UNIT- V

SEARCHING AND SORTING

Contents

Searching- Search Techniques, Sequential search, variant of sequential search- sentinel search, Binary search, Fibonacci search, Case Study- Use of Fibonacci search in non-uniform access memory storage and in Optimization of Uni modal Functions

Sorting- Types of sorting-Internal and external sorting, General sort concepts-sort order, stability, efficiency, number of passes, Bubble sort, Insertion sort, Selection sort, Quick sort, Shell sort, Bucket sort, Radix sort, Comparison of All Sorting Methods, Case Study- Time sort as a hybrid stable sorting algorithm.

Searching

- one of the most common and time consuming tasks in computer science is the retrieval of target information from huge data, which needs searching.
- **The process of locating target data is known as searching.**
- Depending on the way data is scanned for searching a particular record, the search techniques are categorized as follows:

1. Sequential search
2. Binary search
3. Fibonacci search

MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Sequential Search

- The easiest search technique is a sequential search.
- Let us assume that we have a sequential file F, and we wish to retrieve a record with a certain key value k.
- If F has n records with the key value ki such as i =1 to n, then one way to carry out the retrieval is by examining the key values in the order of their arrangement until the correct record is located.
- Such a search is known as sequential search since the records are examined sequentially from the first till the last.

Linear search

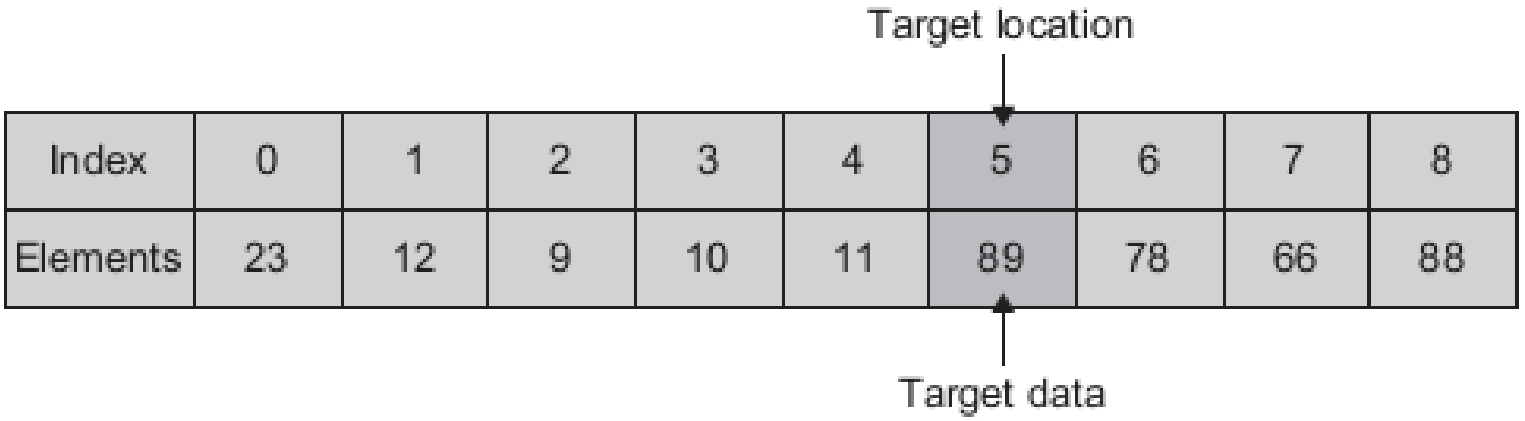
Array

6	3	0	5	1	2	8	-1	4

Element to search: 8

Sequential Search

- search for the target data of 89.



- Initially, i =0 and the target element 89 is to be searched. At each pass, the target 89 is compared with the element at the i^{th} location till it is found or the index I exceeds the size. At i =5, the search is successful.

Sequential Search

```
int SeqSearch (int A[max], int key, int n)
{
    int i, flag = 0, position;
    for(i = 0; i < n; i++)
    {
        if(key == A[i])
        {
            position = i;
            flag = 1;
            break;
        }
    }
    if(flag == 1) // if found return position
    return(position);
    else // return −1 if not found
    return(−1);
}
```

Pros and Cons of Sequential Search

Pros

1. A simple and easy method
2. Efficient for small lists
3. Suitable for unsorted data
4. Suitable for storage structures which do not support direct access to data, for example, magnetic tape, linked list, etc.
5. Best case is one comparison, worst case is n comparisons, and average case is (n +1)/2 comparisons
6. Time complexity is in the order of n denoted as O(n).

Cons

1. Highly inefficient for large data
2. In the case of ordered data other search techniques such as binary search are found more suitable.

Variations of Sequential Search

- There are three such variations:
- 1. Sentinel search
- 2. Probability search
- 3. Ordered list search

Sentinel search

- In simple linear search two comparisons are done: one for the element (key) to be searched and the other for the end of the array.
- The algorithm ends either when the target is found or when the last element is compared.
- The algorithm can be modified by **placing the target at the end of list as just one additional entry.**
- This additional entry at the end of the list is called as a **sentinel.**

Sentinel search

```
int SeqSearch_sentinel (int A[max], int key, int n)
{
    int i, position;
    A[n] = key; // place target at end of the list
    while(key != A[i])
    {
        i = i + 1;
    }
    //if found at sentinel then return position
    if(i < n)
            return(i);
    else // return −1 if not found
            return(−1);
}
```

Binary Search

- Let a_i, $1 \leq i \leq n$ be a list of elements which are sorted in increasing order.
- Consider the problem of determining whether a given element x is present in the list.
- If x is present, we are to determine a value j such that $a_j = x$.
- If x is not in the list then j is to be set to zero.

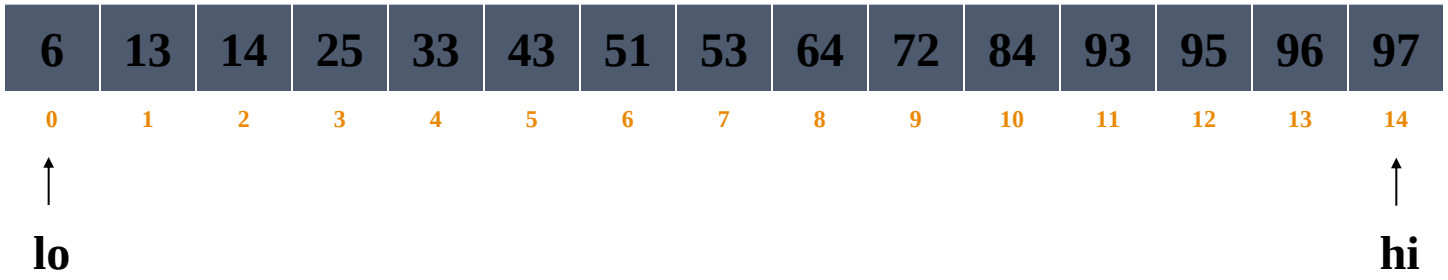
Binary Search

MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

```
Algorithm BINSRCH(a, i, l, x)
//given an array A(i : l) of elements in increasing order, 1 ≤ i ≤ l determine if x is present,
    and if so, set j such that x = a[j] else return 0
{
if (l = i) then
    {
            if( x = a[i]) then return i;
            Else return 0;
    }
Else
    { //reduce P into a smaller subproblem
    mid= ⌊ (i+l)/2 ⌋;
    if (x=a[mid] )then return mid;
    Else if (x<a[mid]) then
    Return BINSRCH(a, i, mid-1,x);
    Else Return BINSRCH(a, i, mid+1,x);
    }
}
```

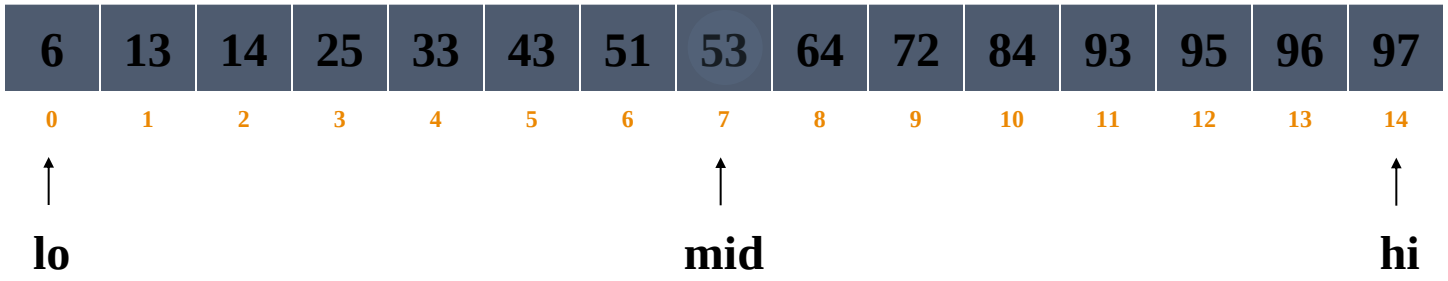
Binary Search

- Ex. Binary search for 33.



