก)
- ในกรณีที่ข้อมูลในแต่ละ Record มีหลาย Field เราต้องพิจารณาเลือก Field ที่สนใจเพื่อใช้ในการเรียงลำดับ เช่น การจัดเรียงลำดับประวัตินักศึกษา อาจใช้หมายเลขประจำตัวของนักศึกษาเป็น Field โดยเรียงจากน้อยไปมาก เป็นต้น

วิธีการจัดเรียงข้อมูล พื้นฐาน

- การจัดเรียงแบบแลกเปลี่ยน (Exchange Sort/Bubble Sort)
- การจัดเรียงแบบแทรก (Insertion Sort)
- การจัดเรียงแบบเลือก (Selection Sort)

Bubble Sort (Exchange Sort)



การจัดเรียงแบบแลกเปลี่ยน หรือแบบฟอง (Bubble Sort)

- เป็นการจัดเรียงโดยการเปรียบเทียบค่า 2 ค่าที่ติดกัน
- ทำต่อเนื่องกันไปเรื่อย ๆ และตัดสินใจว่าจะสลับตำแหน่งกันหรือไม่ เช่น ถ้าต้องการเรียงข้อมูลจากน้อยไปมาก ก็คือ
  - ข้อมูลที่มีค่าน้อย ต้องอยู่ในตำแหน่งหน้า
  - ข้อมูลที่มีค่ามาก จะอยู่ตำแหน่งหลัง
  - ข้อมูล 2 ตัวที่อยู่ติดกัน ถ้า
    - ถ้าข้อมูลตัวแรกมากกว่าตัวหลัง ก็จะต้องสลับตำแหน่งกัน
    - แต่ถ้าข้อมูลตัวแรกน้อยกว่าข้อมูลตัวหลัง ก็ไม่ต้องสลับตำแหน่ง
    - ทำเช่นนี้ซ้ำกันไปเรื่อย ๆ จนกว่าการเปรียบเทียบของข้อมูลตลอดทั้งชุดจะไม่ต้องมีการสลับตำแหน่งเลย

Bubble Sort

- แนวคิด คือค่าที่มากๆ จะต้องถูกนำไป (ลอยไป) ไว้ด้านท้าย
- เหมือนลูกโป่งที่ขนาดใหญ่จะลอยได้เร็วและสูง
  - เริ่มนำข้อมูลตัวแรกเทียบกับตัวที่ 2 ตัวไหนมากก็จะถูกสลับกัน ทำอย่างนี้ไปจนถึงตัวสุดท้าย เราจะได้ค่าที่มากที่สุด 1 ตัวไว้ด้านท้าย
  - แต่ละครั้งจะได้ค่ามากที่สุดไปไว้ท้ายสุด
  - จากนั้นเริ่มการเปรียบเทียบใหม่ตั้งแต่ตัวแรกถึงตัวที่ n-1
  - จากนั้นเริ่มการเปรียบเทียบใหม่ตั้งแต่ตัวแรกถึงตัวที่ n-2
  - ...
  - จากนั้นเริ่มการเปรียบเทียบใหม่ตั้งแต่ตัวแรกถึงตัวที่ n-x
  - ทำจน x = n-1

Bubble Sort

```
void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
    int i,j;
    for (i=0;i<n-1;i++)
    // Last i elements are already in place
    for (j=0;j<n-i-1;j++)
        if (arr[j]> arr[j+1])
            swap(&arr[j],&arr[j+1]);
}
```

Driver code

```
/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i<size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver code
int main()
{
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n); // change it
    cout<<"Sorted array: \n";
    printArray(arr, n);
    return 0;
}
```

ประสิทธิภาพ

- ความเร็ว Big O = ?
- Linked หรือว่า Array ต่างกันไหม
- Singly Linked List หรือว่า Doubly Linked List ล่ะ

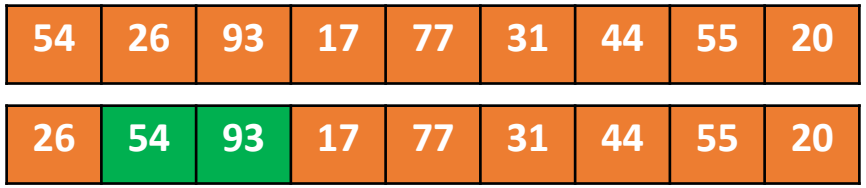
Bubble Sort

First pass


```
void bubbleSort(int arr[], int n)
{
     int i, j;
     for (i = 0; i < n-1; i++)
       for (j = 0; j < n-i-1; j++)
         if (arr[j] > arr[j+1])
             swap(&arr[j], &arr[j+1]);
}
```

Bubble Sort

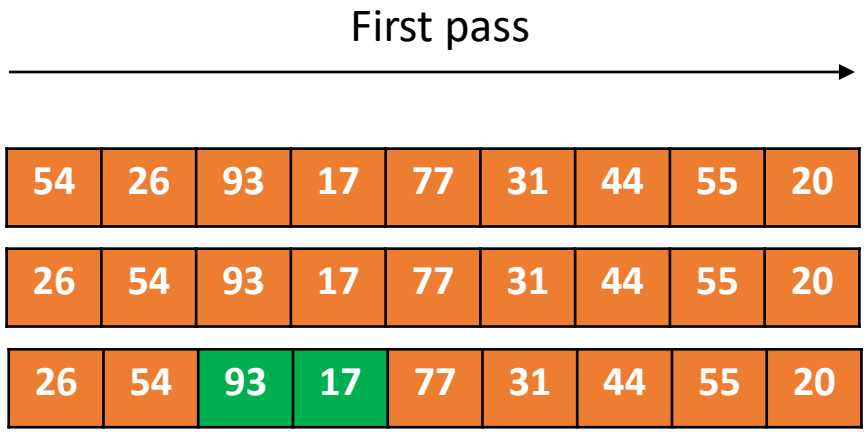
First pass




```
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n-1; i++)
      for (j = 0; j < n-i-1; j++)
        if (arr[j] > arr[j+1])
            swap(&arr[j], &arr[j+1]);
}
```

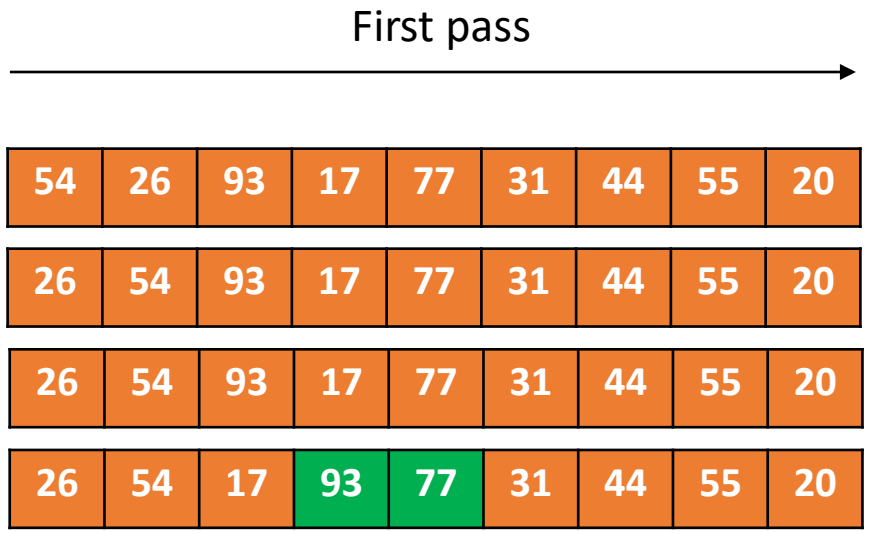
Do nothing

Bubble Sort



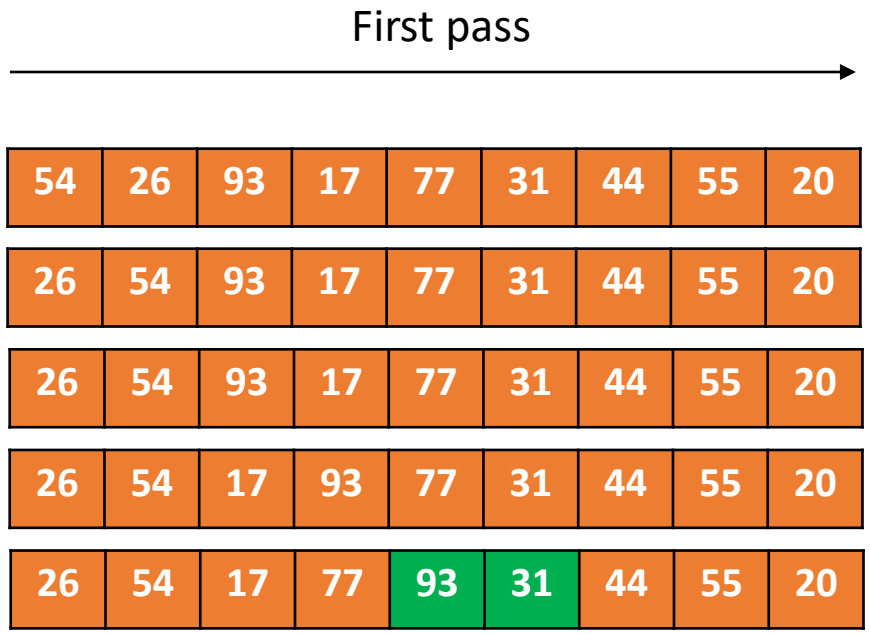

```
void bubbleSort(int arr[], int n)
{
     int i, j;
     for (i = 0; i < n-1; i++)
       for (j = 0; j < n-i-1; j++)
         if (arr[j] > arr[j+1])
             swap(&arr[j], &arr[j+1]);
}
```

Bubble Sort



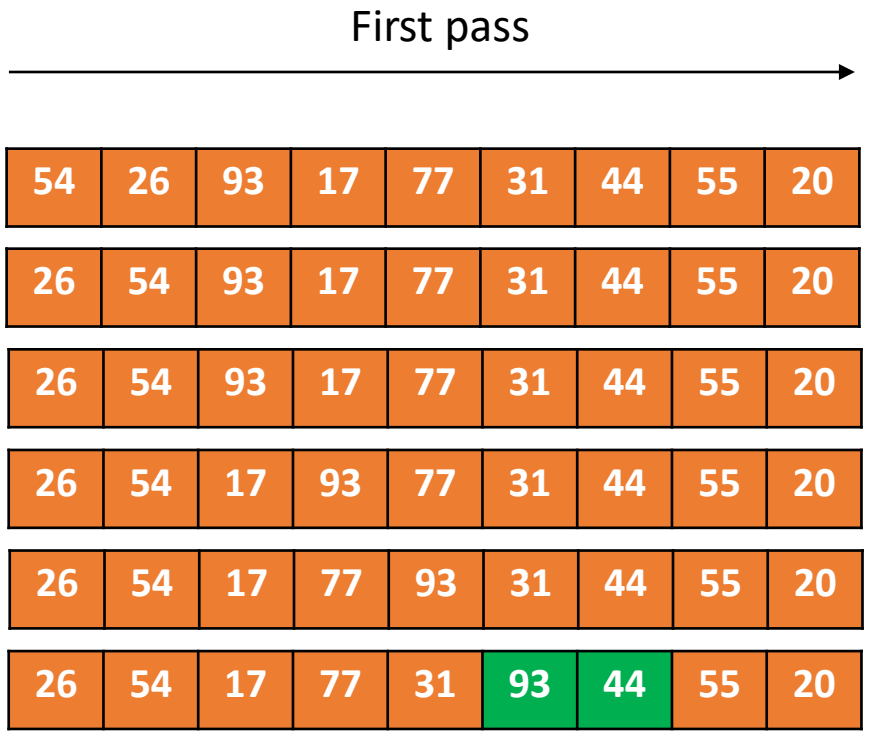

```
void bubbleSort(int arr[], int n)
{
     int i, j;
     for (i = 0; i < n-1; i++)
       for (j = 0; j < n-i-1; j++)
         if (arr[j] > arr[j+1])
             swap(&arr[j], &arr[j+1]);
}
```

Bubble Sort



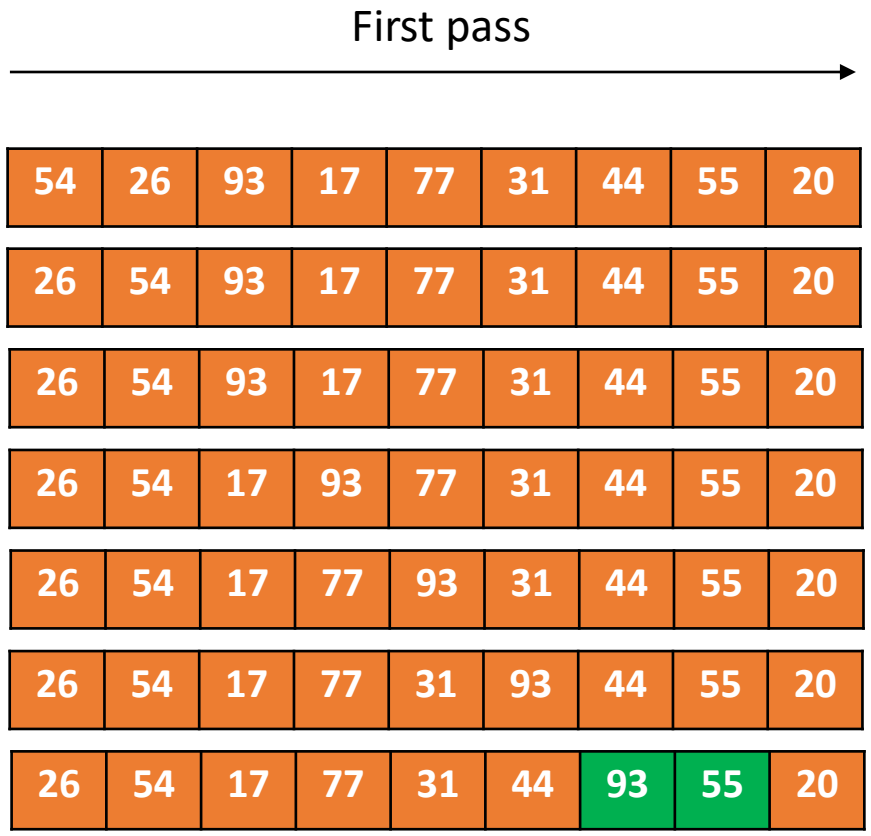

```
void bubbleSort(int arr[], int n)
{
     int i, j;
     for (i = 0; i < n-1; i++)
       for (j = 0; j < n-i-1; j++)
         if (arr[j] > arr[j+1])
             swap(&arr[j], &arr[j+1]);
}
```

Bubble Sort



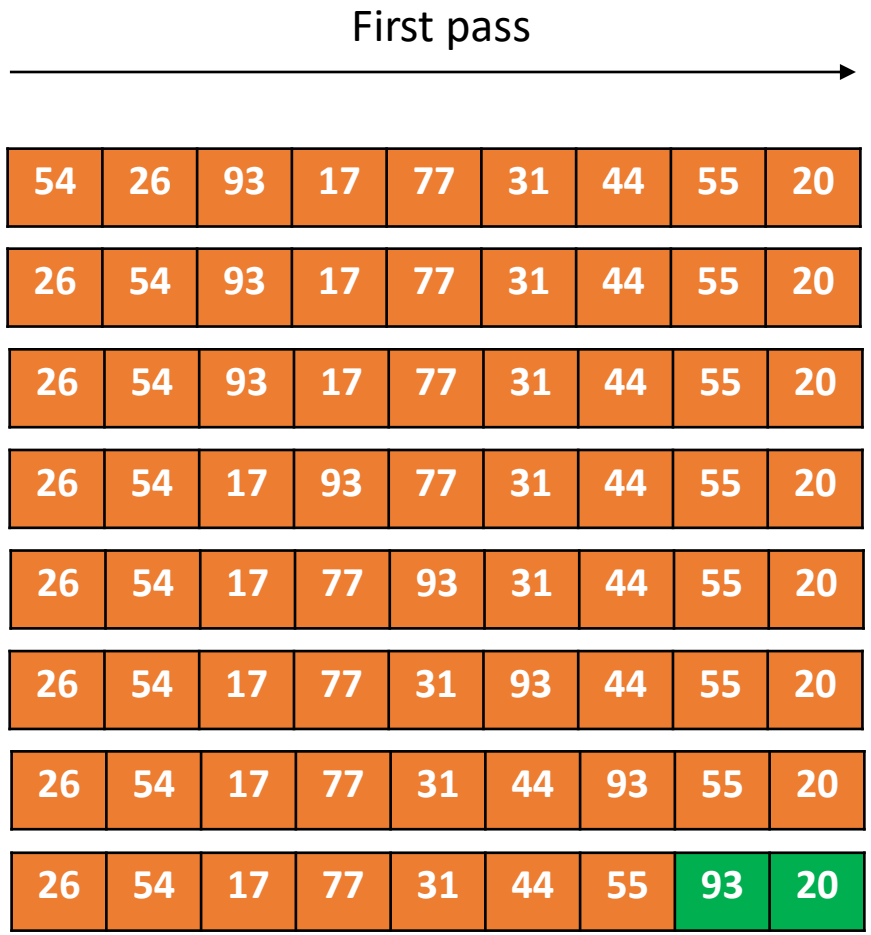

```
void bubbleSort(int arr[], int n)
{
     int i, j;
     for (i = 0; i < n-1; i++)
       for (j = 0; j < n-i-1; j++)
         if (arr[j] > arr[j+1])
             swap(&arr[j], &arr[j+1]);
}
```

Bubble Sort




```
void bubbleSort(int arr[], int n)
{
     int i, j;
     for (i = 0; i < n-1; i++)
       for (j = 0; j < n-i-1; j++)
         if (arr[j] > arr[j+1])
             swap(&arr[j], &arr[j+1]);
}
```

Bubble Sort




```
void bubbleSort(int arr[], int n)
{
     int i, j;
     for (i = 0; i < n-1; i++)
       for (j = 0; j < n-i-1; j++)
         if (arr[j] > arr[j+1])
             swap(&arr[j], &arr[j+1]);
}
```

Bubble Sort


```
void bubbleSort(int arr[], int n)
{
     int i, j;
     for (i = 0; i < n-1; i++)
       for (j = 0; j < n-i-1; j++)
         if (arr[j] > arr[j+1])
             swap(&arr[j], &arr[j+1]);
}
```