Binary Search

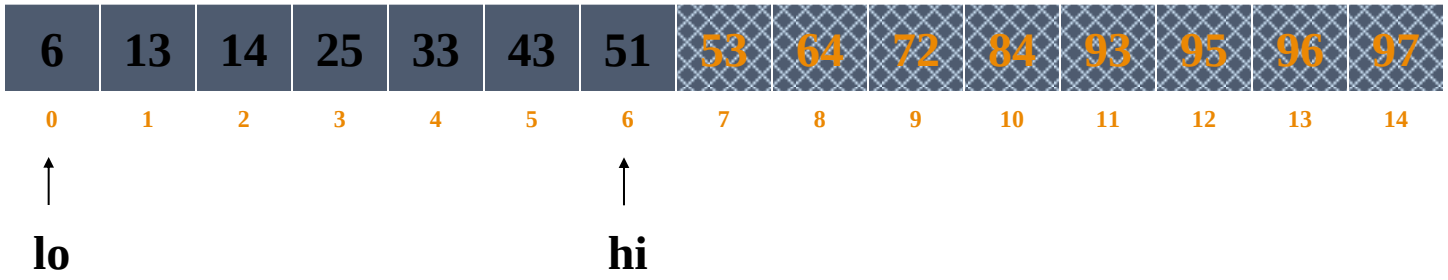
- Ex. Binary search for 33.



MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

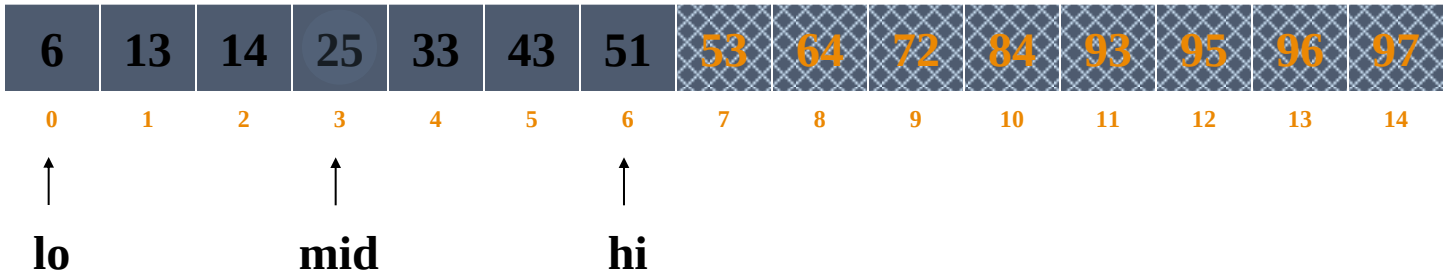
Binary Search

- Ex. Binary search for 33.



Binary Search

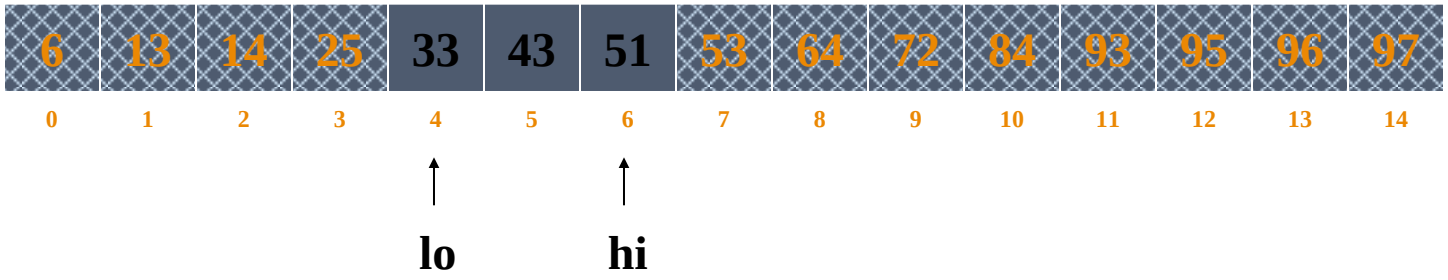
- Ex. Binary search for 33.



MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Binary Search

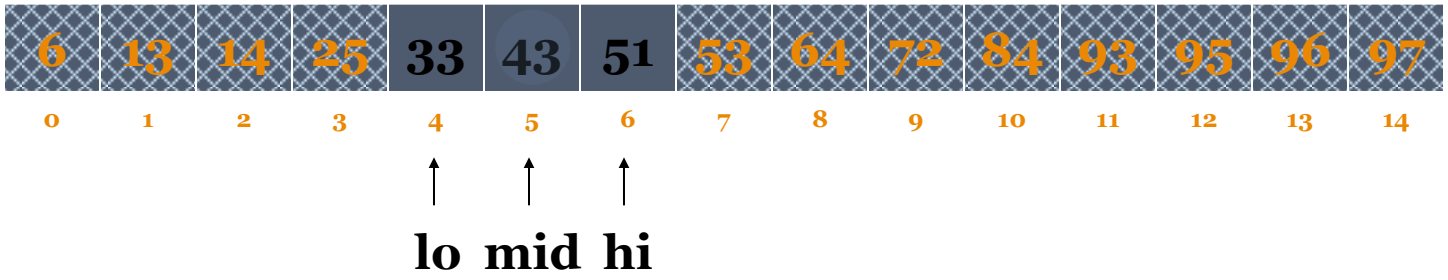
- Ex. Binary search for 33.



MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Binary Search

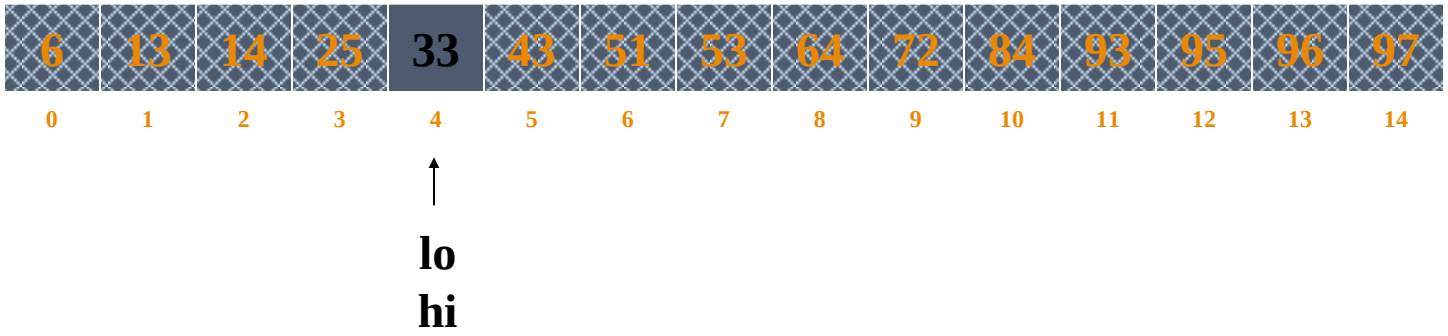
- Ex. Binary search for 33.



MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Binary Search

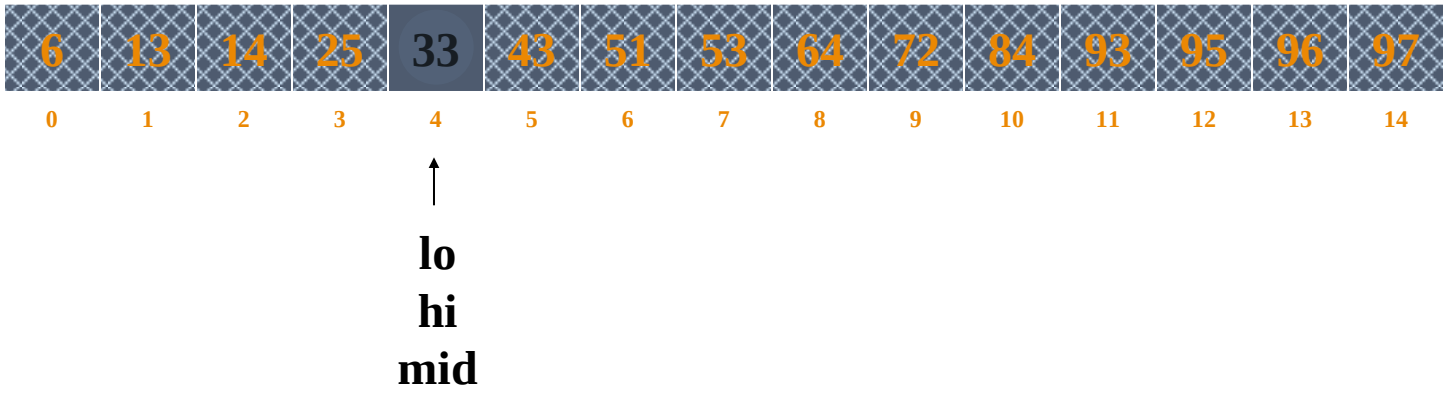
- Ex. Binary search for 33.



MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Binary Search

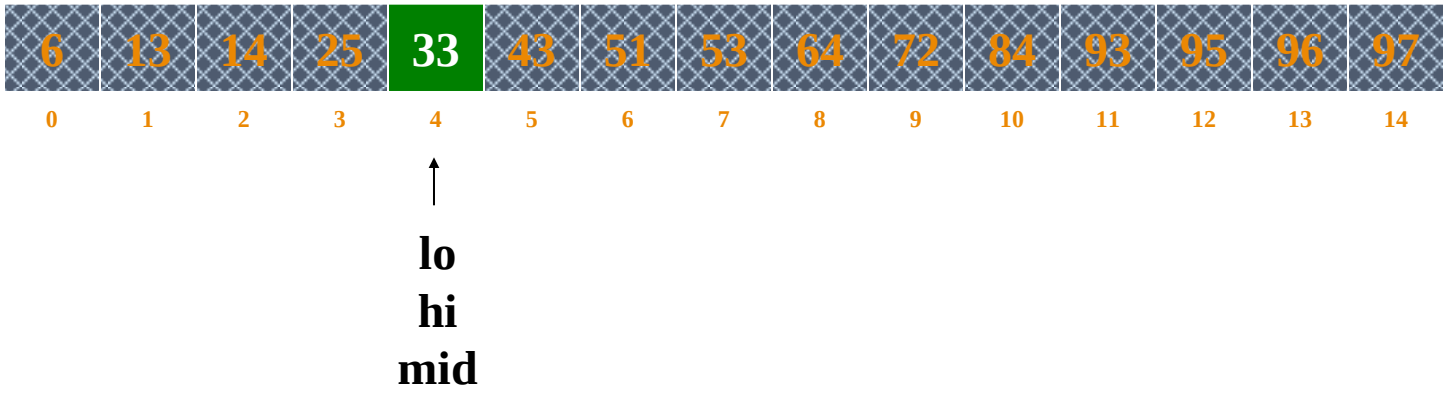
- Ex. Binary search for 33.



MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Binary Search

- Ex. Binary search for 33.



- Time complexity of binary search is O(log(n)) as it halves the list size in each step.

Pros and Cons of Binary Search

Pros

1. Suitable for sorted data
2. Efficient for large lists
3. Suitable for storage structures that support direct access to data
4. Time complexity is O(log2(n))

Cons

1. Not applicable for unsorted data
2. Not suitable for storage structures that do not support direct access to data, for example,

magnetic tape and linked list

3. Inefficient for small lists

Fibonacci Search

- **Differences with Binary Search**:
- Fibonacci Search divides given **array into unequal parts**
- Binary Search uses a division operator to divide range. Fibonacci Search doesn't use /, but uses + and -. The division operator may be costly on some CPUs.
- Fibonacci Search examines relatively closer elements in subsequent steps. So when the input array is big that cannot fit in CPU cache or even in RAM, Fibonacci Search can be useful.

Fibonacci Series

- **Step 1:** Find F(k) {k^{th} Fibonacci number} which is greater than or equal to n.
- **Step 2:** If F(k) = 0
  stop and print search unsuccessful.
- **Step 3:** Offset = -1
- **Step 4:** i = min(offset +F(k-2), n-1)
  if key = A[i]
  return i & stop searching.
- **Step 5:** If key > A[i]
  k = k-1
  offset = i
  repeat steps 4 & 5
  if key < A[i]
  k = k-2
  repeat steps 4 & 5

MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Fibonacci Search

	0	1	2	3	4	5	6	7	8	9
A	10	15	20	27	33	40	44	57	60	70

key = 60 and n = 10

k	0	1	2	3	4	5	6	7	8	9
F(k)	0	1	1	2	3	5	8	13	21	34

F(k) = 13 at k = 7

offset = -1

i = min(offset + F(k-2), n-1)

= min(-1+5, 9)

= min(4, 9)

= 4

i = 4

A[i] = 33

If (60 > 33)

k = k -1 = 7-1 = 6

offset = 4

k = 6 **offset = 4**

MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Fibonacci Search

	0	1	2	3	4	5	6	7	8	9
A	10	15	20	27	33	40	44	57	60	70

key = 60 and n = 10

k	0	1	2	3	4	5	6	7	8	9
F(k)	0	1	1	2	3	5	8	13	21	34

k = 6 ,offset = 4

i = min(offset + F(k-2), n-1)
= min(4+3, 9)
= min(7, 9)
= 7

i = 7
A[i] = 57
If (60 > 57)
k = k -1 = 6-1 = 5
offset = i = 7

k = 5 **offset = 7**

MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Fibonacci Search

	0	1	2	3	4	5	6	7	8	9
A	10	15	20	27	33	40	44	57	60	70

key = 60 and n = 10

k	0	1	2	3	4	5	6	7	8	9
F(k)	0	1	1	2	3	5	8	13	21	34

k = 5 **offset = 7**

i = min(offset + F(k-2), n-1)
= min(7+2, 9)
= min(9, 9)
= 9

i = 9
A[i] = 70
If (60 < 70)
k = k -2 = 6-2 = 4

k = 4 **offset = 7**

MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Fibonacci Search

	0	1	2	3	4	5	6	7	8	9
A	10	15	20	27	33	40	44	57	60	70

key = 60 and n = 10

k	0	1	2	3	4	5	6	7	8	9
F(k)	0	1	1	2	3	5	8	13	21	34

k = 4 **offset = 7**

i = min(offset + F(k-2), n-1)
= min(7+1, 9)
= min(8, 9)
= 9

i = 8
A[i] = 60
60 = 60
Offset = i =8
Successful search

Pros and Cons of Fibonacci Search

Pros

1. Faster than binary search for larger lists
2. Suitable for sorted lists

Con

1. Inefficient for smaller lists

Sorting

- One of the fundamental problem
- **Sorting** is the operation of arranging the records of a table according to the key value of each record, or it can be defined as the process of converting an unordered set of elements to an ordered set.

Types of Sorting

- Sorting algorithms are divided into two categories:
  - Internal sort
  - External sort

Internal sort

- Any sort algorithm that uses main memory exclusively during the sorting is called as an internal sort algorithm.
- Internal sorting is faster than external sorting. The various internal sorting techniques are the following:

1. Bubble sort
2. Insertion sort
3. Selection sort
4. Quick sort
5. Heap sort
6. Shell sort
7. Bucket sort
8. Radix sort
9. File sort
10. Merge sort
11. external sort

External Sorting

- Any sort algorithm that uses external memory, such as tape or disk, during the sorting is called as an external sort algorithm.
- Merge sort uses external memory.
- other algorithms may read the initial values from a magnetic tape or write sorted values to a disk, but they do not use external memory during the sort.

General Sort concepts

- **Sort Order :** The order in which the data is organized, either ascending or descending, is called sort order.
- Example: percentage- descending order and telephone directory are organized alphabetically in ascending order.

General Sort concepts

- **Sort Stability :** A sorting method is said to be stable if at the end of the method, identical elements occur in the same relative order as in the original unsorted set.

Name	Uma	Saurabh	Sanika	Kasturi	Ashish	Harsha	Lelo
Marks	80	90	93	95	83	90	83

Solution The stable sort method will sort the sequence as

Name	Kasturi	Sanika	**Saurabh**	**Harsha**	**Ashish**	**Lelo**	Uma
Marks	95	93	**90**	**90**	**83**	**83**	80

whereas, the unstable sort method may sort the same sequence as

Name	Kasturi	Sanika	**Harsha**	**Saurabh**	**Lelo**	**Ashish**	Uma
Marks	95	93	**90**	**90**	**83**	**83**	80

General Sort concepts

- **Sort Efficiency :**
- Amount of time necessary for running the program
- the amount of space required for the program.
- The amount of time for running a program is proportional to the number of key comparisons and the movement of records or the movement of pointers to records.

General Sort concepts

- **Passes :** During the sorted process, the data is traversed many times.
- Each traversal of the data is referred to as a sort pass.

Bubble sort

- The bubble sort is the oldest and the simplest sort in use. Unfortunately, it is also the slow-est.
- The bubble sort works by comparing each item in the list with the item next to it and swapping them if required.