93 in place after first pass

Bubble Sort

Second pass


```
void bubbleSort(int arr[], int n)
{
     int i, j;
     for (i = 0; i < n-1; i++)
       for (j = 0; j < n-i-1; j++)
         if (arr[j] > arr[j+1])
             swap(&arr[j], &arr[j+1]);
}
```

คำถาม ใน pass ที่สองเราจะได้ข้อมูลหน้าตาอย่างไร

Bubble Sort

- คำถาม –
  ถ้าเรามีลิสต์ <19, 1, 9, 7, 3, 10, 13, 15, 8, 12> เมื่อเวลาผ่านไปหนึ่งลูป ลิสต์จะมีสภาพเป็นอย่างไร

  (A) <1, 9, 19, 7, 3, 10, 13, 15, 8, 12>

  (B) <1, 3, 7, 9, 10, 8, 12, 13, 15, 19>

  (C) <1, 7, 3, 9, 10, 13, 8, 12, 15, 19>

  (D) <1, 9, 19, 7, 3, 10, 13, 15, 8, 12>

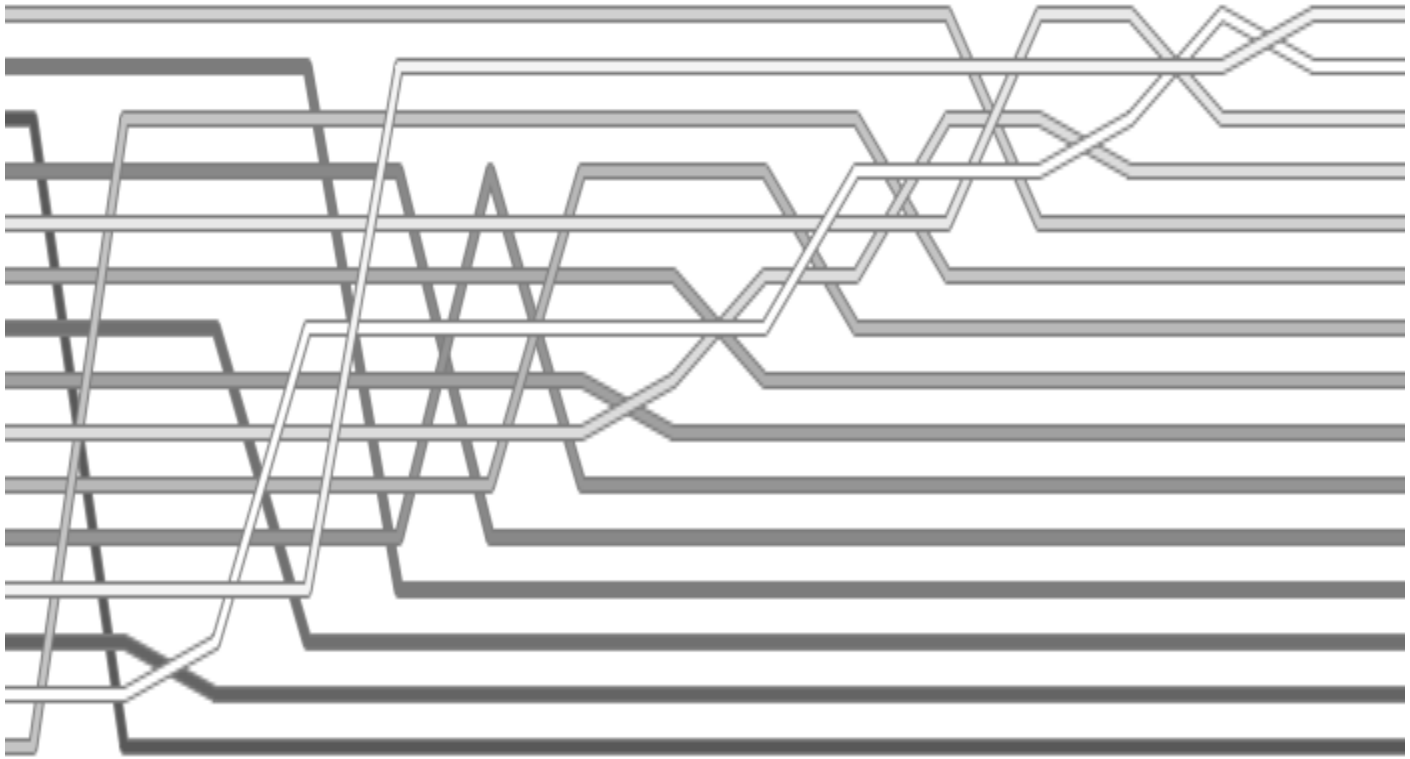
วิเคราะห์ bubble sort

```
for (i = 0; i < n-1; i++)
    for (j = 0; j < n-i-1; j++)
      if (arr[j] > arr[j+1])
          swap(&arr[j], &arr[j+1]);
```

Let n = size of the array

- The outer loop is executed n-1 times (call it n, that's close enough)
- Each time the outer loop is executed, the inner loop is executed
  - Inner loop executes n-1 times at first, linearly dropping to just once
  - On average, inner loop executes about $n/2$ times for each execution of the outer loop
  - In the inner loop, the comparison is always done (constant time), the swap might be done (also constant time)
- Result is $n \times n/2 \times k$, that is, $O(kn^2/2) = O(n^2)$

Selection Sort



การจัดเรียงแบบเลือก (Selection Sort)

- เป็นการจัดเรียงโดยการเริ่มต้นค้นหาข้อมูลตัวที่น้อยที่สุดจากข้อมูลที่มีอยู่ทั้งหมด
- แล้วเอามาเก็บไว้ข้างนอก
- แล้วกลับไปหาข้อมูลตัวที่น้อยที่สุดในกองต่อไปจนกว่าจะหมดกอง

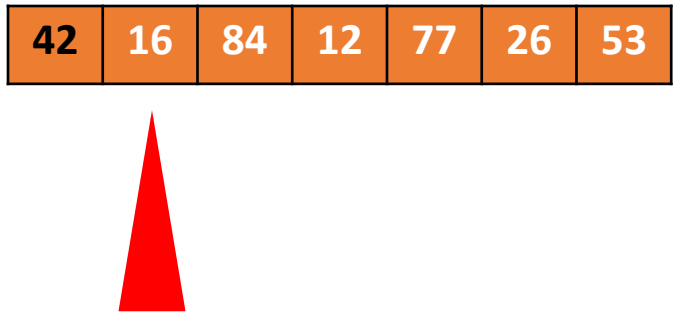
Selection Sort

```
void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // One by one move boundary of unsorted subarray
    for (i=0;i<n-1;i++)
    {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j=i+1;j<n;j++)
        if (arr[j] < arr[min_idx])
            min_idx = j;

// Swap the found minimum element with the first element
        swap(&arr[min_idx], &arr[i]);
    }
}
```

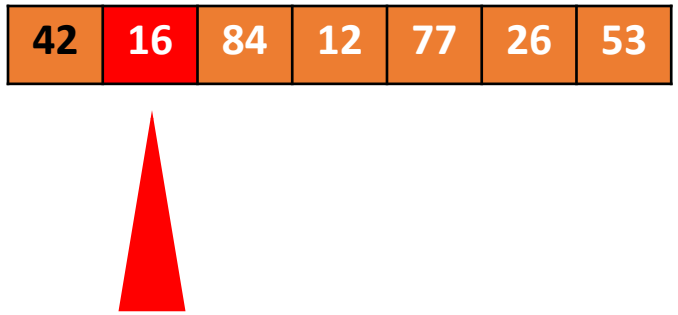
Selection Sort



The array, before the selection sort operation begins.

```
for (i = 0; i < n-1; i++)
{
    min_idx = i;
        for (j = i+1; j < n; j++)
           if (arr[j] < arr[min_idx])
               min_idx = j;
   swap(&arr[min_idx], &arr[i]);
}
```

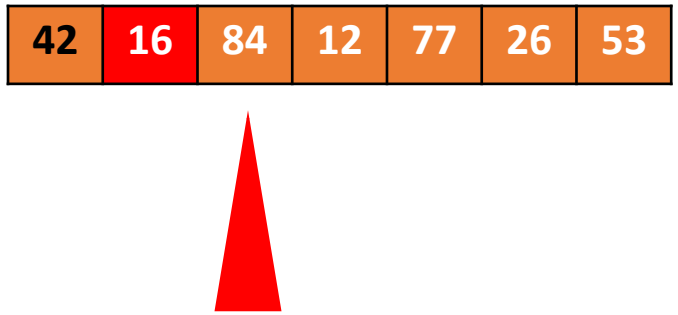
Selection Sort



The array, before the selection sort operation begins.

```
for (i = 0; i < n-1; i++)
{
    min_idx = i;
        for (j = i+1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;
   swap(&arr[min_idx], &arr[i]);
}
```

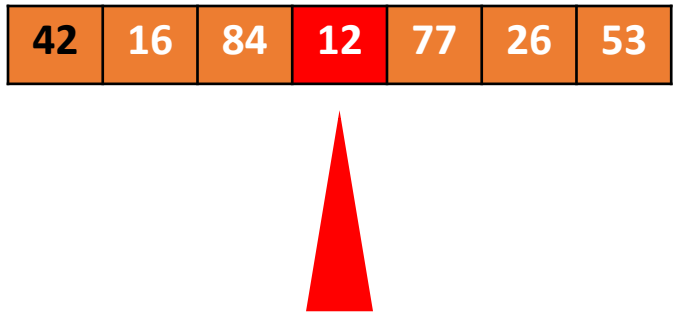
Selection Sort



The array, before the selection sort operation begins.

```
for (i = 0; i < n-1; i++)
{
    min_idx = i;
        for (j = i+1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;
   swap(&arr[min_idx], &arr[i]);
}
```

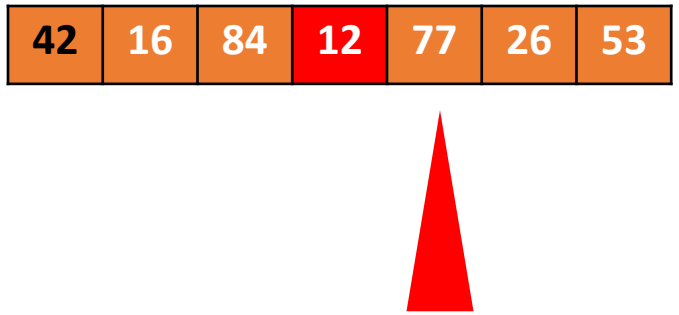
Selection Sort



The array, before the selection sort operation begins.

```
for (i = 0; i < n-1; i++)
{
    min_idx = i;
        for (j = i+1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;
   swap(&arr[min_idx], &arr[i]);
}
```

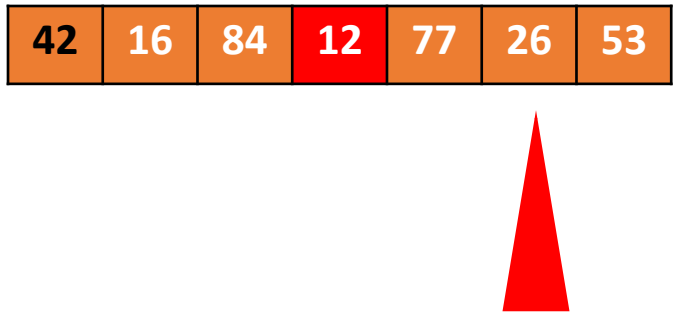
Selection Sort



The array, before the selection sort operation begins.

```
for (i = 0; i < n-1; i++)
{
    min_idx = i;
        for (j = i+1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;
   swap(&arr[min_idx], &arr[i]);
}
```

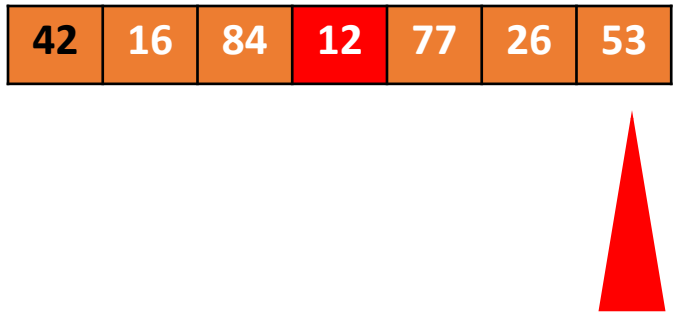
Selection Sort



The array, before the selection sort operation begins.

```
for (i = 0; i < n-1; i++)
{
    min_idx = i;
        for (j = i+1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;
   swap(&arr[min_idx], &arr[i]);
}
```

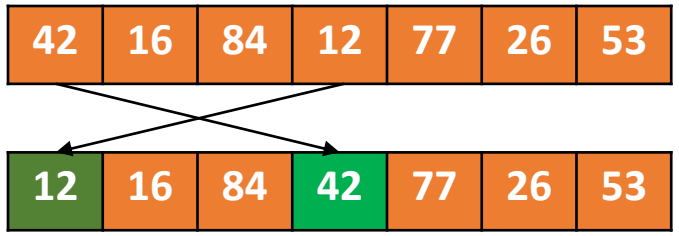
Selection Sort



The array, before the selection sort operation begins.

```
for (i = 0; i < n-1; i++)
{
    min_idx = i;
        for (j = i+1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;
   swap(&arr[min_idx], &arr[i]);
}
```

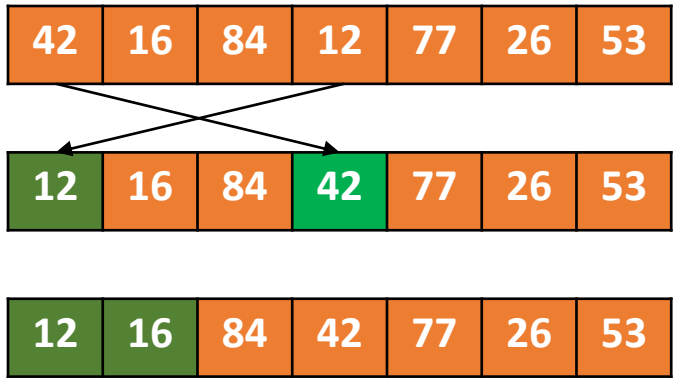
Selection Sort



The array, before the selection sort operation begins.

The smallest number (12) is swapped into the first element in the structure

Selection Sort

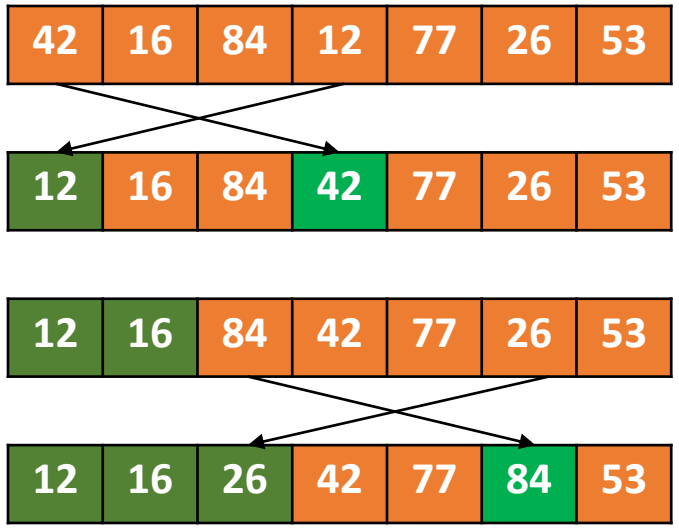


The array, before the selection sort operation begins.

The smallest number (12) is swapped into the first element in the structure

16 is the smallest; and does not need to be swapped.

Selection Sort



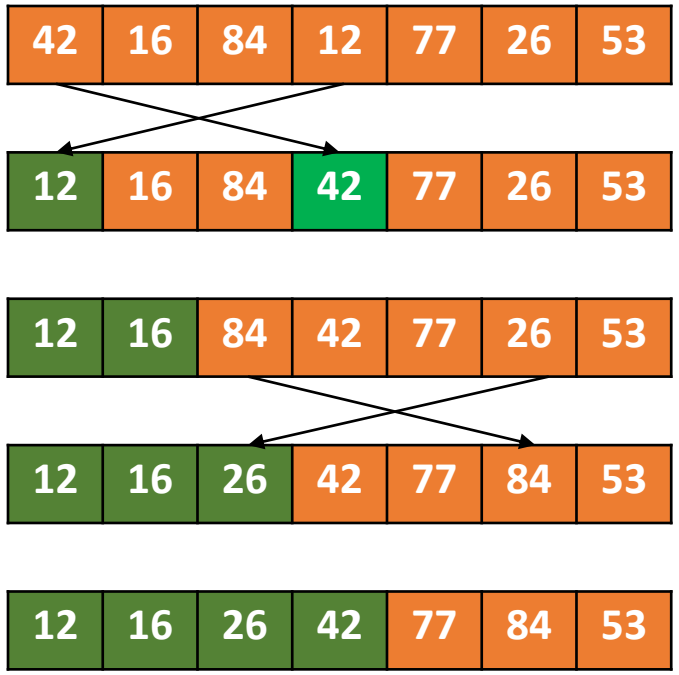
The array, before the selection sort operation begins.

The smallest number (12) is swapped into the first element in the structure

16 is the smallest; and does not need to be swapped.

26 is the next smallest number and is swapped into the third position.

Selection Sort



The array, before the selection sort operation begins.