6 5 3 1 8 7 2 4

MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Bubble sort

```
void bubblesort(int A[max], int n)
{
    int i, j,temp;
    for(i = 1; i < n; i++) // number of passes
        {
        for(j = 0; j < n − i; j++) // j varies from 0 to // n − i
        {
            if( A[j] > A[j + 1] ) // compare two successive  numbers
            {
            temp = A[j]; // swap A[j] with A[j + 1]
            A[j] = A[j + 1];
            A[j + 1] = temp;
        }
        }
    }
}
```

MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Bubble Sort

```
for i = 1 to n – 1
      for j = 0 to n – i-1
      begin
            Swap
       end
end
```

n = 8

Array	i	J= 0 to 6	6	5	3	1	8	7	2	4
Pass1	1	0	5	6	3	1	8	7	2	4
	1	1	5	3	6	1	8	7	2	4
	1	2	5	3	1	6	8	7	2	4
	1	3	5	3	1	6	8	7	2	4
	1	4	5	3	1	6	7	8	2	4
	1	5	5	3	1	6	7	2	8	4
	1	6	5	3	1	6	7	2	4	8

MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Bubble Sort

```
for i = 1 to n – 1
      for j = 0 to n – i-1
      begin
            Swap
       end
end
```

n = 8

Array	i	J= 0 to 5	5	3	1	6	7	2	4	8
Pass2	2	0	3	5	1	6	7	2	4	8
	2	1	3	1	5	6	7	2	4	8
	2	2	3	1	5	6	7	2	4	8
	2	3	3	1	5	6	7	2	4	8
	2	4	3	1	5	6	2	7	4	8
	2	5	3	1	5	6	2	4	7	8

MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Bubble Sort

```
for i = 1 to n – 1
      for j = 0 to n – i-1
      begin
            Swap
       end
end
```

n = 8

Array	i	J= 0 to 4	3	1	5	6	2	4	7	8
Pass3	3	0	1	3	5	6	2	4	7	8
	3	1	1	3	5	6	2	4	7	8
	3	2	1	3	5	6	2	4	7	8
	3	3	1	3	5	2	6	4	7	8
	3	4	1	3	5	2	4	6	7	8

MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Bubble Sort

```
for i = 1 to n – 1
      for j = 0 to n – i-1
      begin
            Swap
       end
end
```

n = 8

Array	i	J= 0 to 3	1	3	5	2	4	6	7	8
Pass4	4	0	1	3	5	2	4	6	7	8
	4	1	1	3	5	2	4	6	7	8
	4	2	1	3	2	5	4	6	7	8
	4	3	1	3	2	4	5	6	7	8

MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Bubble Sort

```
for i = 1 to n – 1
      for j = 0 to n – i-1
      begin
            Swap
       end
end
```

n = 8

Array	i	J= 0 to 2	1	3	2	4	5	6	7	8
Pass5	5	0	1	3	2	4	5	6	7	8
	5	1	1	2	3	4	5	6	7	8
	5	2	1	2	3	4	5	6	7	8

MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Bubble Sort

```
for i = 1 to n – 1
      for j = 0 to n – i-1
      begin
            Swap
       end
end
```

n = 8

Array	i	J= 0 to 1	1	2	3	4	5	6	7	8
Pass6	6	0	1	2	3	4	5	6	7	8
	6	1	1	2	3	4	5	6	7	8

MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Bubble Sort

```
for i = 1 to n – 1
      for j = 0 to n – i-1
      begin
            Swap
       end
end
```

n = 8

Array	i	J= 0 to 0	1	2	3	4	5	6	7	8
Pass7	7	0	1	2	3	4	5	6	7	8

No of passes =n-1

Analysis of Bubble Sort

- To sort a unsorted list with '**n**' number of elements we need to make

((n-1)+(n-2)+(n-3)+.....+1) = (n (n-1))/2

number of comparisons in the worst case.

If the list already sorted, then it requires '**n**' number of comparisons.

1. Average case complexity =$O(n^2)$
2. Best case complexity =$O(n)$
3. Worst case complexity =$O(n^2)$

MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Insertion Sort

- The insertion sort works just like its name suggests—it inserts each item into its proper place in the final list.

6 5 3 1 8 7 2 4

```
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;
         // Move elements of arr[0..i-1], that are greater than key
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

Array	i	j	76	67	36	55	23	14	6
Pass1	1	0	67	76	36	55	23	14	6

Array	i	j	67	76	36	55	23	14	6
Pass2	2	1	67	36	76	55	23	14	6
		0	36	67	76	55	23	14	6

Array	i	j	36	67	76	55	23	14	6
Pass3	3	2	36	67	55	76	23	14	6
		1	36	55	67	76	23	14	6
		0	36	55	67	76	23	14	6

Array	i	j	36	55	67	76	23	14	6
Pass4	4	3	36	55	67	23	76	14	6
		2	36	55	23	67	76	14	6
		1	36	23	55	67	76	14	6
		0	23	36	55	67	76	14	6

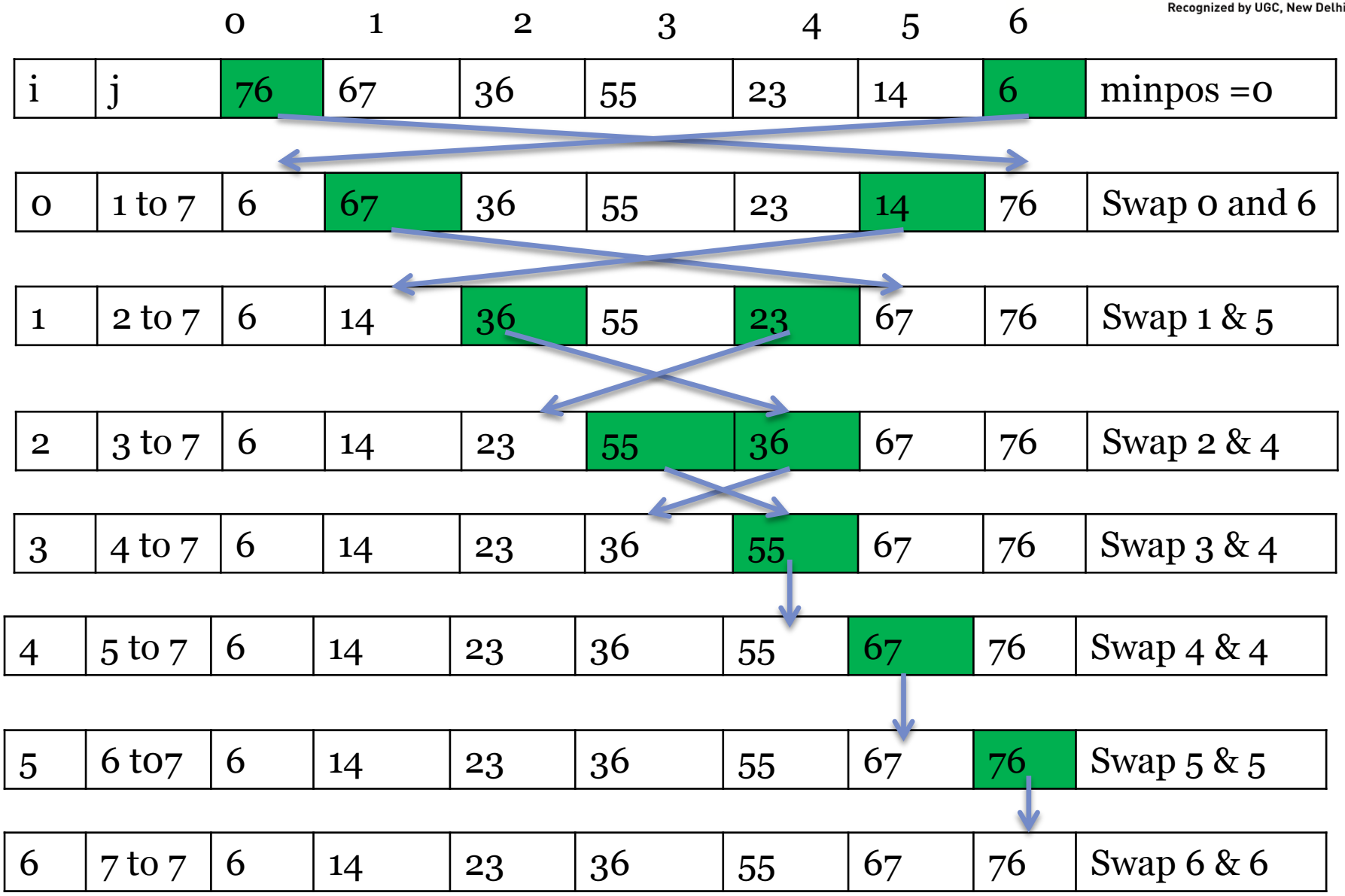
Array	i	j	23	36	55	67	76	14	6
Pass5	5	4	23	36	55	67	14	76	6
		3	23	36	55	14	67	76	6
		2	23	36	14	55	67	76	6
		1	23	14	36	55	67	76	6
		0	14	23	36	55	67	76	6

Array	i	j	14	23	36	55	67	76	6
Pass5	6	5	14	23	36	55	67	6	76
		4	14	23	36	55	6	67	76
		3	14	23	36	6	55	67	76
		2	14	23	6	36	55	67	76
		1	14	6	23	36	55	67	76
		0	6	14	23	36	55	67	76

Selection sort

- The selection sort algorithms construct the sorted sequence, one element at a time, by adding elements to the sorted sequence in order.