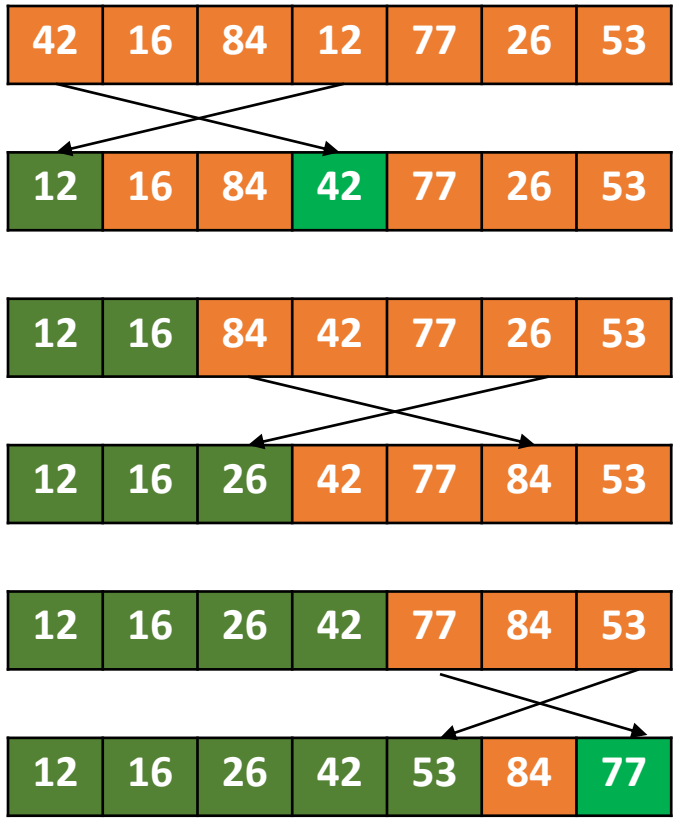
The smallest number (12) is swapped into the first element in the structure

16 is the smallest; and does not need to be swapped.

26 is the next smallest number and is swapped into the third position.

42 is the next smallest number and is swapped into the right position.

Selection Sort



The array, before the selection sort operation begins.

The smallest number (12) is swapped into the first element in the structure

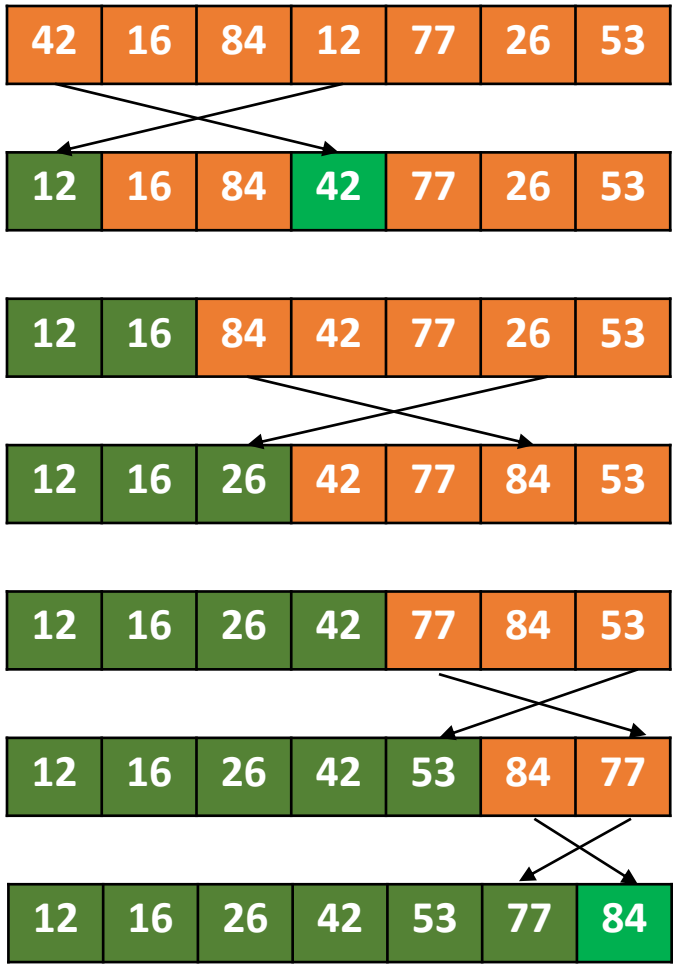
16 is the smallest; and does not need to be swapped.

26 is the next smallest number and is swapped into the third position.

42 is the next smallest number and is swapped into the right position.

53 is the next smallest number and is swapped into the appropriate position.

Selection Sort



The array, before the selection sort operation begins.

The smallest number (12) is swapped into the first element in the structure

16 is the smallest; and does not need to be swapped.

26 is the next smallest number and is swapped into the third position.

42 is the next smallest number and is swapped into the right position.

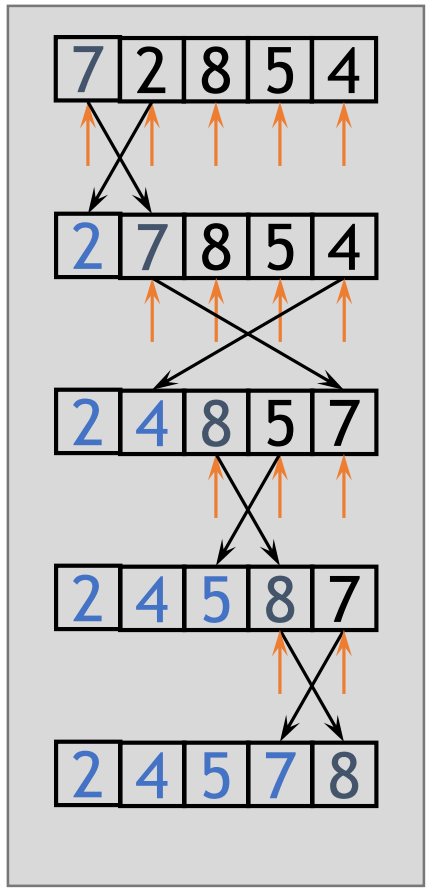
53 is the next smallest number and is swapped into the appropriate position.

77 is swapped. The selection sort is now complete

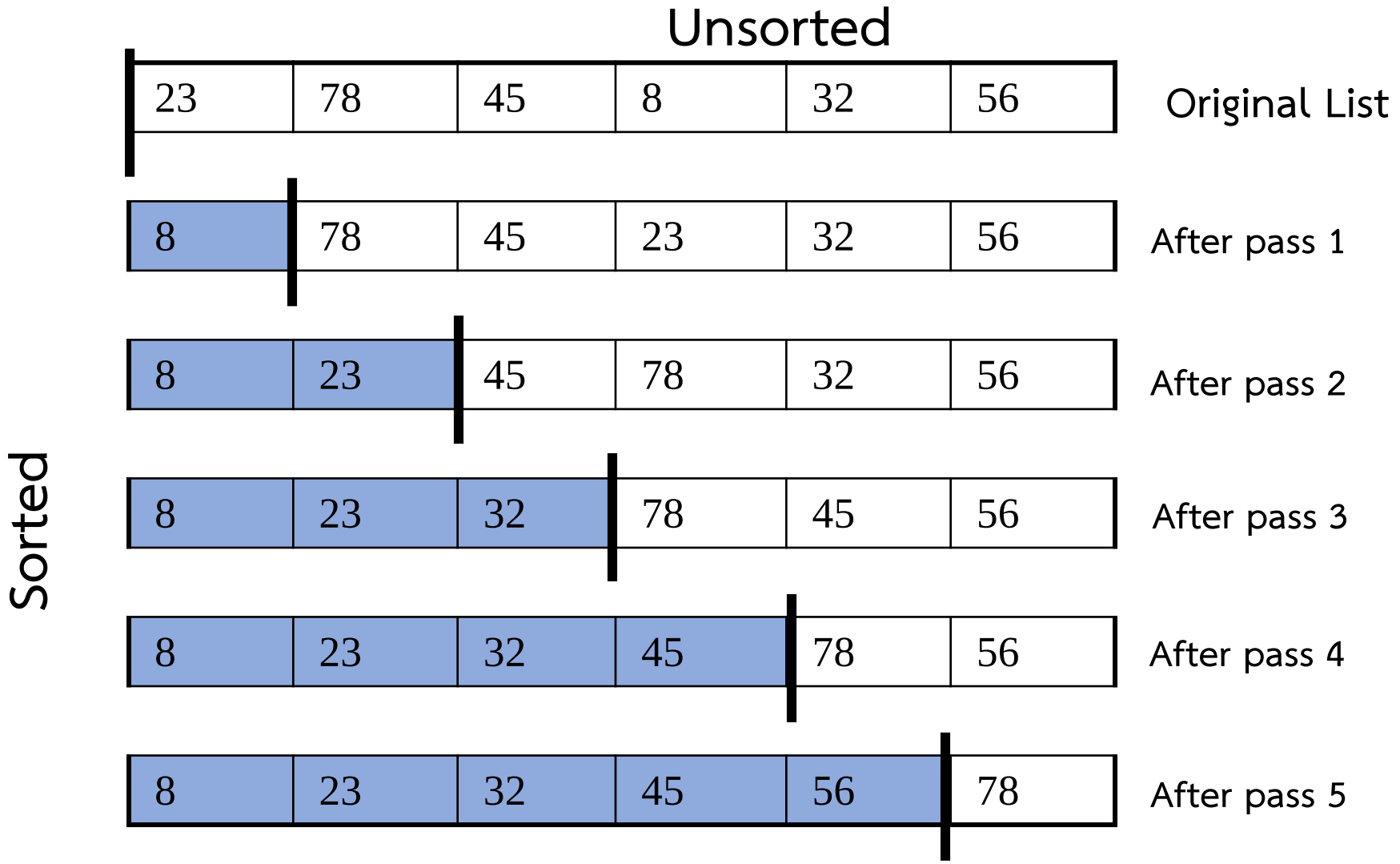
ประสิทธิภาพ

- ความเร็ว Big O = ?
- Linked หรือว่า Array ต่างกันไหม
- Singly Linked List หรือว่า Doubly Linked List ล่ะ
- เร็วหรือว่าช้ากว่า Bubble

วิเคราะห์ selection sort



- The selection sort might swap an array element with itself--this is harmless, and not worth checking for
- Analysis:
  - The outer loop executes n-1 times
  - The inner loop executes about $n/2$ times on average (from n to 2 times)
  - Work done in the inner loop is constant (swap two array elements)
  - Time required is roughly $(n\text{-}1)*(n/2)$
  - You should recognize this as $O(n^2)$


Unsorted
23 78 45 8 32 56
Original List
8 78 45 23 32 56
After pass 1
8 23 45 78 32 56
After pass 2
8 23 32 78 45 56
After pass 3
8 23 32 45 78 56
After pass 4
8 23 32 45 56 78
After pass 5
Sorted

Insertion Sort

Insertion Sort

การจัดเรียงแบบแทรก (Insertion Sort)

- เป็นการจัดเรียงโดยการนำข้อมูลที่จะทำการเรียงนั้น ๆ ไปจัดเรียงทีละตัว
- โดยการแทรกตัวที่จะเรียงไว้ในตำแหน่งที่เหมาะสมของข้อมูลที่มีการจัดเรียงเรียบร้อยแล้ว ณ ตำแหน่งที่ถูกต้อง
- วิธีการลักษณะนี้จะคล้ายกับการหยิบไพ่ขึ้นมาเรียงทีละใบ ซึ่ง
  - ไพ่ใบแรกจะไม่ต้องสนใจอะไร
  - แต่เมื่อหยิบไพ่ใบที่ 2 ก็จะต้องพิจารณาว่าจะไว้ก่อนหรือไว้หลังใบแรก
  - และเช่นเดียวกัน เมื่อหยิบไพ่ใบถัด ๆ มา ก็จะต้องพิจารณาว่าจะวางใบตำแหน่งใดเพื่อให้เกิดการเรียงลำดับ จนกระทั่งหมด

Insertion Sort

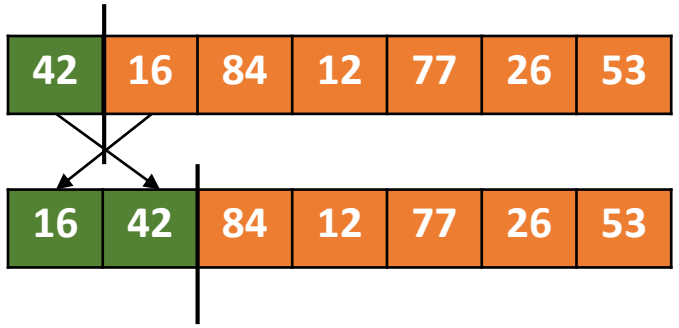
```
void insertionSort(int arr[], int n)
{
     int i,j,key;
     for (i=1;i<n;i++)
      {
        key = arr[i];
        for (j=1;j>0&&arr[j-1]>key;j--)
        }
              arr[j+1] = arr[j];
        }
        arr[j+1] = key;
  }
}
```

```
if(arr[j-1]<=key)
    for(j=i;j>0;j--)
```

Insertion Sort

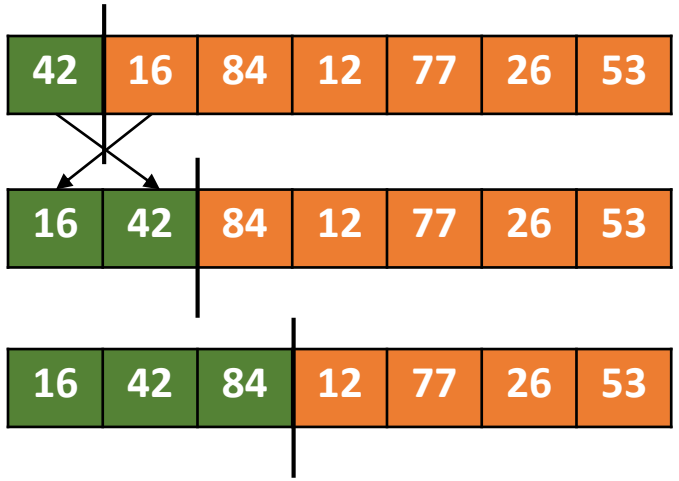

```
for (i = 1; i < n; i++){
   key = arr[i];
   for (j = 1 ; j > 0 && arr[j-1] >key;j--)
   {
      arr[j + 1] = arr[j];
   }
   arr[j + 1] = key;
}
```

Insertion Sort



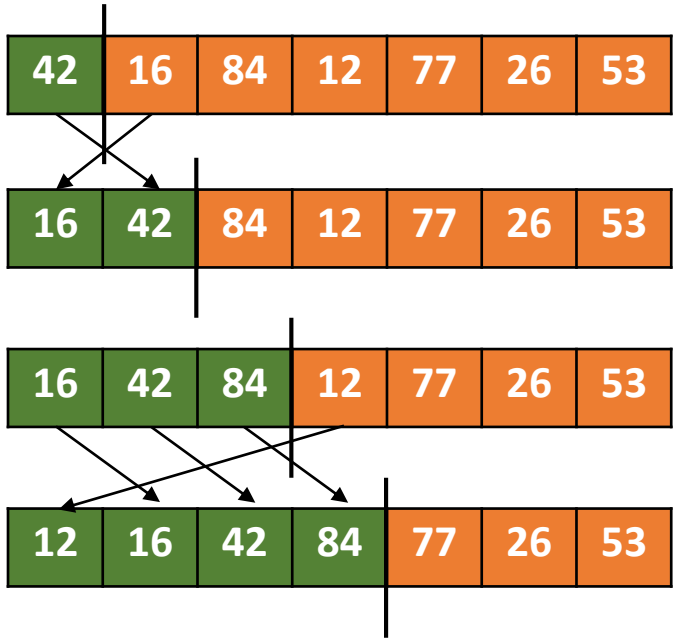

```
for (i = 1; i < n; i++){
   key = arr[i];
   for (j = 1 ; j > 0 && arr[j-1] >key;j--)
   {
      arr[j + 1] = arr[j];
   }
   arr[j + 1] = key;
}
```

Insertion Sort



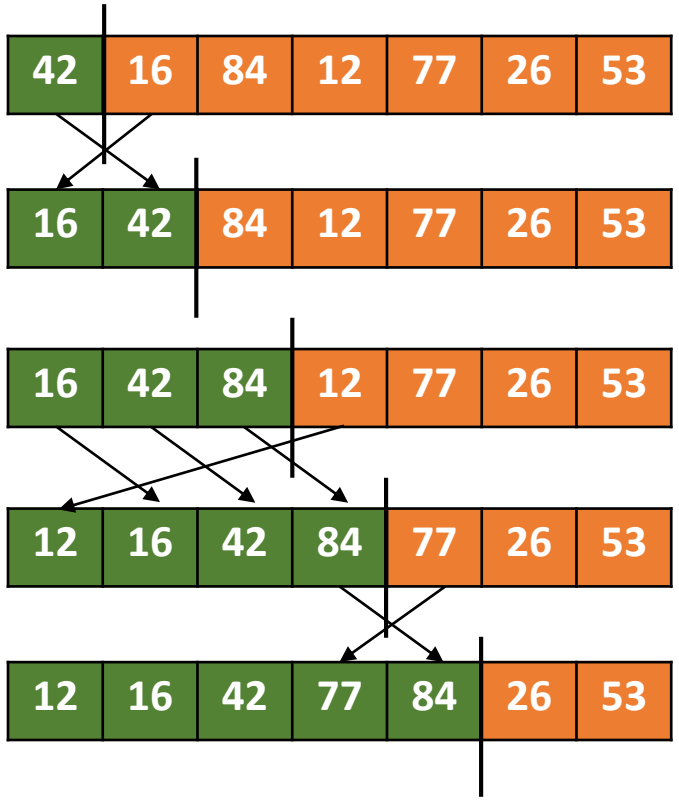

```
for (i = 1; i < n; i++){
   key = arr[i];
   for (j = 1 ; j > 0 && arr[j-1] >key;j--)
   {
      arr[j + 1] = arr[j];
   }
   arr[j + 1] = key;
}
```