```
void SelectionSort(int A[], int n)
{
int i, j, minpos, temp;
for(i = 0; i < n − 1; i++)
{
    minpos = i;
    for(j = i + 1; j < n; j++) //find the position of min element as minpos from i + 1 to n − 1
    {
        if(A[j] < A[minpos])
        minpos = j;
    }
    if(minpos != i)
    {
        temp = A[i];    // swap the ith element and minpos element
        A[i] = A[minpos];
        A[minpos] = temp;
    }
    }
}
```


0 1 2 3 4 5 6
i j 76 67 36 55 23 14 6 minpos =0
0 1 to 7 6 67 36 55 23 14 76 Swap 0 and 6
1 2 to 7 6 14 36 55 23 67 76 Swap 1 & 5
2 3 to 7 6 14 23 55 36 67 76 Swap 2 & 4
3 4 to 7 6 14 23 36 55 67 76 Swap 3 & 4
4 5 to 7 6 14 23 36 55 67 76 Swap 4 & 4
5 6 to7 6 14 23 36 55 67 76 Swap 5 & 5
6 7 to 7 6 14 23 36 55 67 76 Swap 6 & 6

Selection sort Analysis

- The total number of comparisons is as follows:
- $(n -1) +(n -2) +\ldots +1 = n(n -1)/2$
- Therefore, the number of comparisons for the selection sort is proportional to n^2, which means that it is $O(n^2)$. The different cases are as follows:
- Average case: $O(n^2)$ Best case: $O(n^2)$ Worst case: $O(n^2)$

Quicksort Algorithm

Given an array of *n* elements (e.g., integers):

- If array only contains one element, return
- Else
  - pick one element to use as *pivot.*
  - Partition elements into two sub-arrays:
    - Elements less than or equal to pivot
    - Elements greater than pivot
  - Quicksort two sub-arrays
  - Return results

MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Example

We are given array of n integers to sort:

40	20	10	80	60	50	7	30	100

MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Pick Pivot Element

There are a number of ways to pick the pivot element.

In this example, we will use the first element in the array:

40	20	10	80	60	50	7	30	100

Partitioning Array

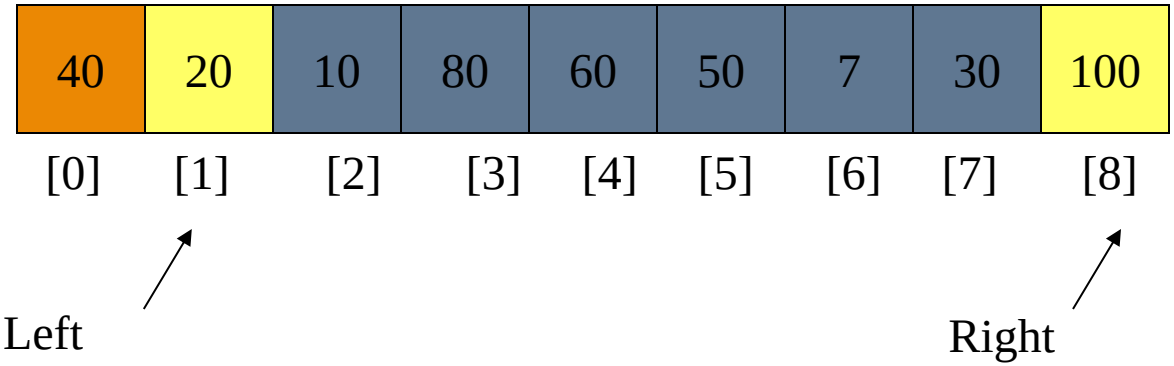
Given a pivot, partition the elements of the array such that the resulting array consists of:

1. One sub-array that contains elements >= pivot
2. Another sub-array that contains elements < pivot

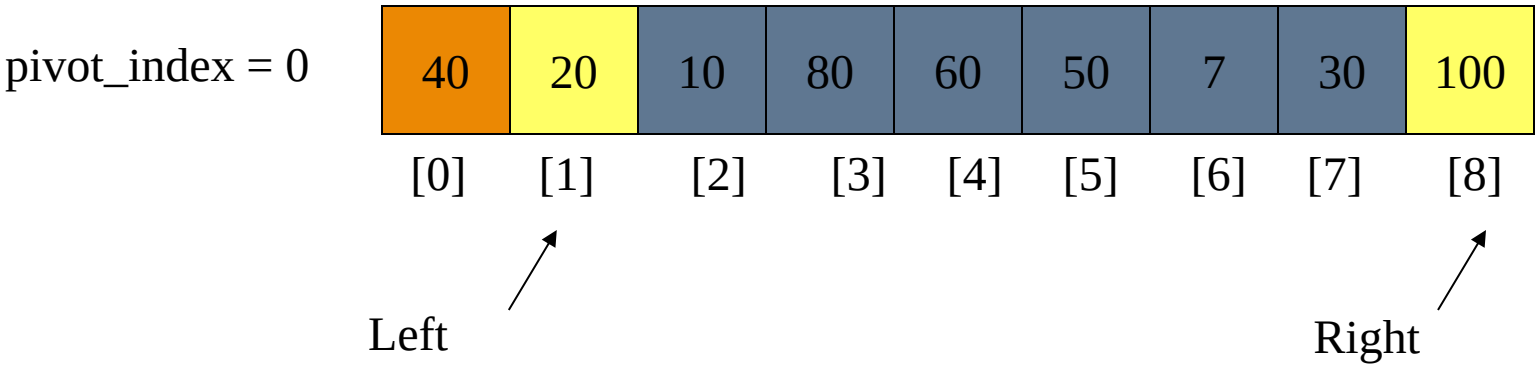
The sub-arrays are stored in the original data array.

Partitioning loops through, swapping elements below/above pivot.

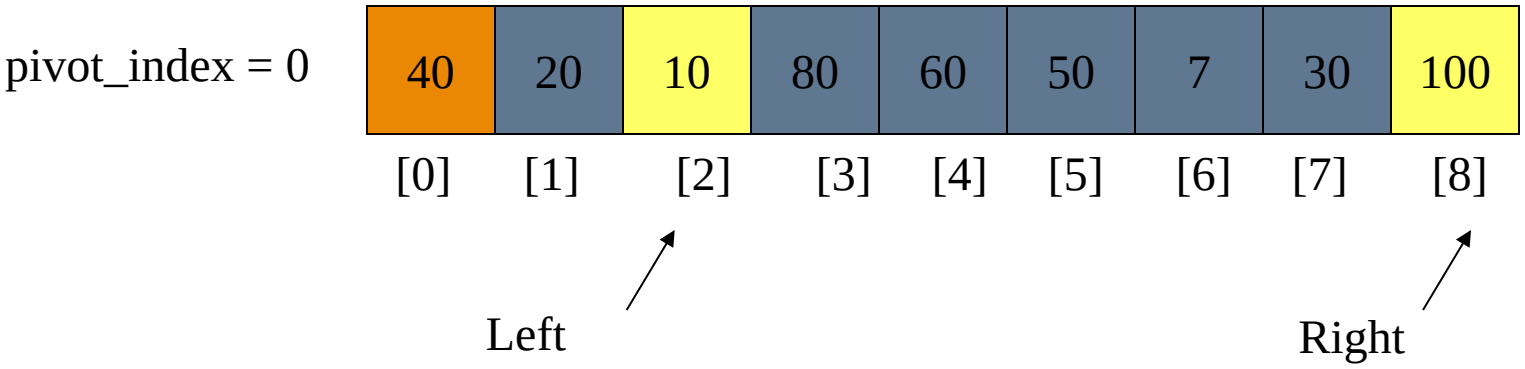
pivot_index = 0