Insertion Sort



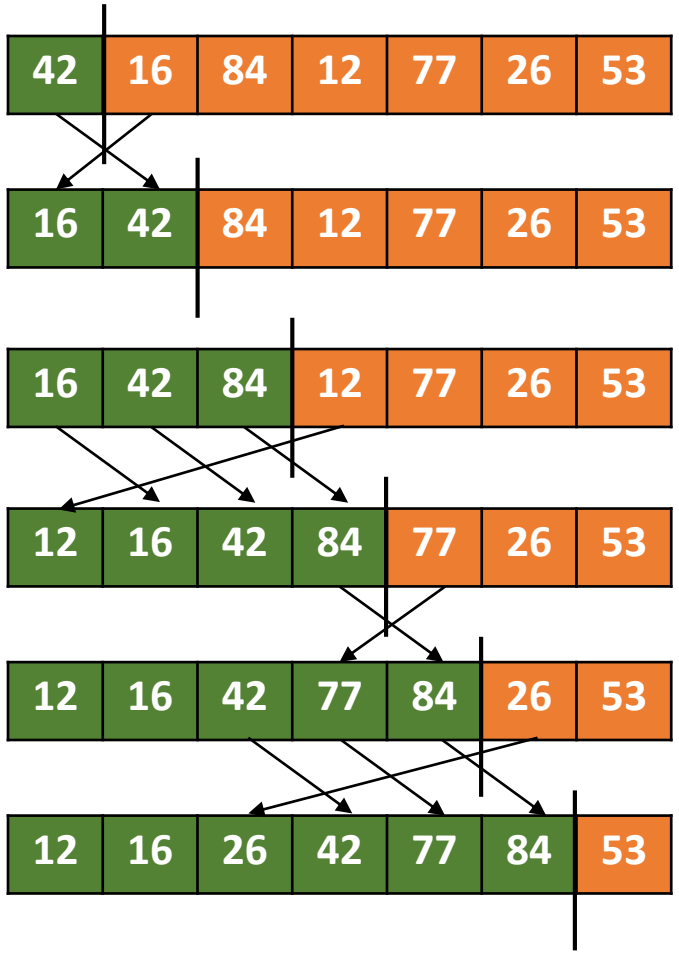

```
for (i = 1; i < n; i++){
    key = arr[i];
    for (j = 1 ; j > 0 && arr[j-1] >key;j--)
    {
        arr[j + 1] = arr[j];
    }
    arr[j + 1] = key;
}
```

Insertion Sort



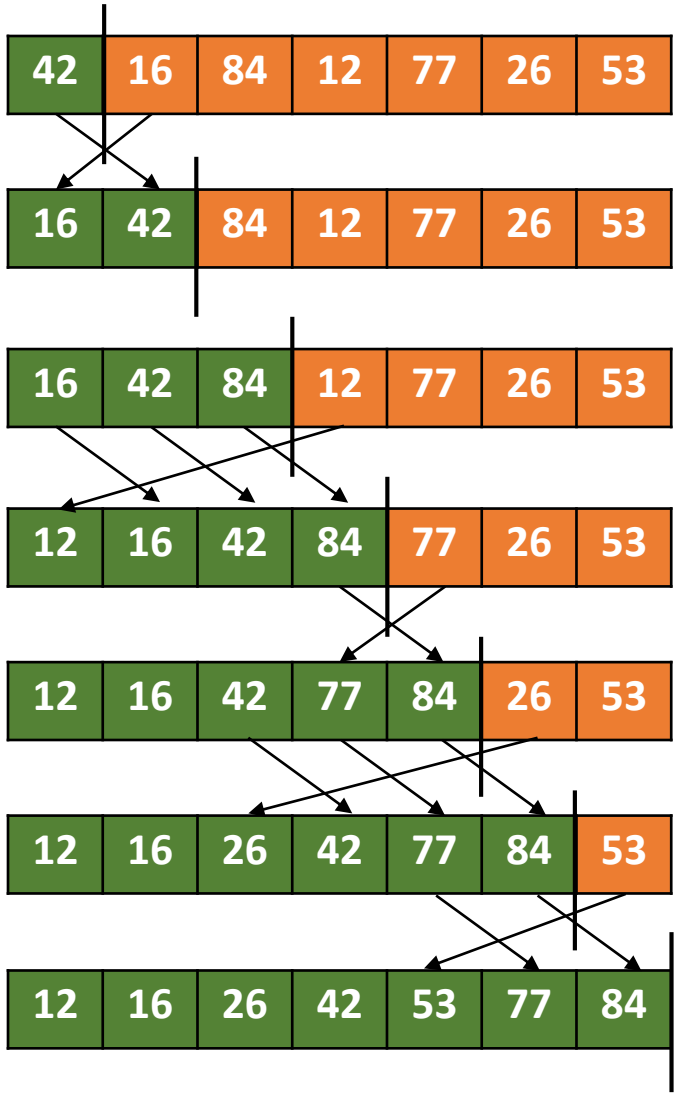

```
for (i = 1; i < n; i++){
    key = arr[i];
    for (j = 1 ; j > 0 && arr[j-1] >key;j--)
    {
        arr[j + 1] = arr[j];
    }
    arr[j + 1] = key;
}
```

Insertion Sort




```
for (i = 1; i < n; i++){
    key = arr[i];
    for (j = 1 ; j > 0 && arr[j-1] >key;j--)
    {
        arr[j + 1] = arr[j];
    }
    arr[j + 1] = key;
}
```

Insertion Sort




```
for (i = 1; i < n; i++){
    key = arr[i];
    for (j = 1 ; j > 0 && arr[j-1] >key;j--)
    {
        arr[j + 1] = arr[j];
    }
    arr[j + 1] = key;
}
```

ประสิทธิภาพ

- ความเร็ว Big O = ?
- Linked หรือว่า Array ต่างกันไหม
- Singly Linked List หรือว่า Doubly Linked List ล่ะ
- เทียบกันกับ Selection Sort
- สอง Array เร็วกว่าไหม
- เหมาะกับข้อมูลแบบไหน

วิเคราะห์ insertion sort

- We run once through the outer loop, inserting each of n elements; this is a factor of n
- On average, there are $n/2$ elements already sorted
  - The inner loop looks at (and moves) half of these
  - This gives a second factor of $n/4$
- Hence, the time required for an insertion sort of an array of n elements is proportional to $n^2/4$
- Discarding constants, we find that insertion sort is $O(n^2)$

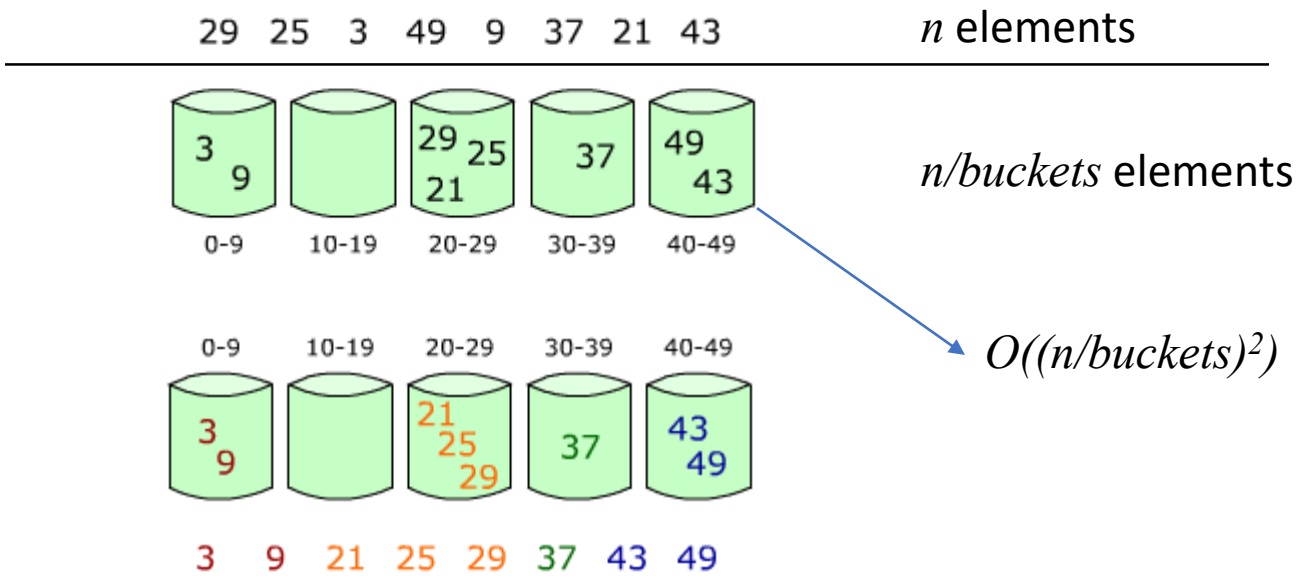
เปรียบเทียบ N, $logN$ กับ N^2

N	$O(LogN)$	$O(N^2)$
16	4	256
64	6	4K
256	8	64K
1,024	10	1M
16,384	14	256M
131,072	17	16G
262,144	18	6.87E+10
524,288	19	2.74E+11
1,048,576	20	1.09E+12
1,073,741,824	30	1.15E+18

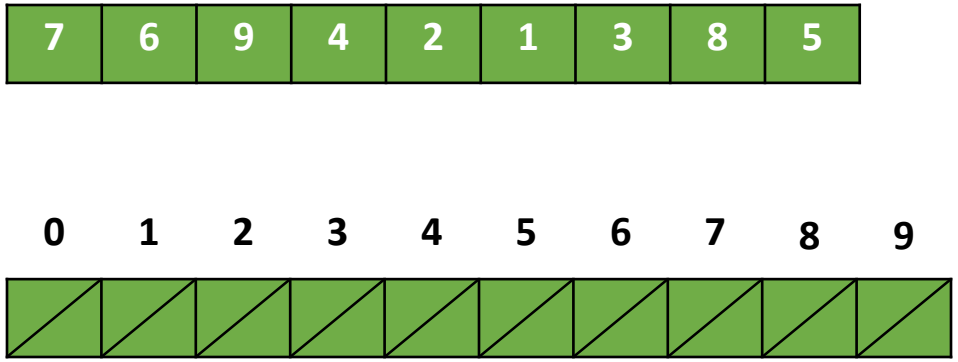
เปรียบเทียบ Sorting Algorithms

Algorithm	Exact time	Complexity
Bubble Sort	$n * (n/2) * k$	n^2
Selection Sort	$(n-1) * (n/2)$	n^2
Insertion Sort	$n^2/4$	n^2