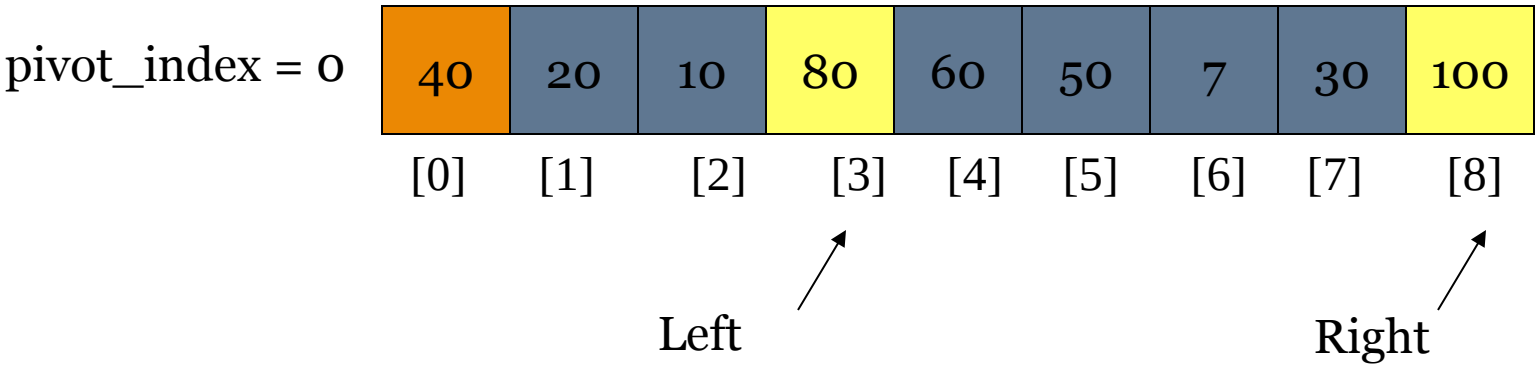
1. While data[Left] <= data[pivot]
   ++Left



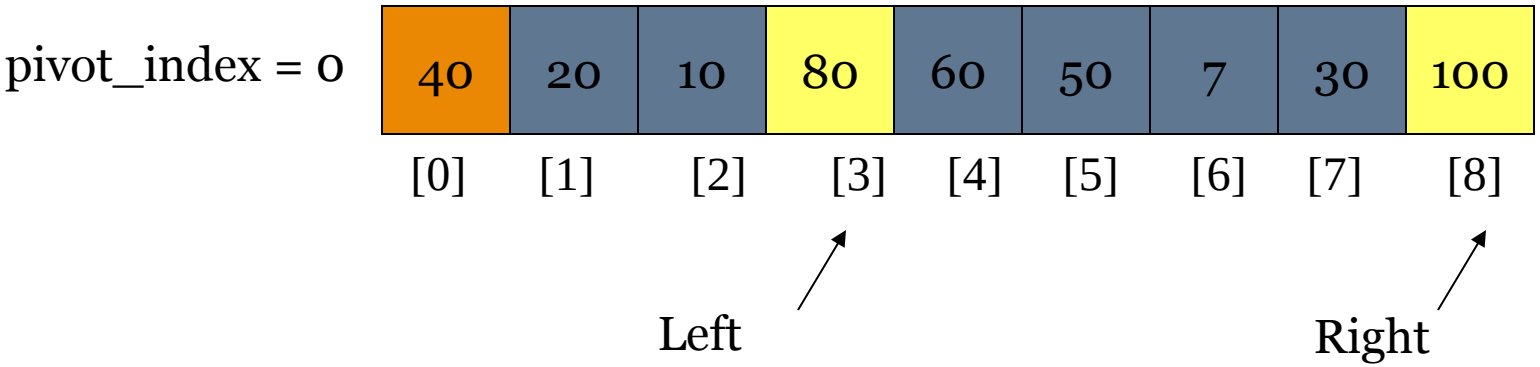
1. While data[Left] <= data[pivot]
   ++Left



1. While data[Left] <= data[pivot]
   ++Left

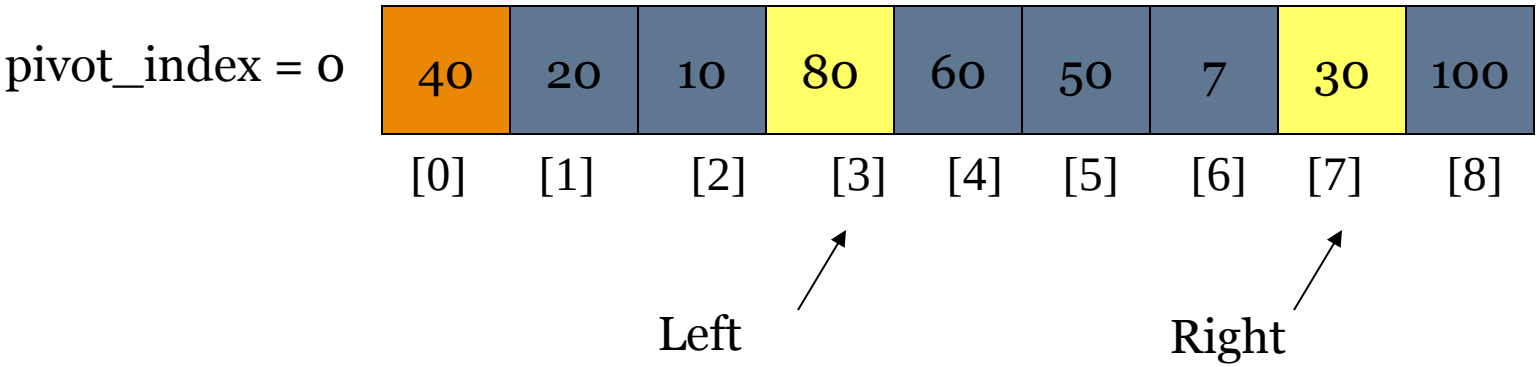


1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right

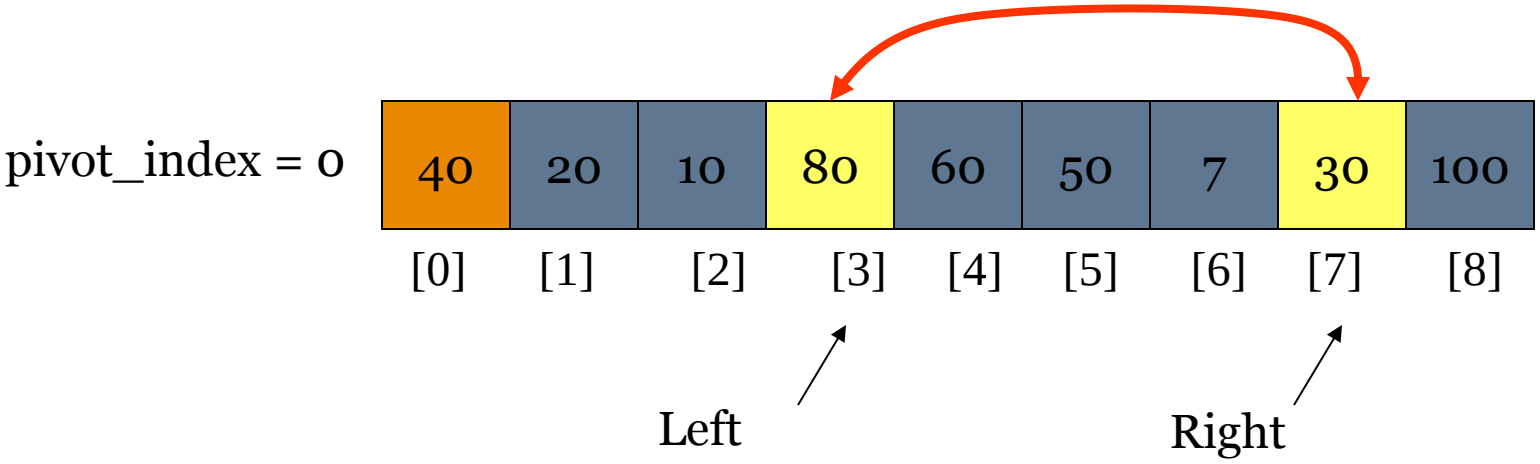


MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

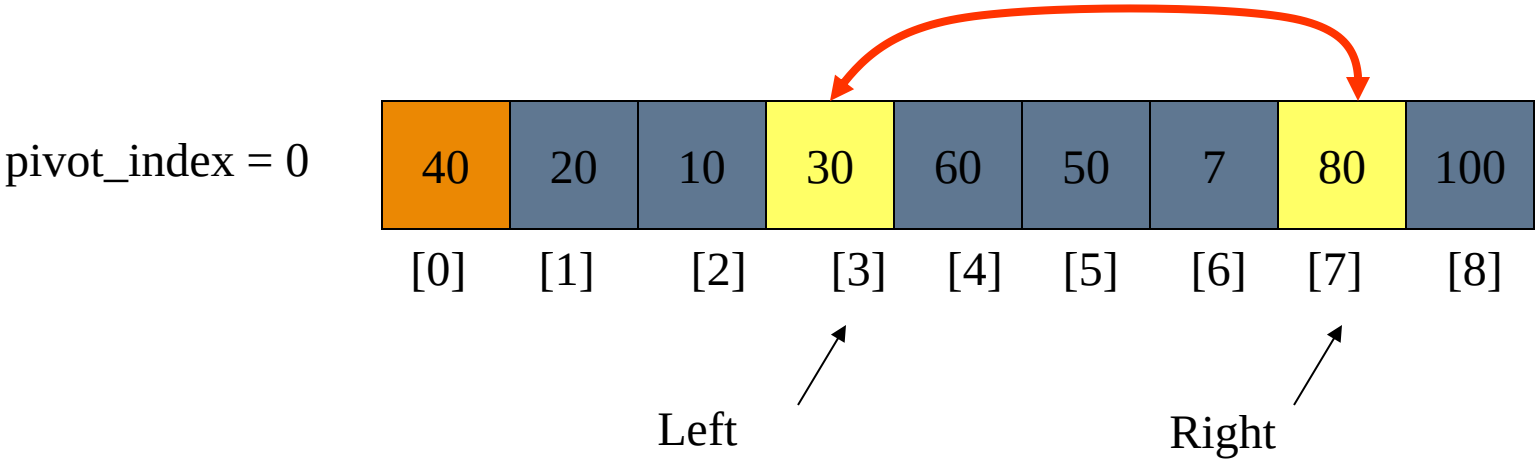
1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right



1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right
3. If Left > Right
   swap data[Left] and data[Right]

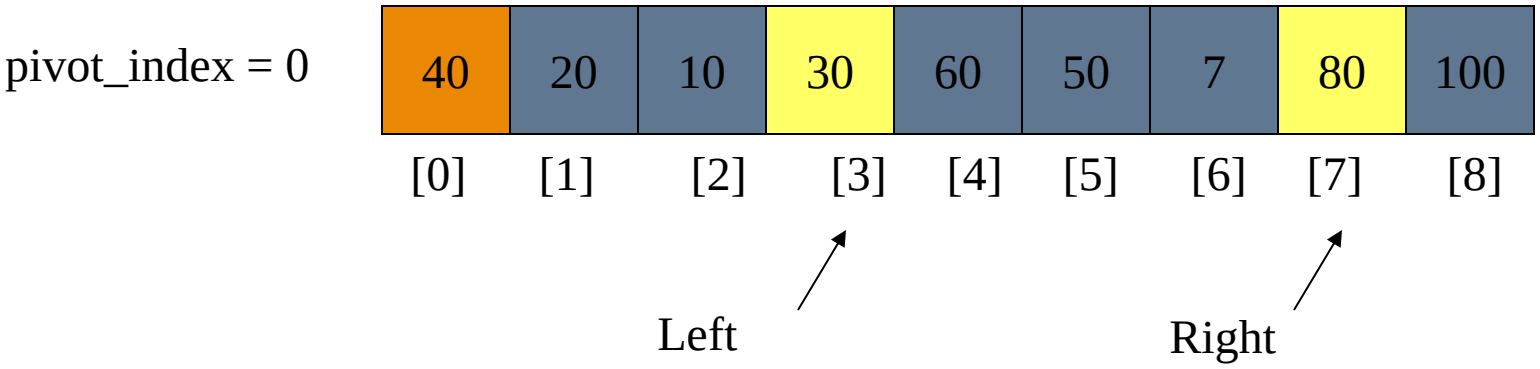


1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right
3. If Left > Right
   swap data[Left] and data[Right]

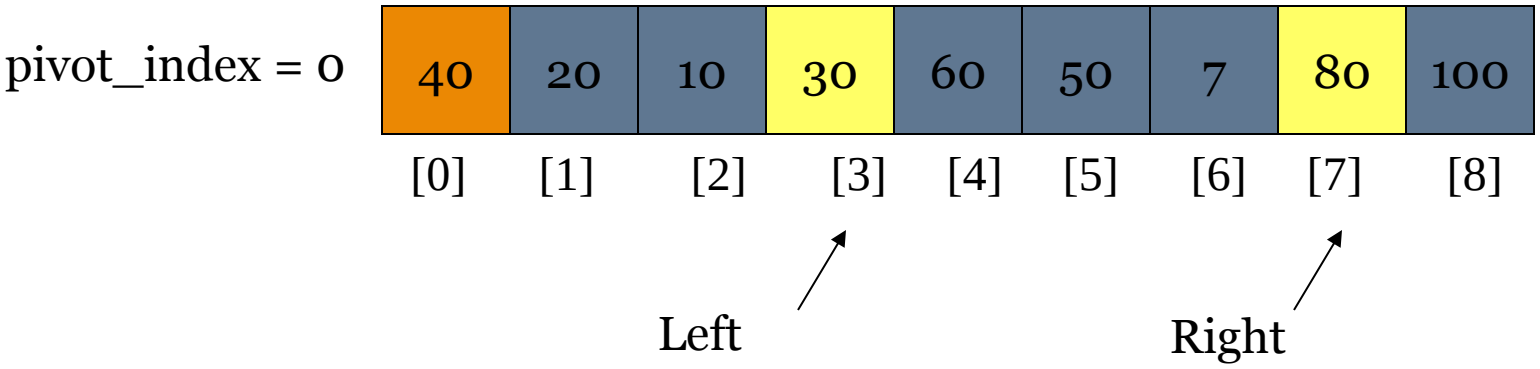


MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

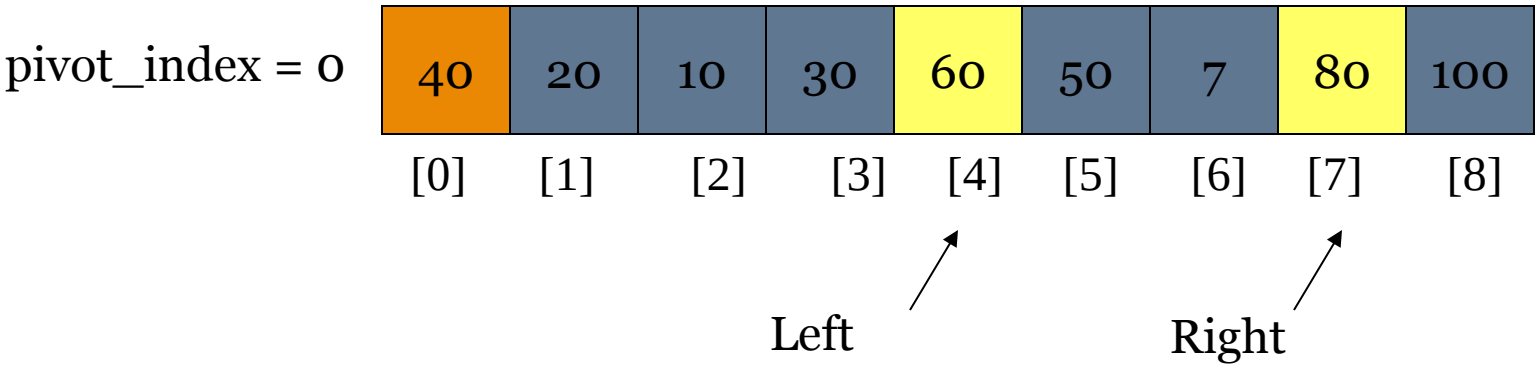
1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right
3. If Left > Right
   swap data[Left] and data[Right]
4. While Right > Left, go to 1.



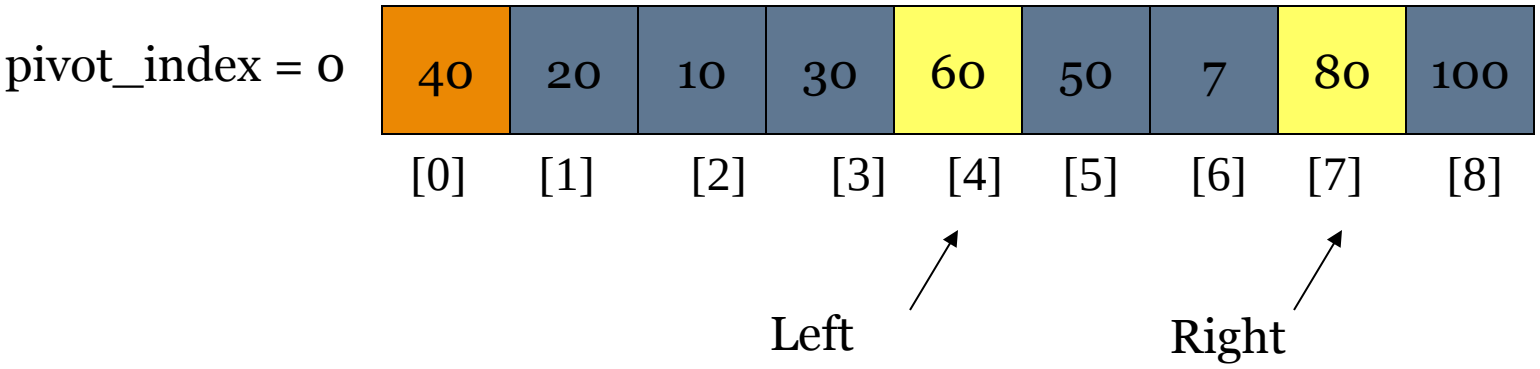