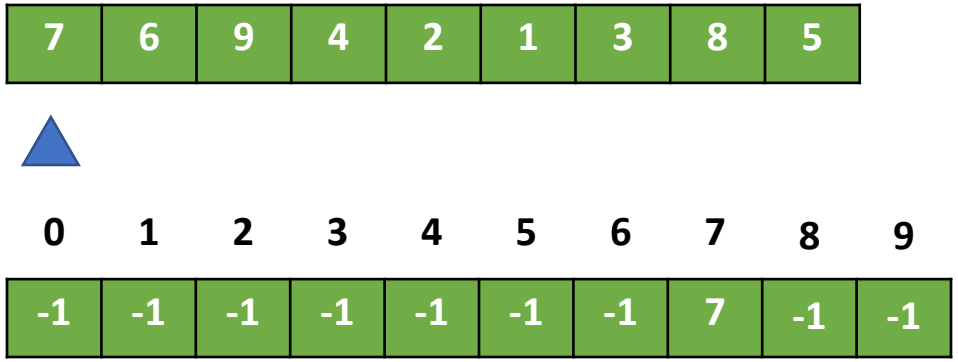
แลกหน่วยความจำกับเวลาประมวลผล



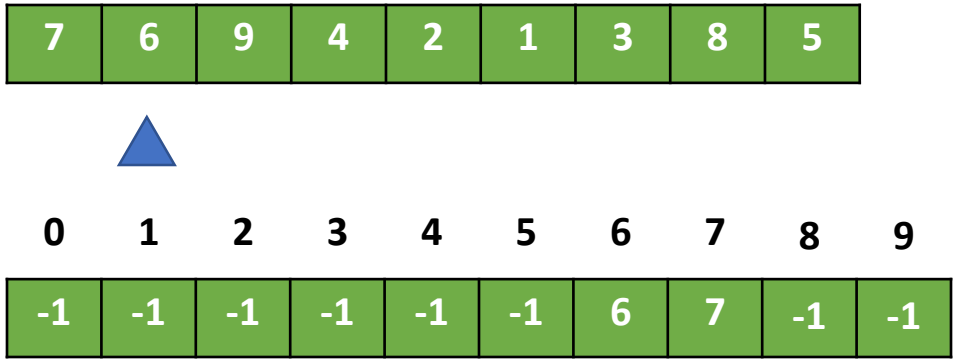
Bucket Sort



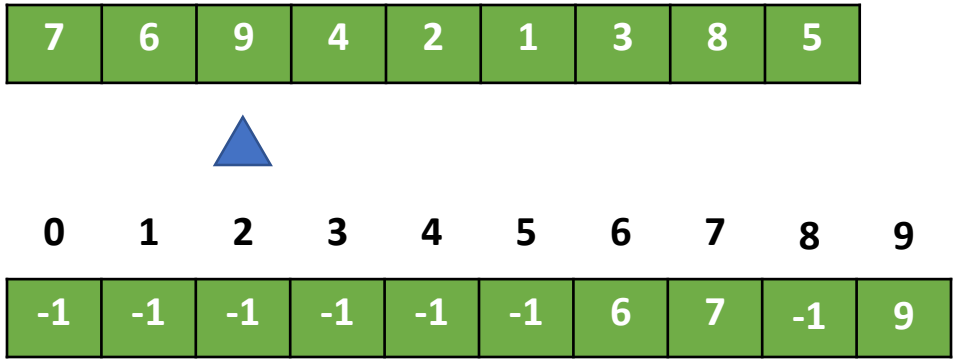
Bucket Sort



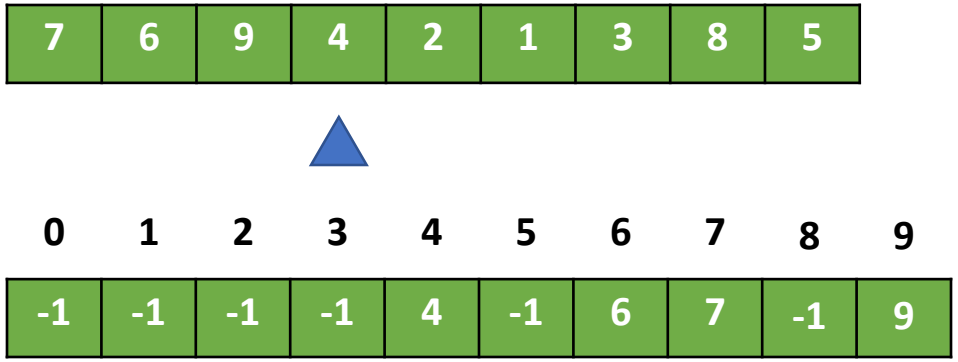
Bucket Sort



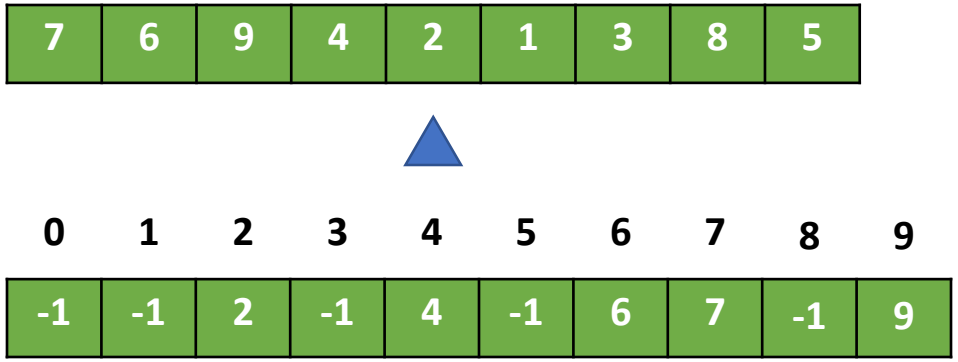
Bucket Sort



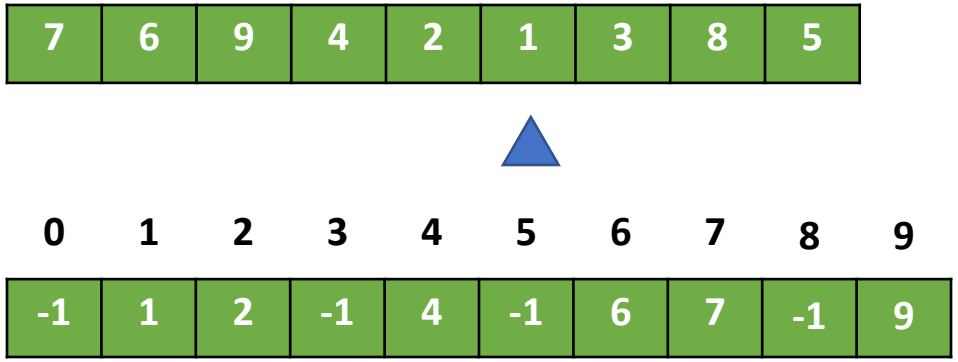
Bucket Sort



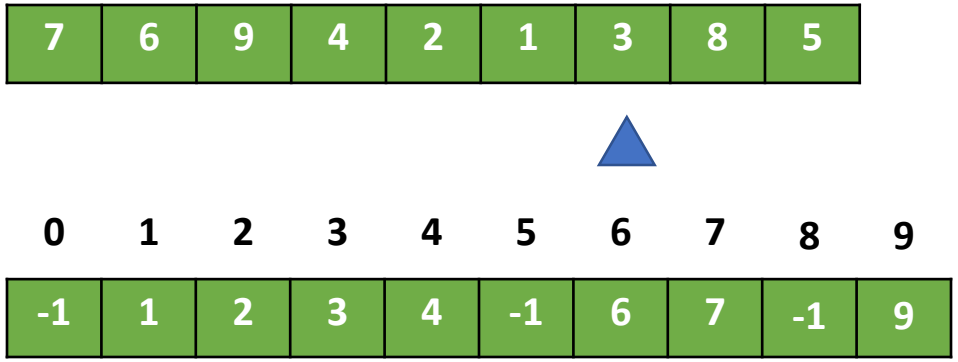
Bucket Sort



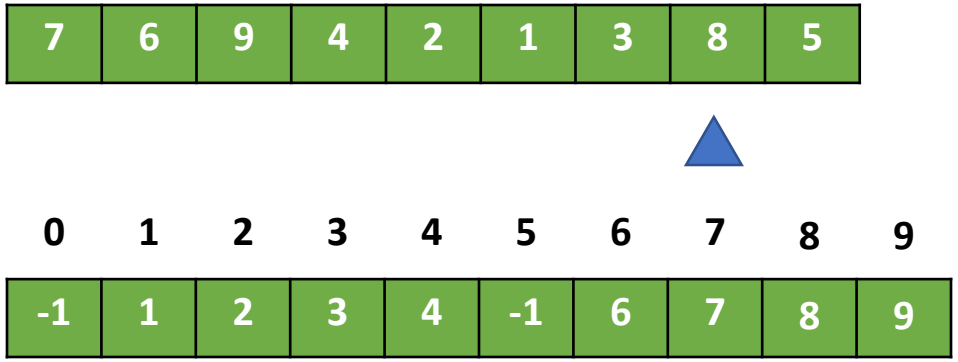
Bucket Sort



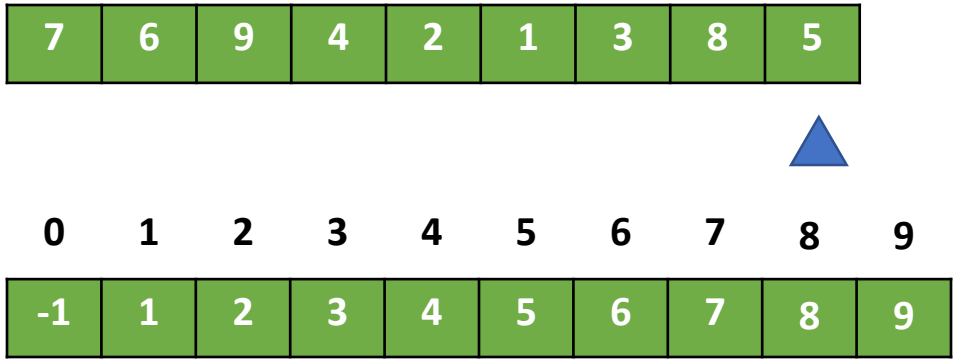
Bucket Sort



Bucket Sort



Bucket Sort



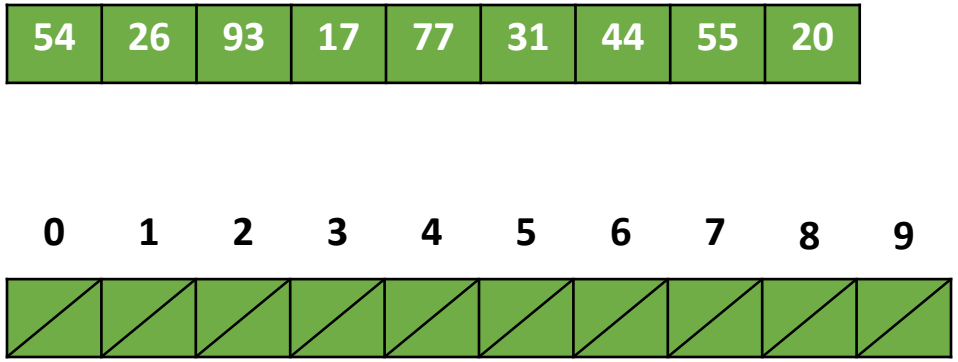
Bucket Sort + Insertion Sort

- Sorting on the go
  - โยนใส่ Bucket ปุ๊บ Sort เลย

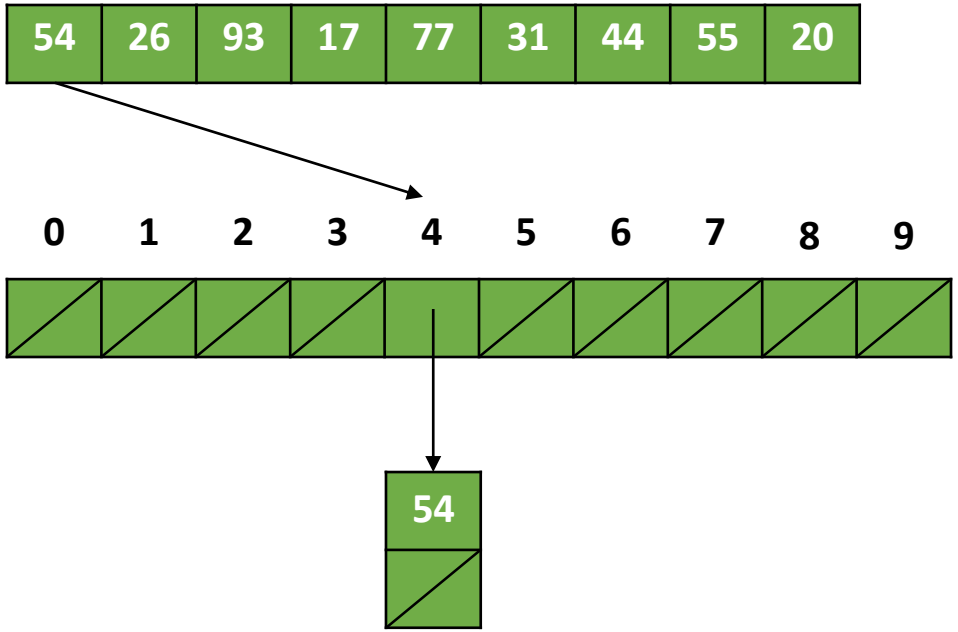
Bucket Sort + Recursive (Radix)

- Sort ทีละหลัก
  - โยนใส่ Bucket แล้วโยนอีกที

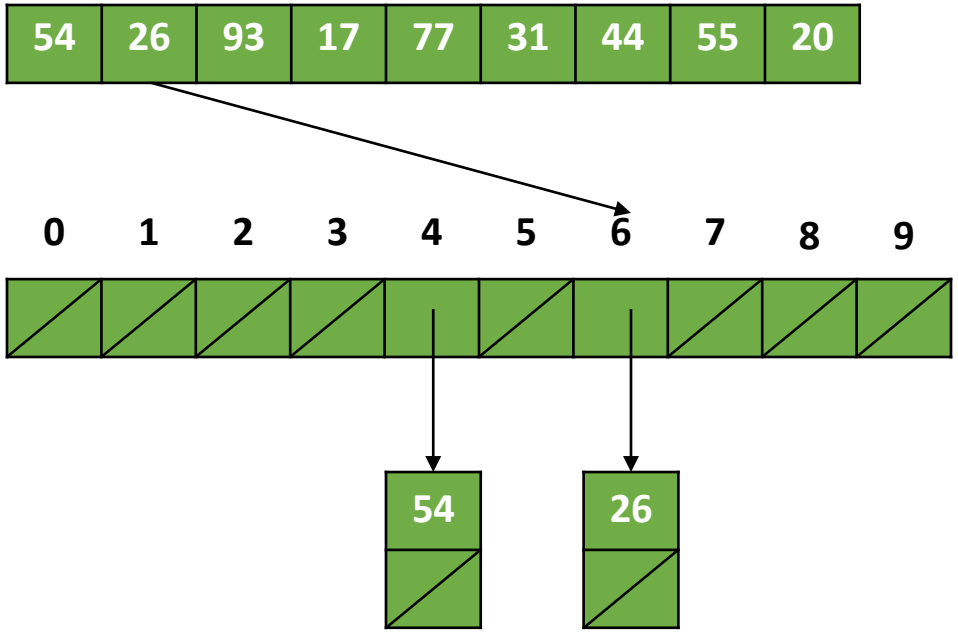
Radix Sort - LSD



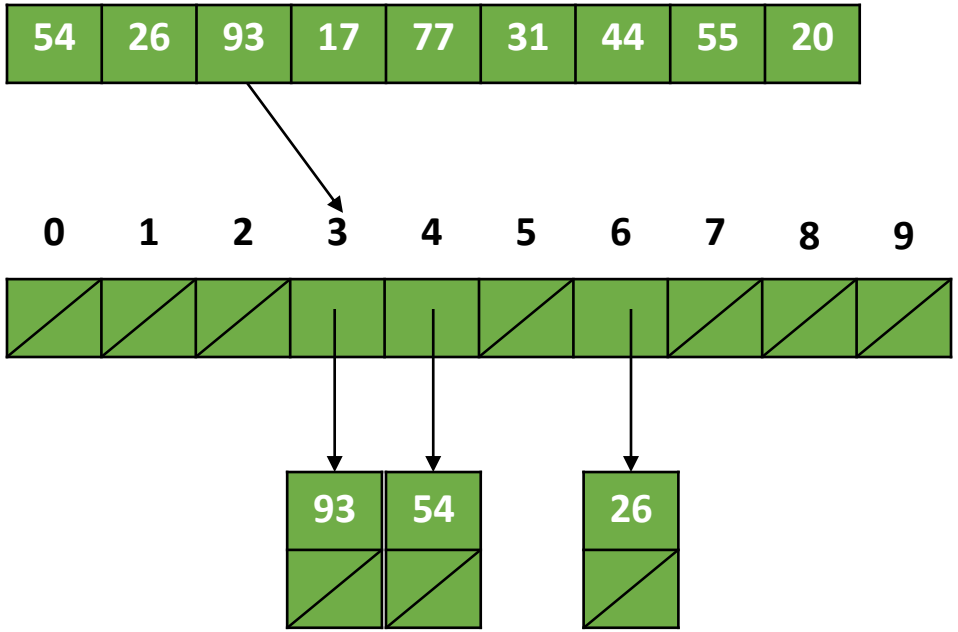
Radix Sort - LSD



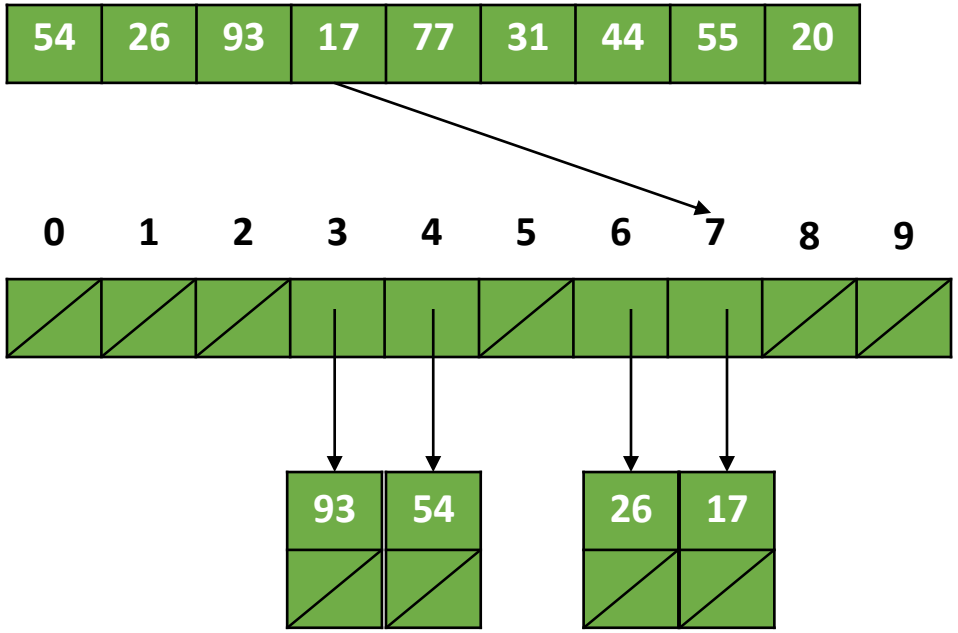
Radix Sort - LSD



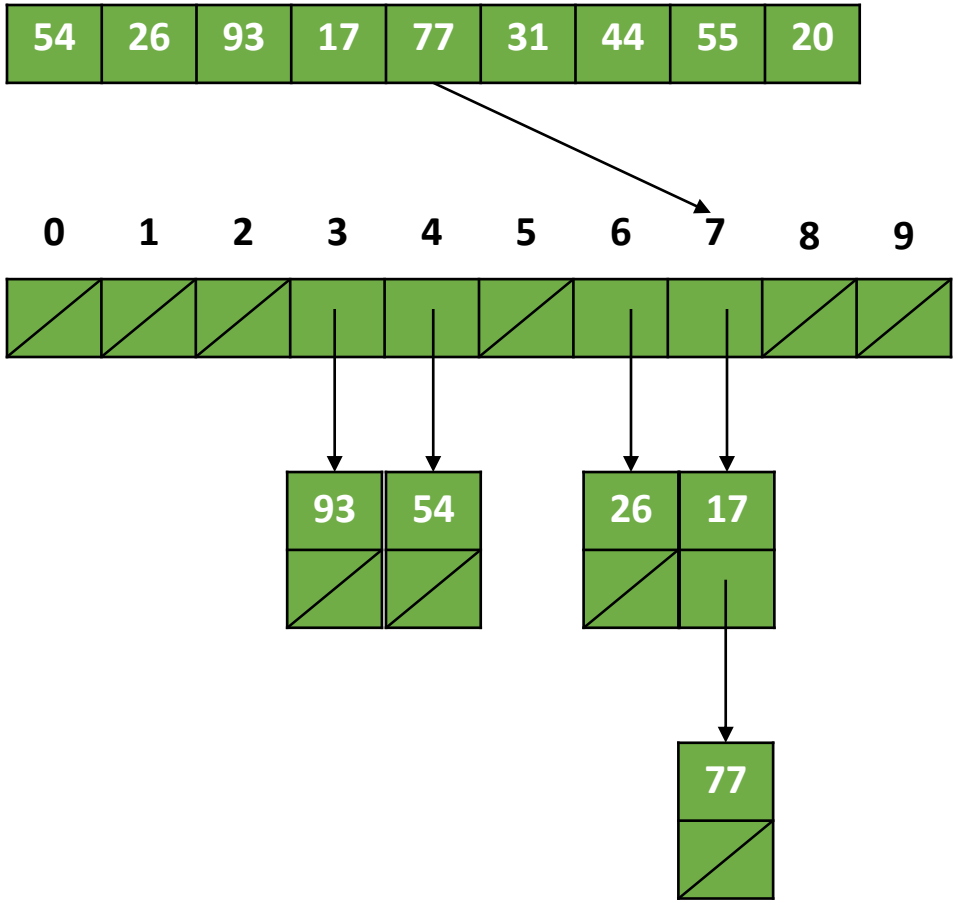
Radix Sort - LSD



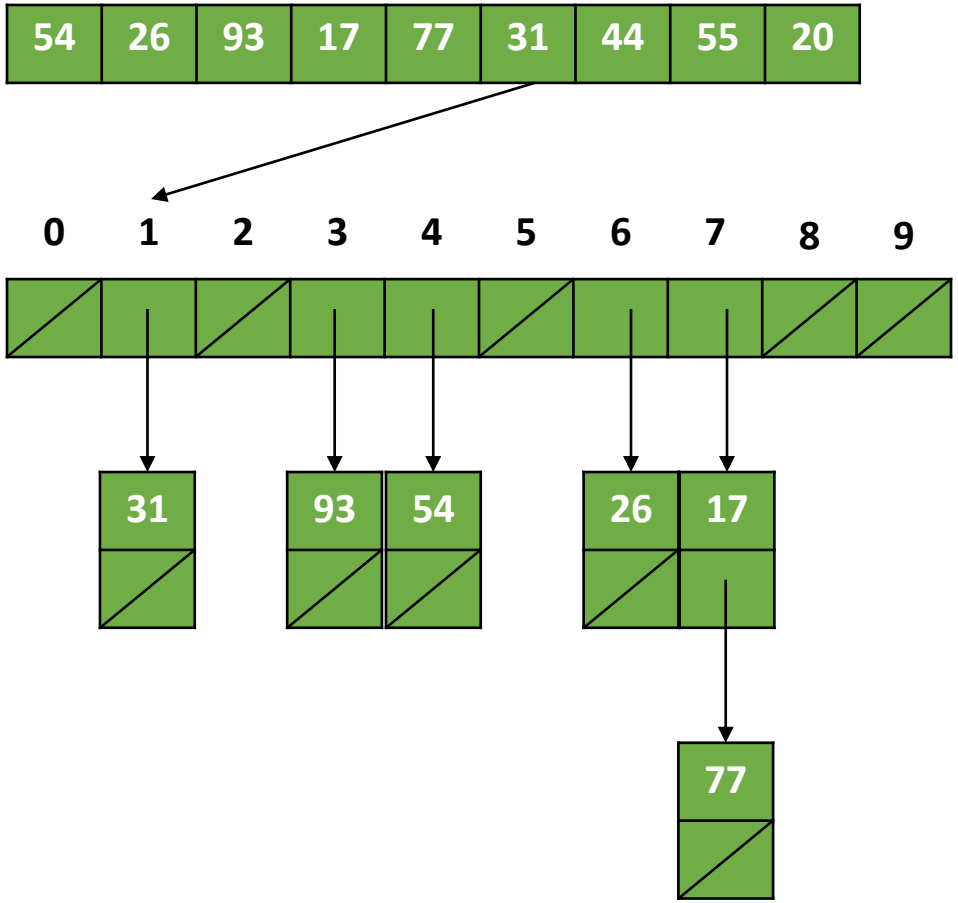
Radix Sort - LSD



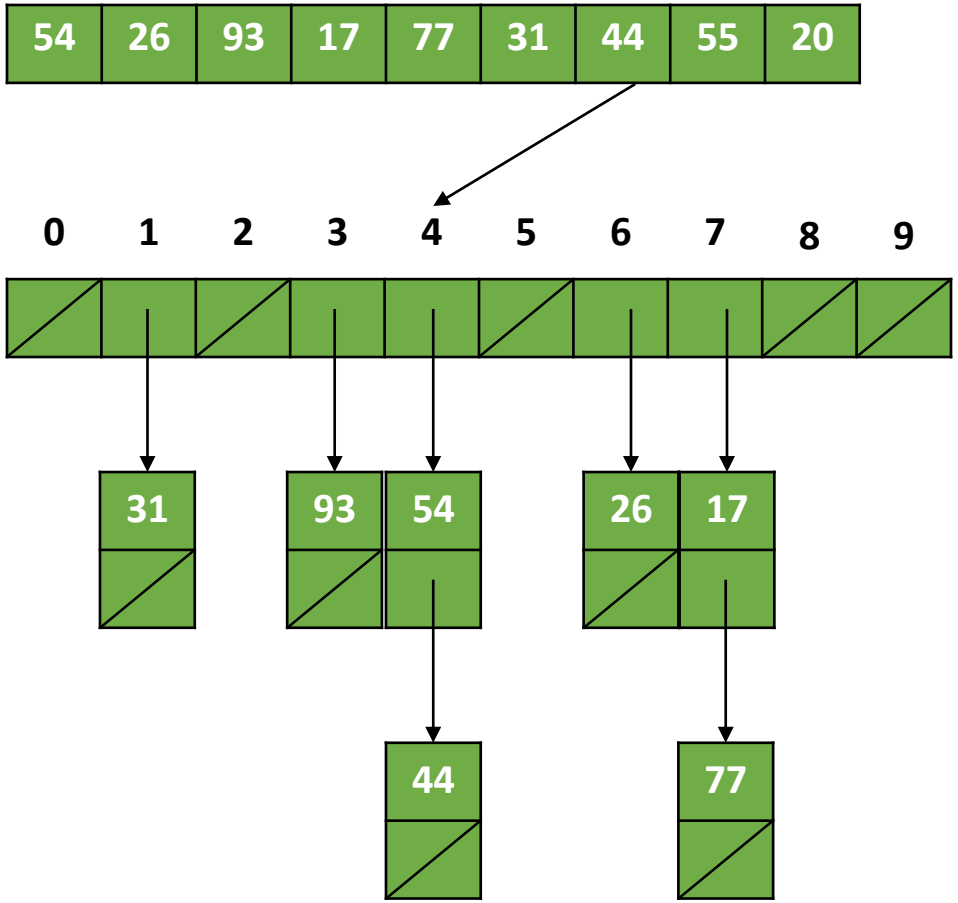
Radix Sort - LSD



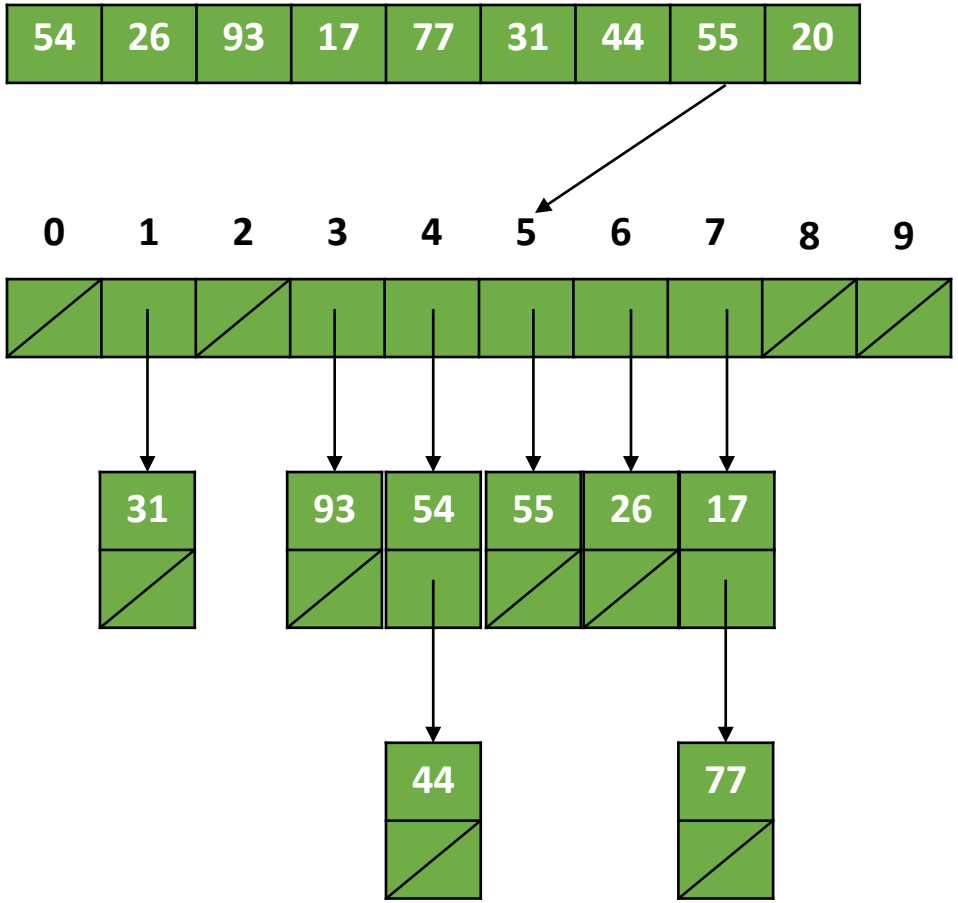
Radix Sort - LSD



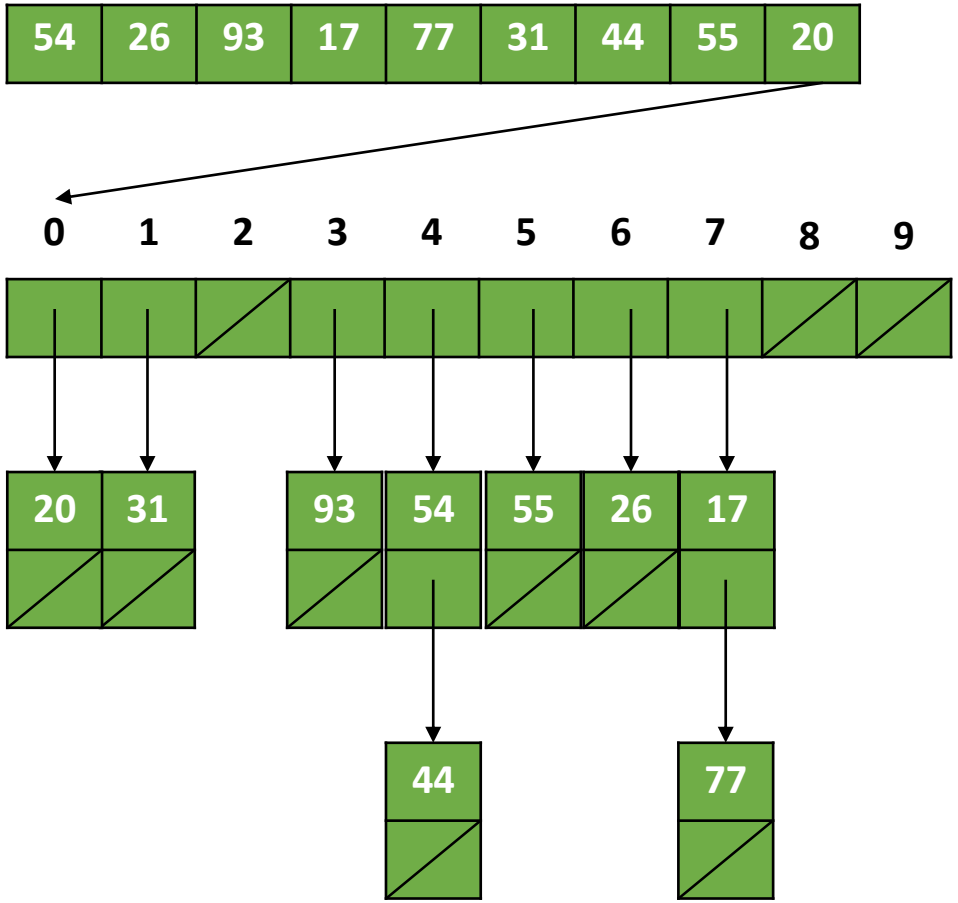
Radix Sort - LSD



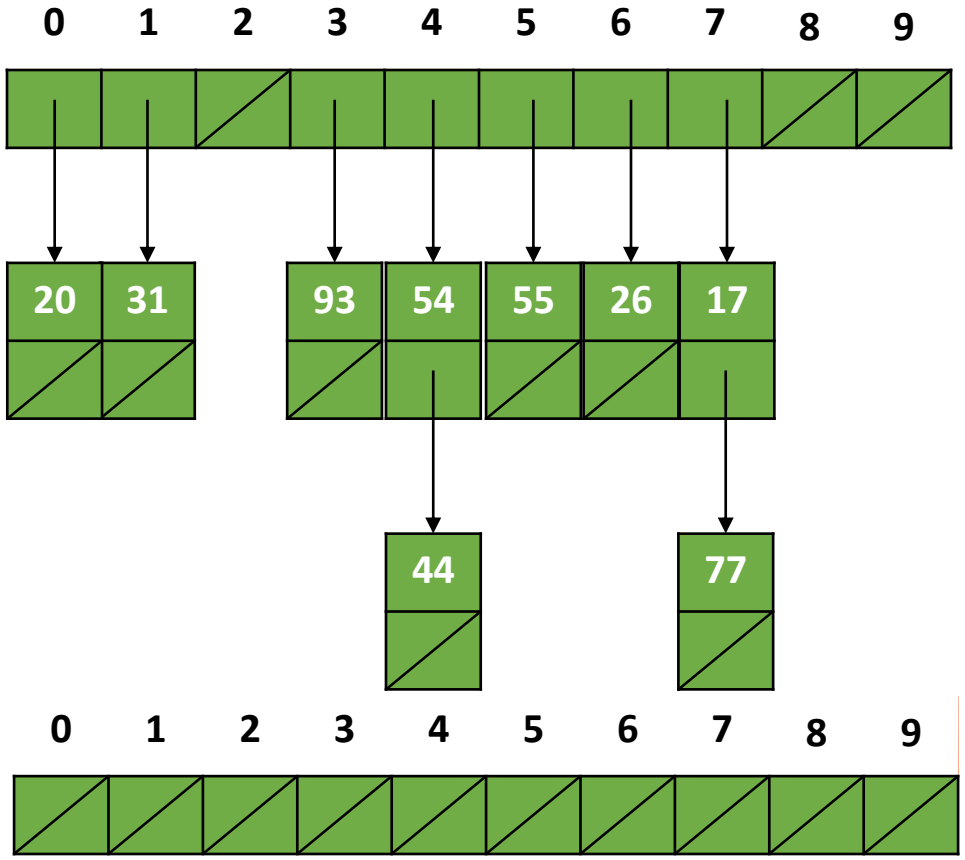
Radix Sort - LSD



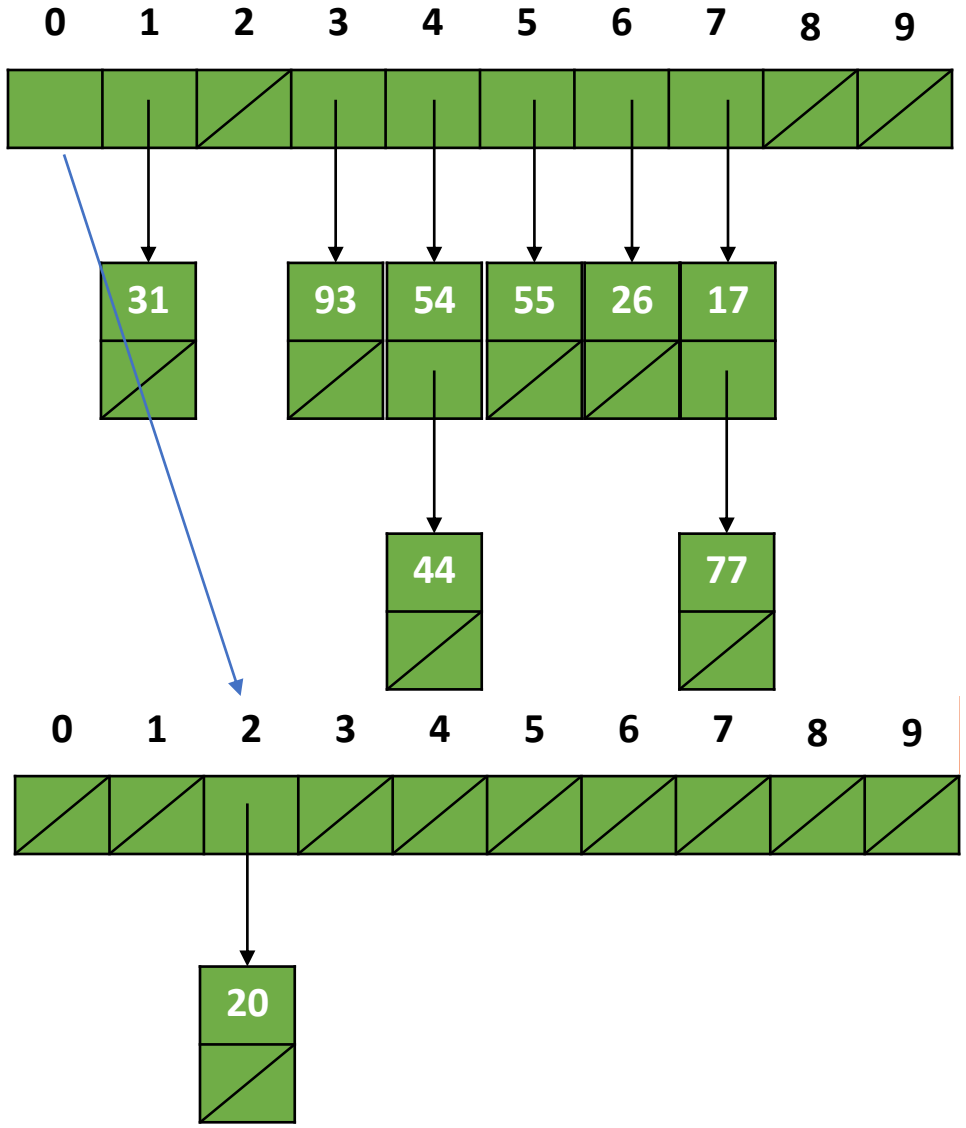
Radix Sort - LSD



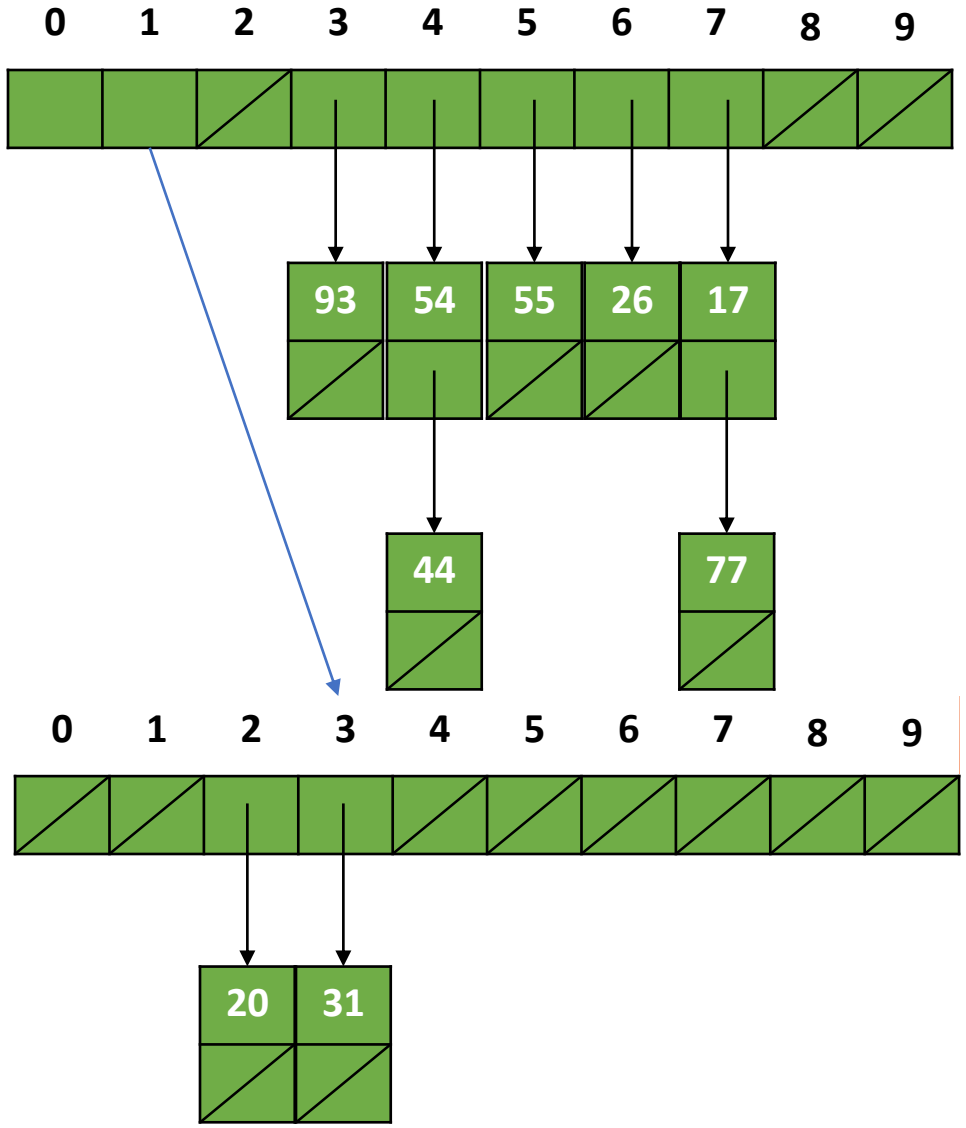
Radix Sort - LSD



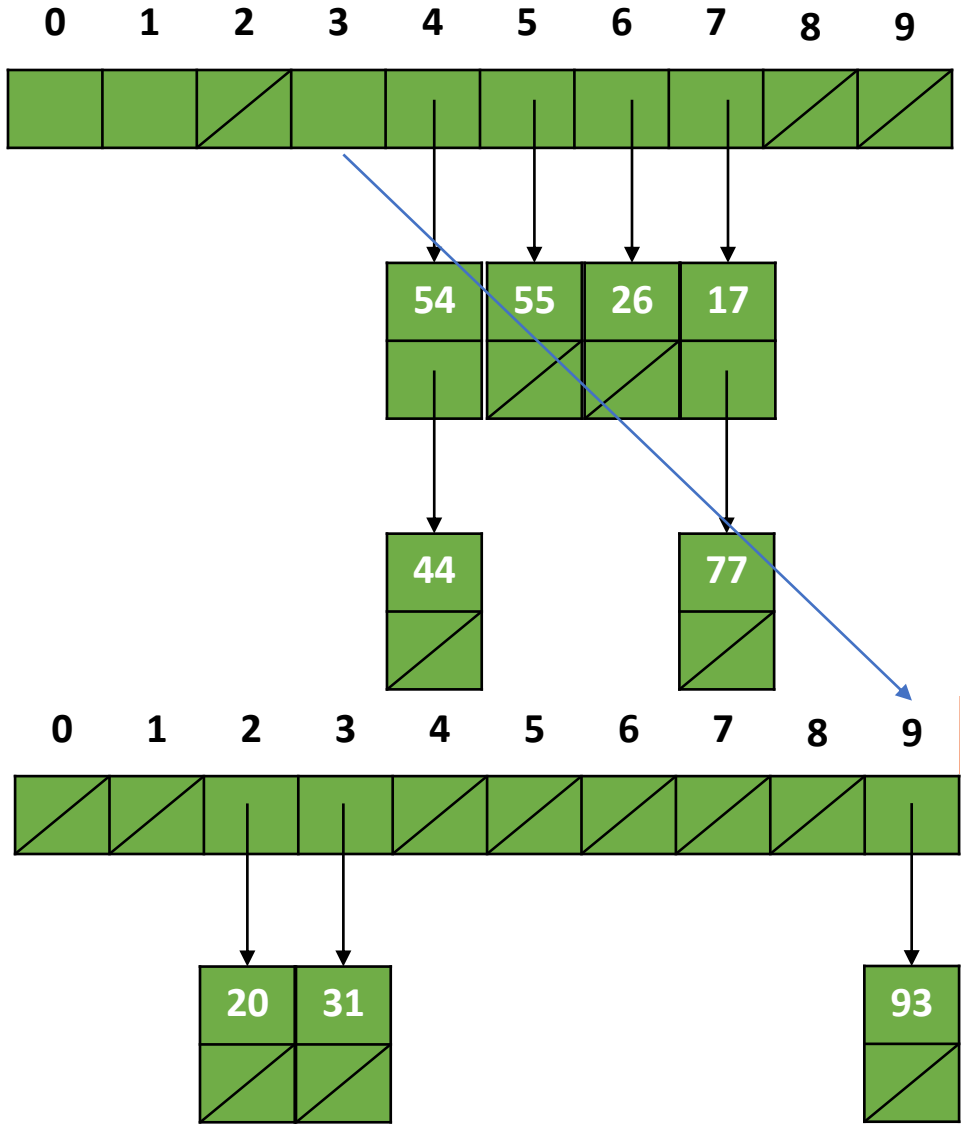
Radix Sort - LSD



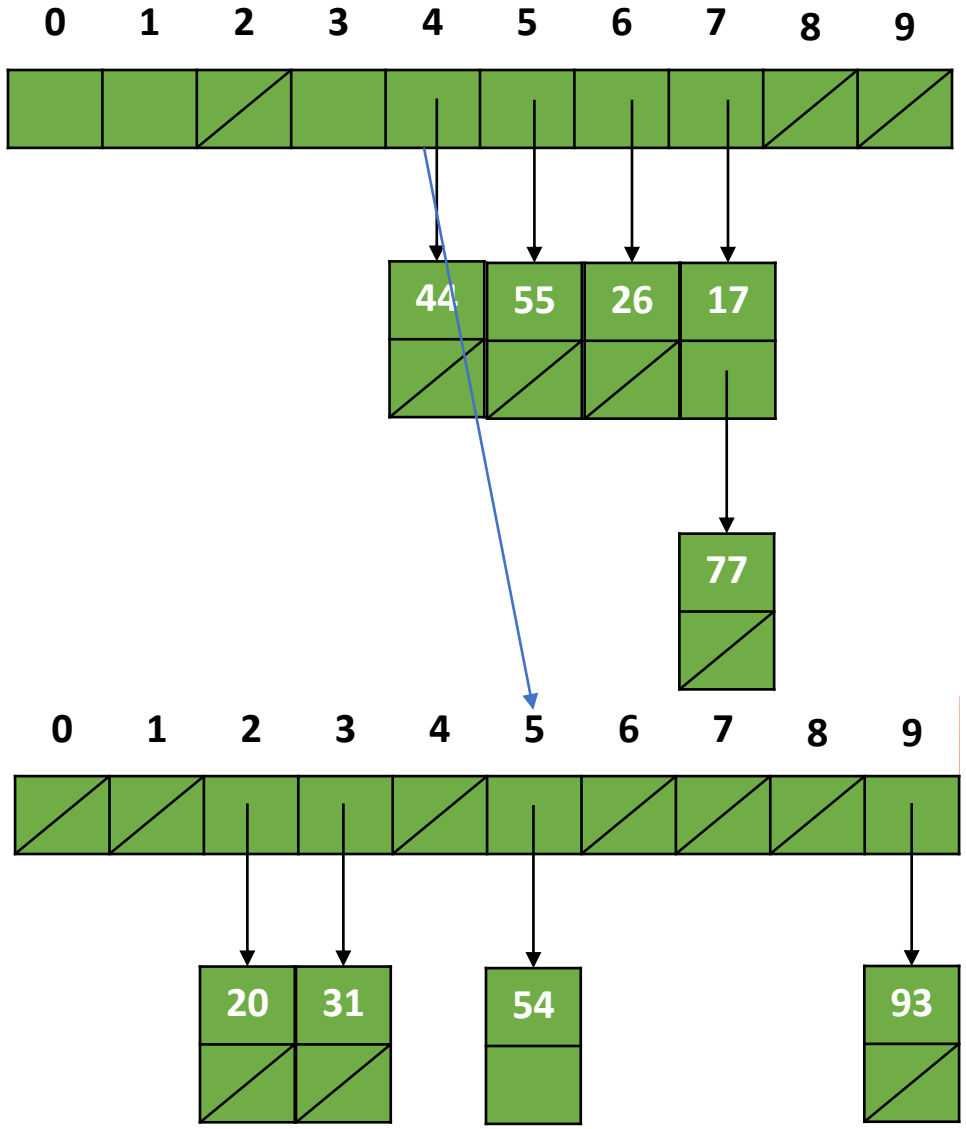
Radix Sort - LSD



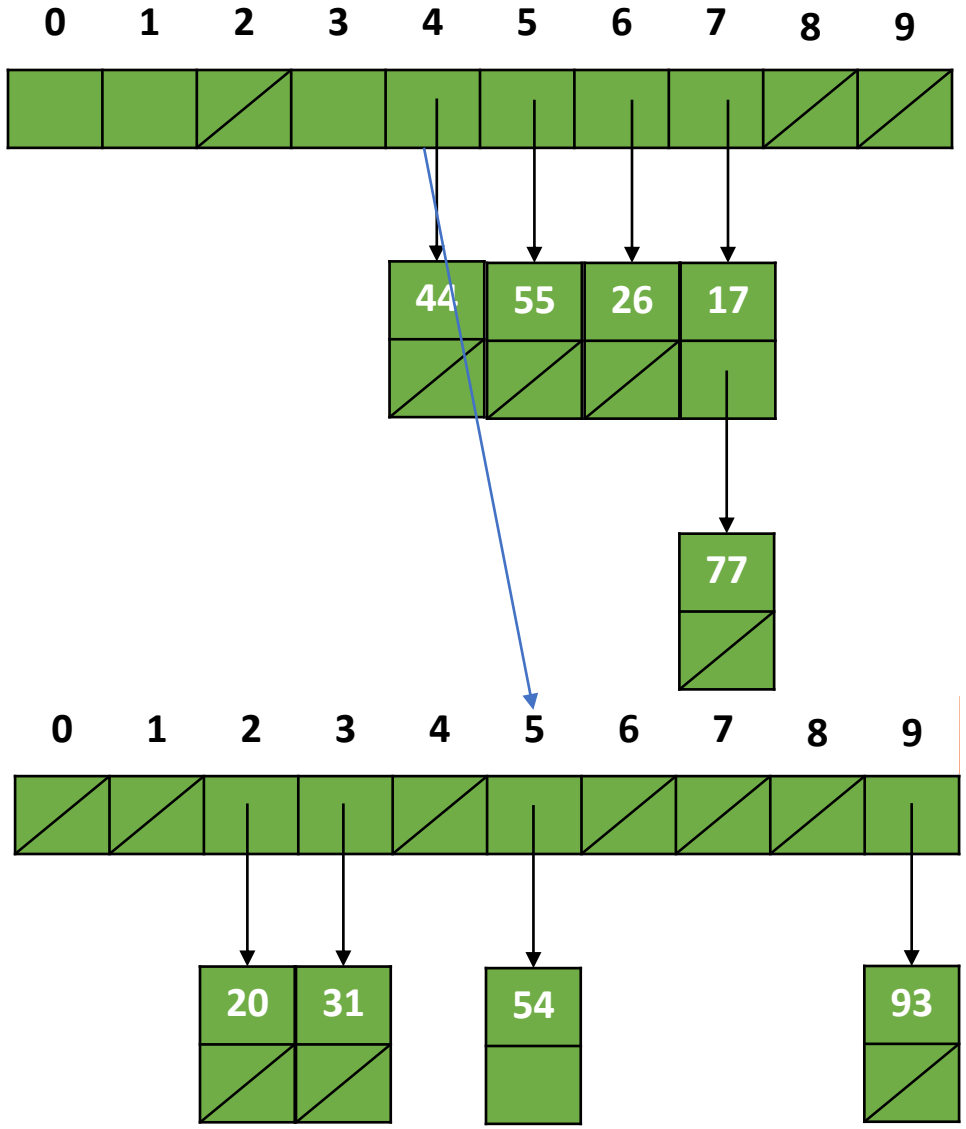
Radix Sort - LSD



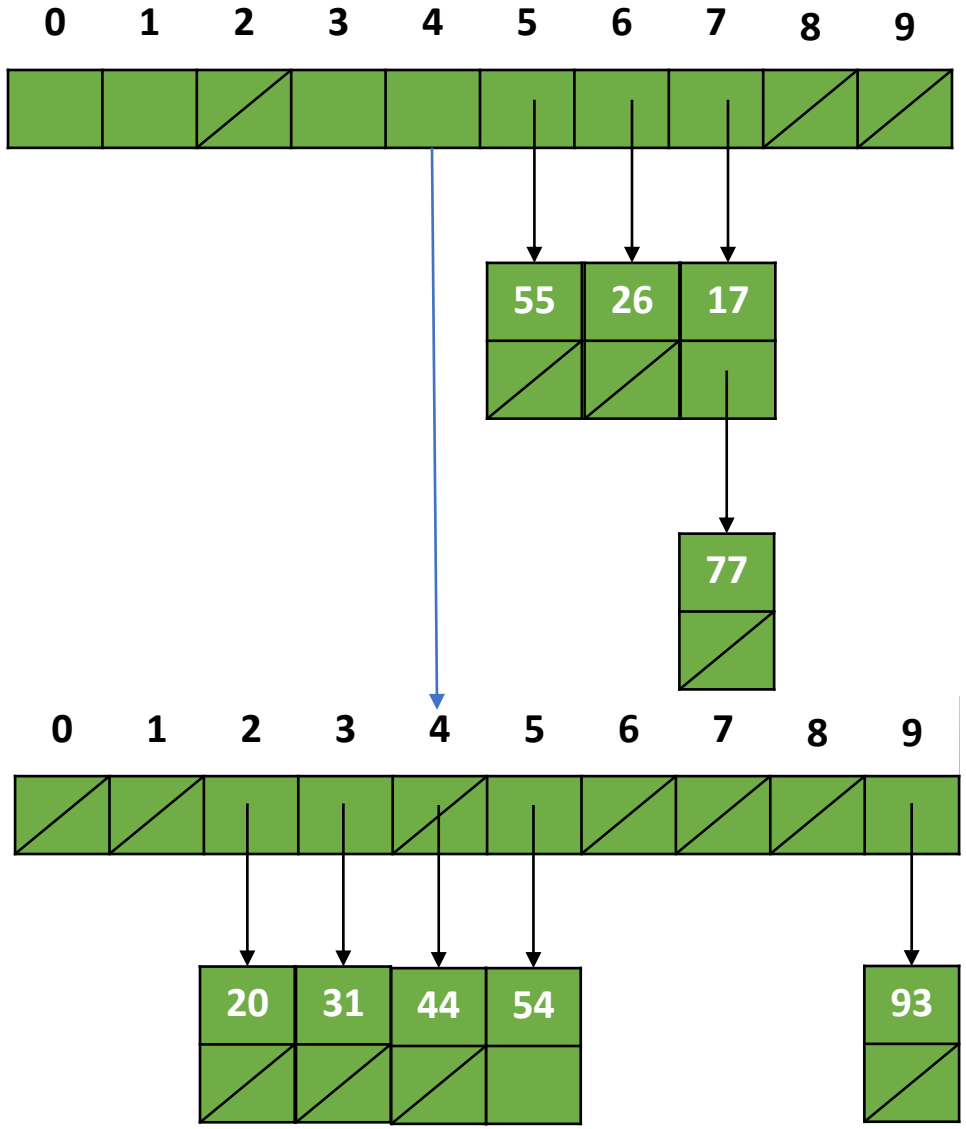
Radix Sort - LSD



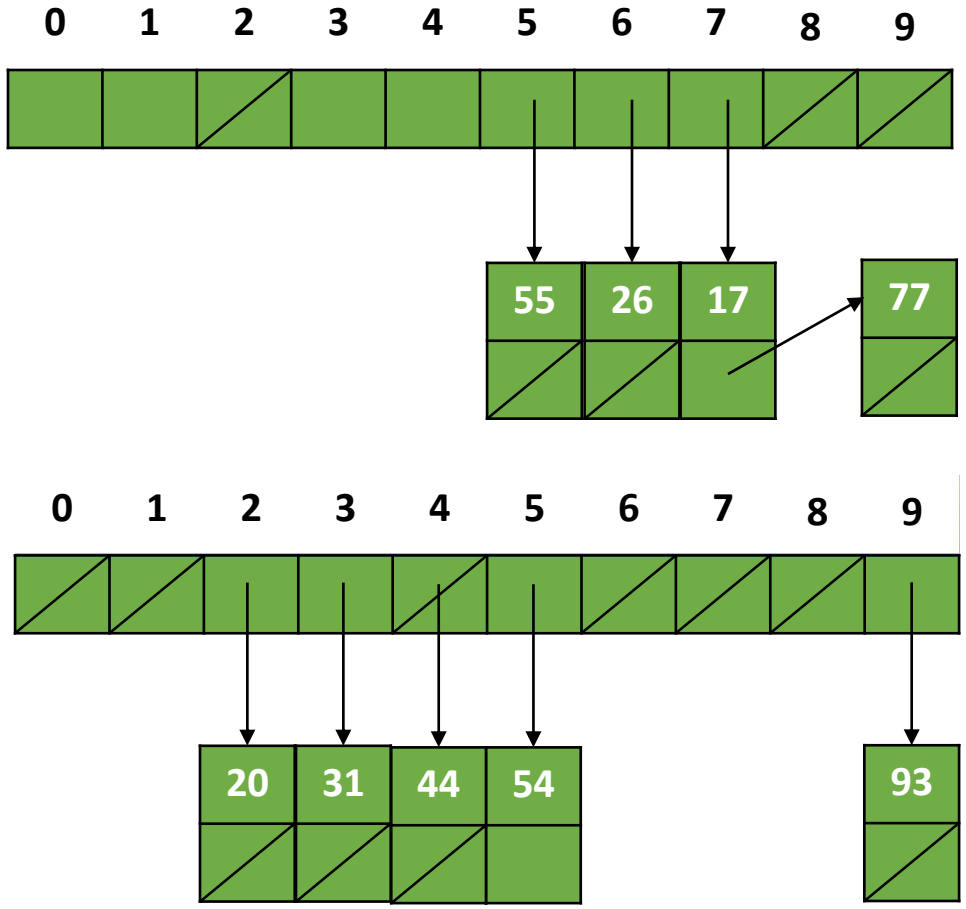
Radix Sort - LSD



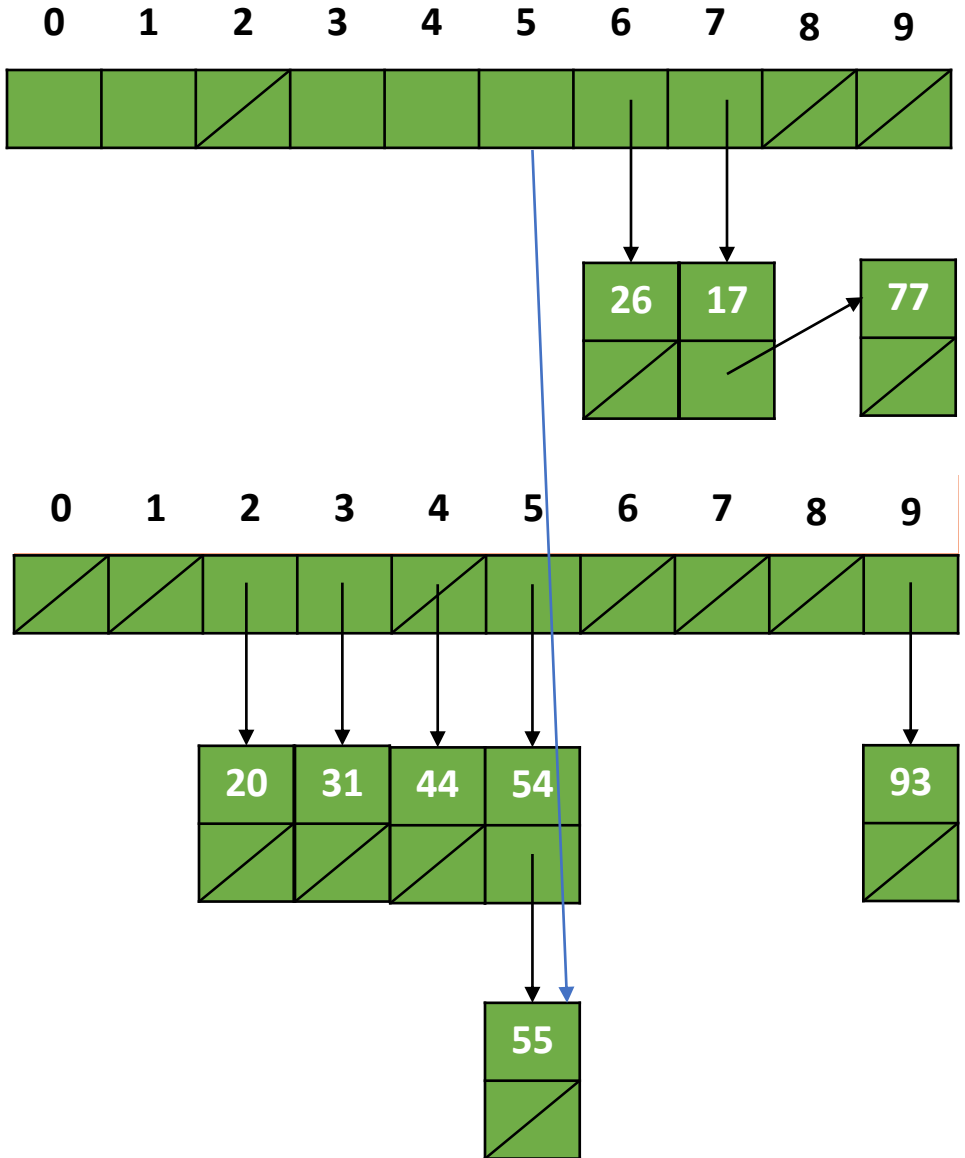
Radix Sort - LSD



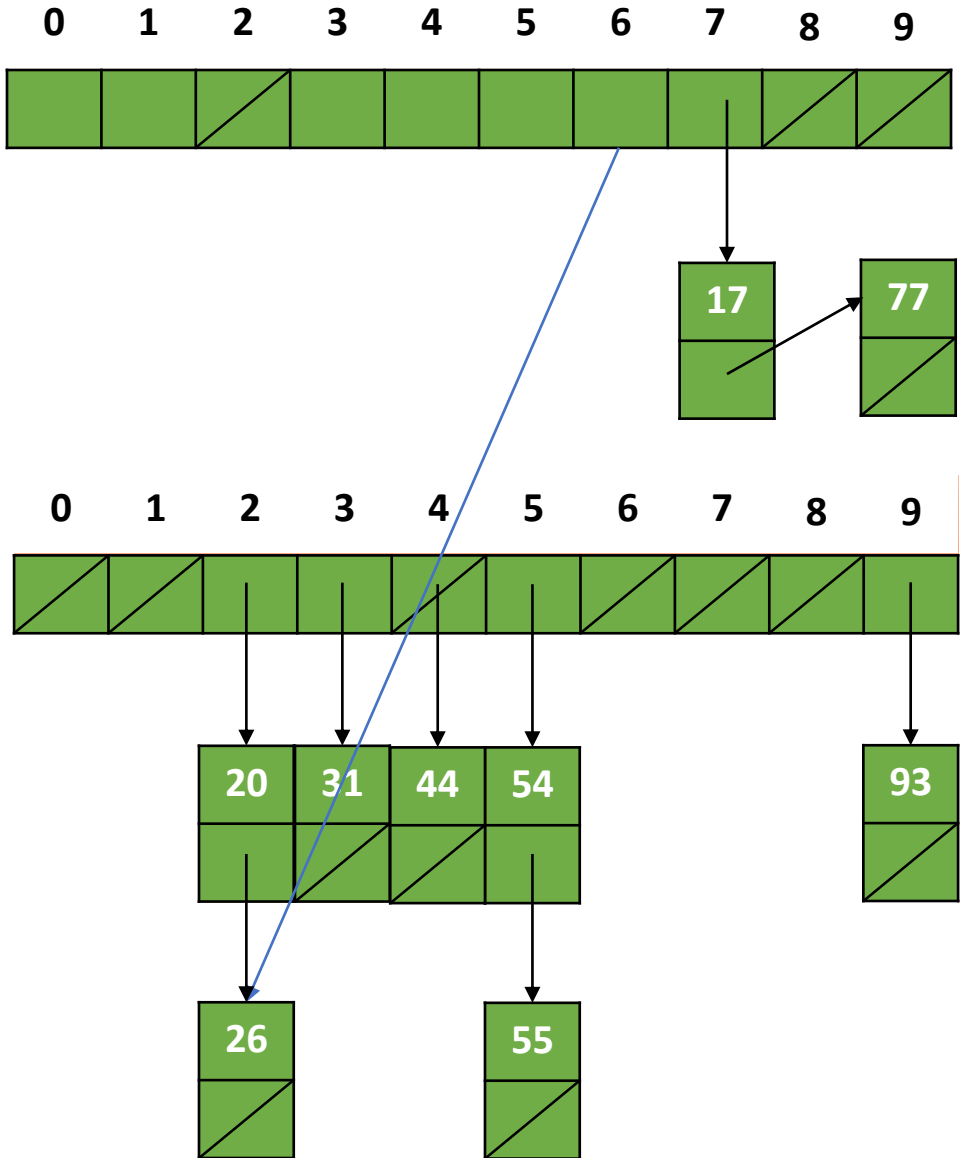
Radix Sort - LSD



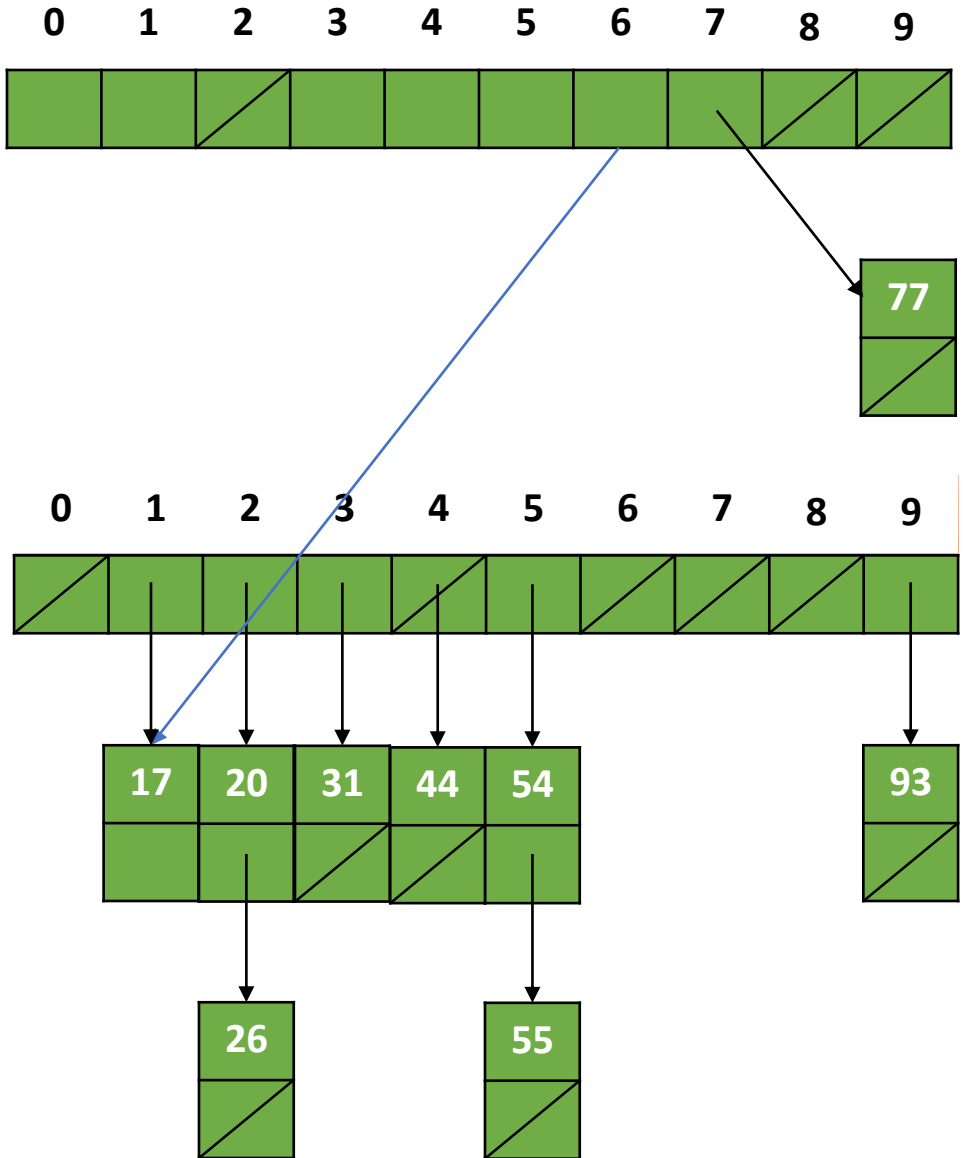
Radix Sort - LSD



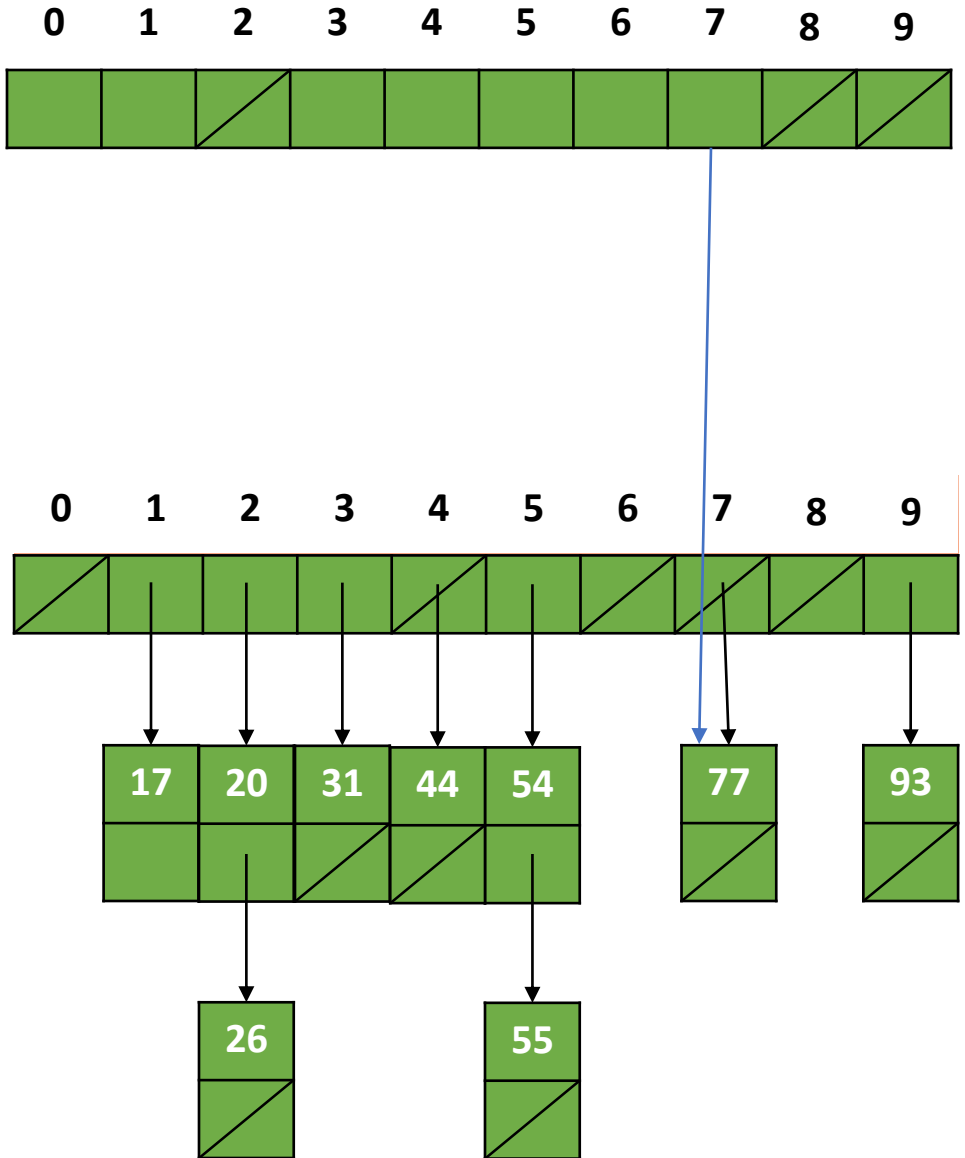
Radix Sort - LSD



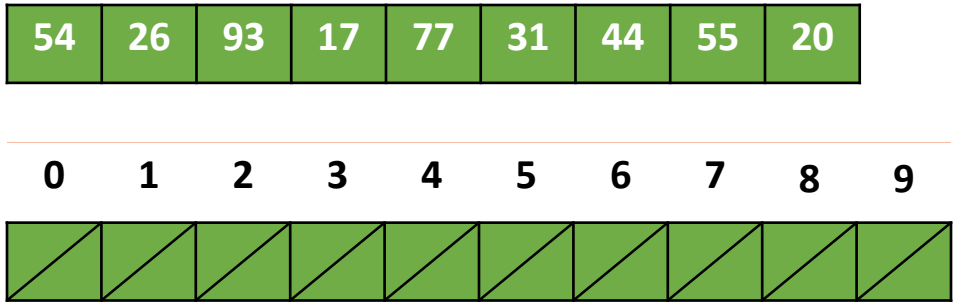
Radix Sort - LSD



Radix Sort - LSD

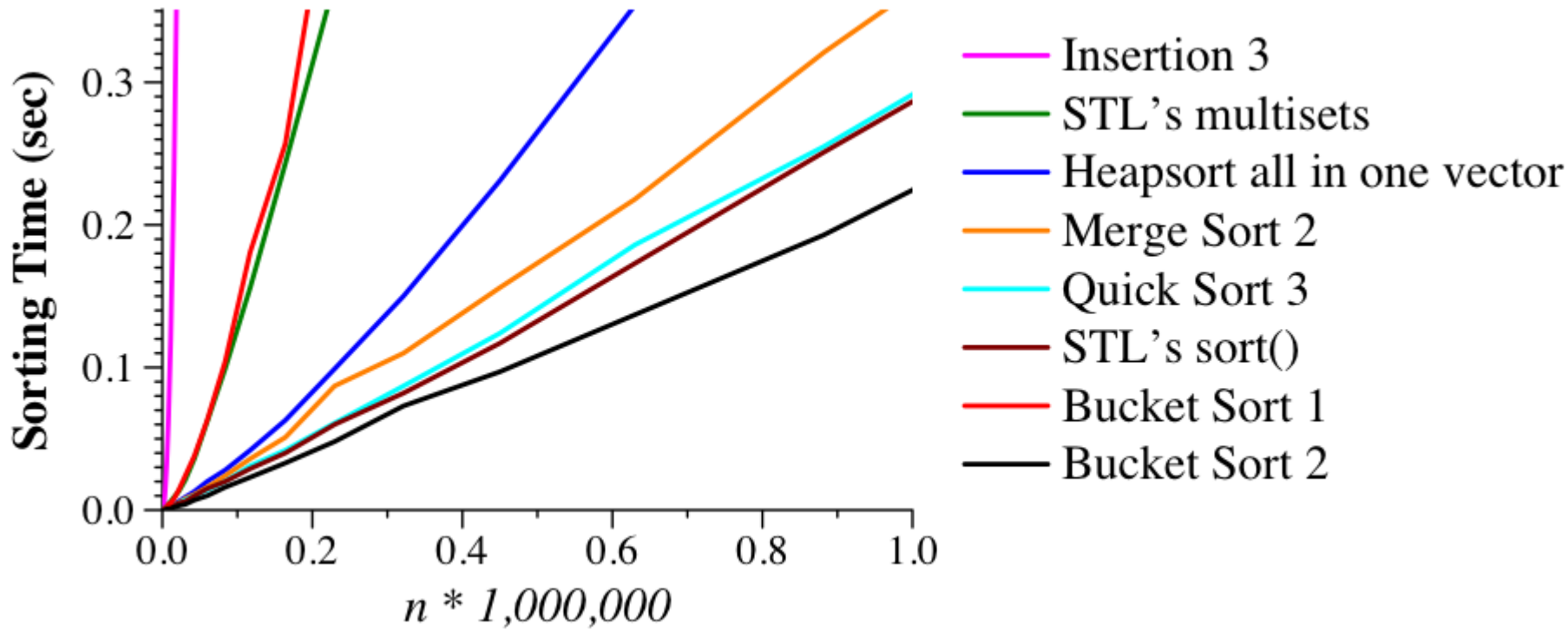


Radix Sort – MSD – Try it yourself



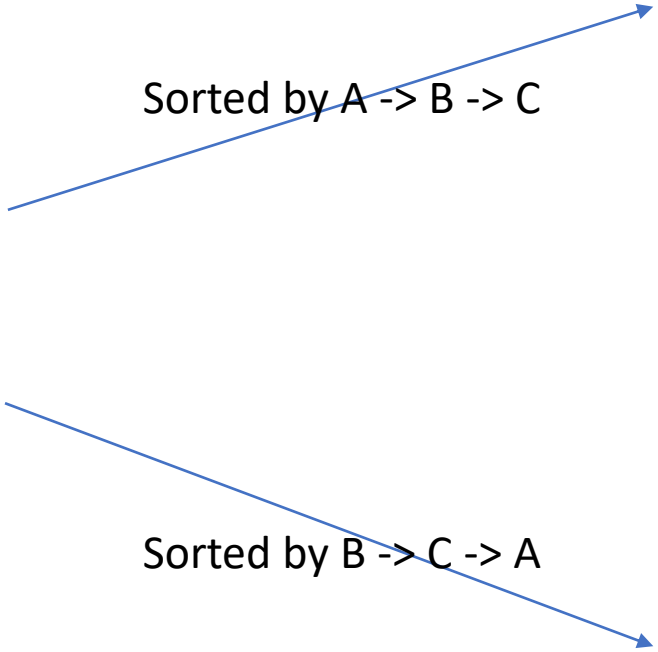
Bucket Sort

- ถ้าใช้ Array แทน Linked ผลจะเป็นอย่างไร
- MSD vs LSD
- เหมาะกับข้อมูลแบบไหน ไม่เหมาะกับข้อมูลแบบไหน


Sorting Time (sec)
0.0
0.1
0.2
0.3
0.0
0.2
0.4
0.6
0.8
1.0
n * 1,000,000
Insertion 3
STL's multisets
Heapsort all in one vector
Merge Sort 2
Quick Sort 3
STL's sort()
Bucket Sort 1
Bucket Sort 2

โบนัส 2 คะแนน multi-column sorting

No	A	B	C
1	1	1	4
2	3	1	1
3	4	4	4
4	2	4	4
5	3	5	3
6	4	3	3
7	1	3	3
8	2	4	3
9	3	3	5
10	1	5	3
11	1	1	4
12	4	1	1
13	5	2	3
14	3	5	2



No	A	B	C
1	1	1	4
11	1	1	4
7	1	3	3
10	1	5	3
8	2	4	3
4	2	4	4
2	3	1	1
9	3	3	5
14	3	5	2
5	3	5	3
12	4	1	1
6	4	3	3
3	4	4	4
13	5	2	3

No	A	B	C
2	3	1	1
12	4	1	1
1	1	1	4
11	1	1	4
13	5	2	3
7	1	3	3
6	4	3	3
9	3	3	5
8	2	4	3
4	2	4	4
3	4	4	4
14	3	5	2
10	1	5	3
5	3	5	3