1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right
3. If Left > Right
   swap data[Left] and data[Right]
4. While Right > Left, go to 1.



1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right
3. If Left > Right
   swap data[Left] and data[Right]
4. While Right > Left, go to 1.

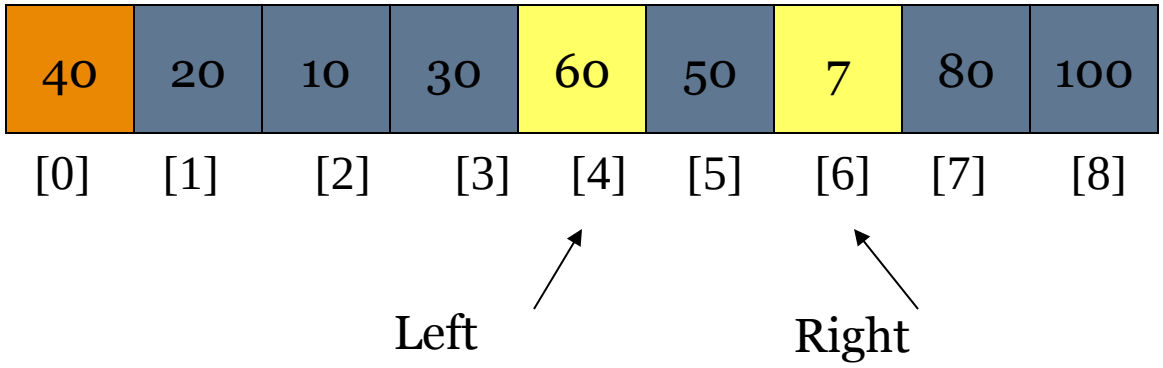


1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right
3. If Left > Right
   swap data[Left] and data[Right]
4. While Right > Left, go to 1.

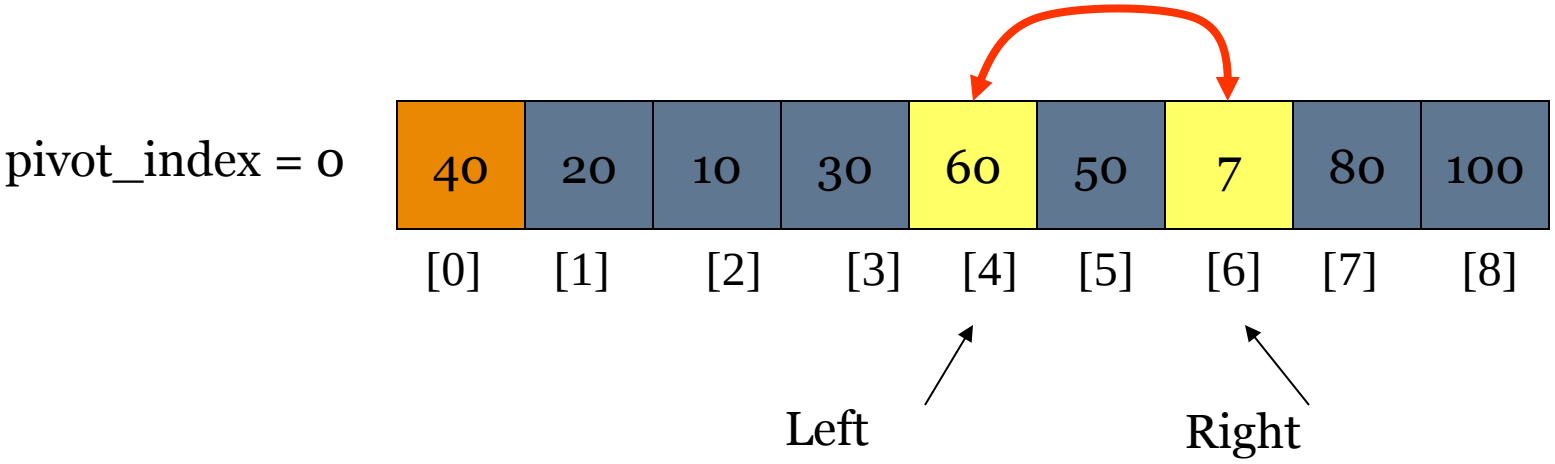


1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right
3. If Left > Right
   swap data[Left] and data[Right]
4. While Right > Left, go to 1.

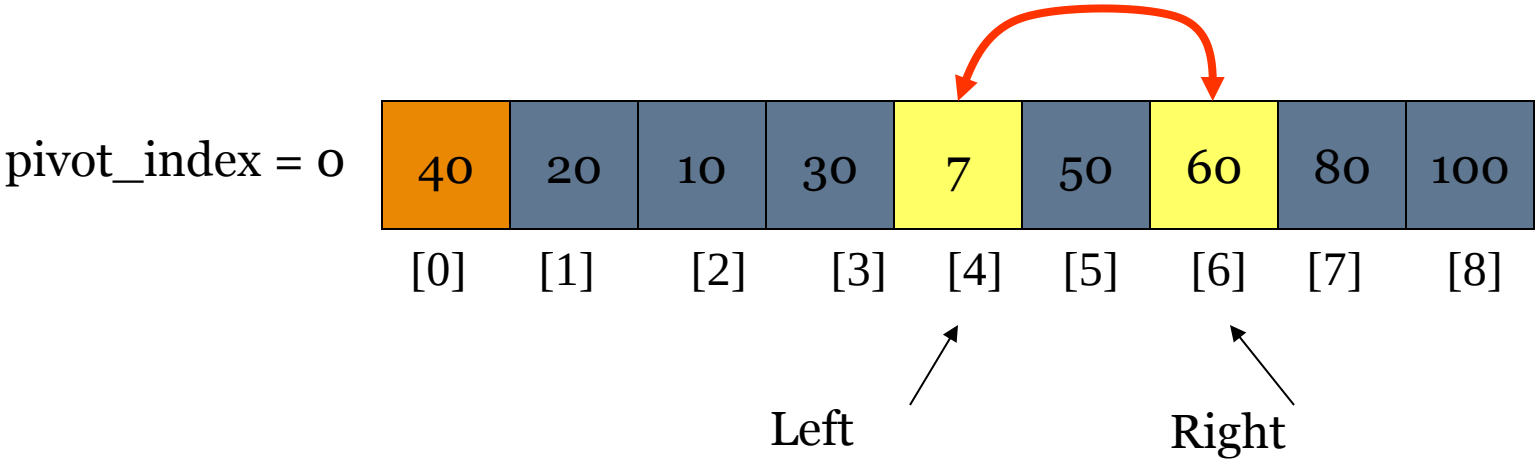
pivot_index = 0



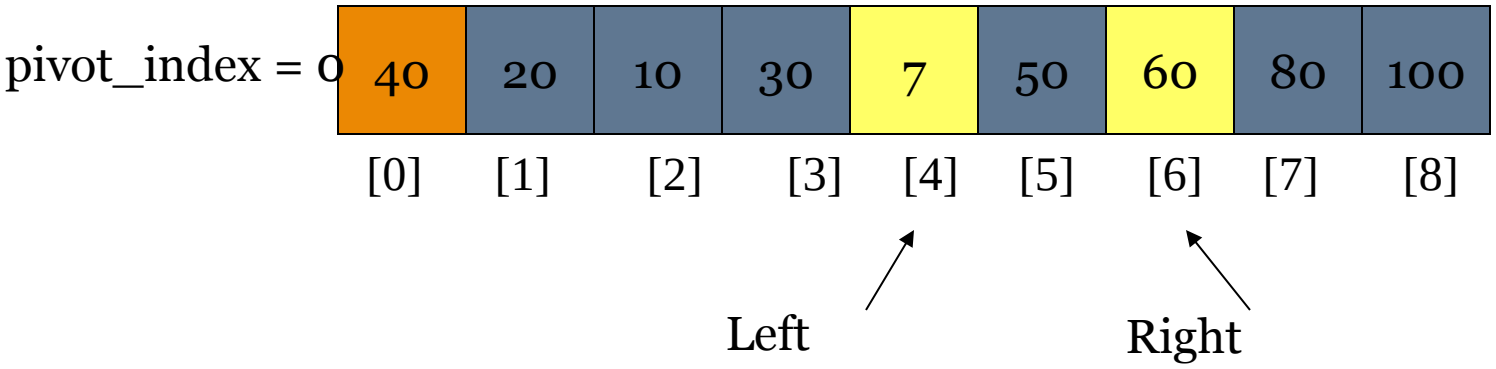
1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right
3. If Left > Right
   swap data[Left] and data[Right]
4. While Right > Left, go to 1.



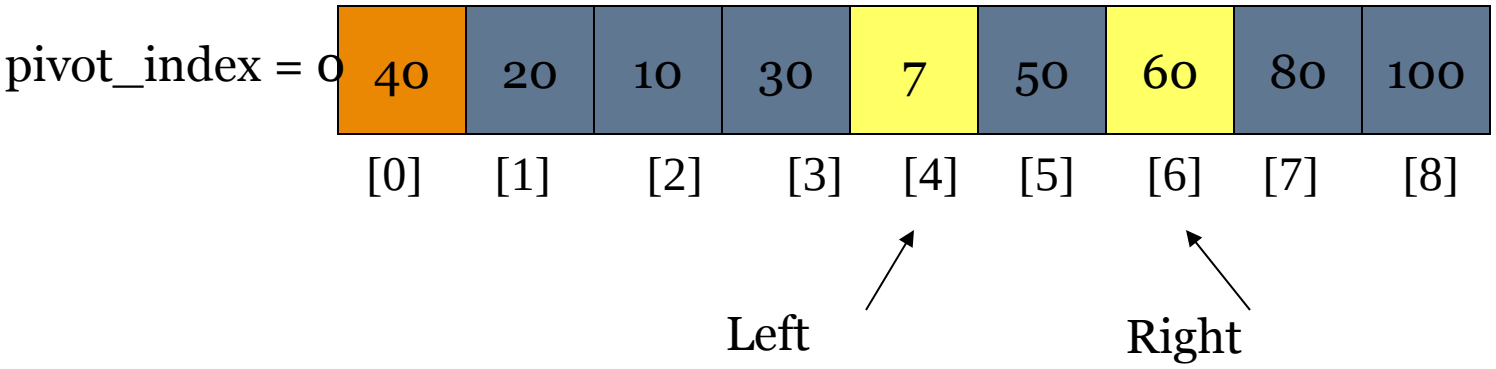
1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right
3. If Left > Right
   swap data[Left] and data[Right]
4. While Right > Left, go to 1.



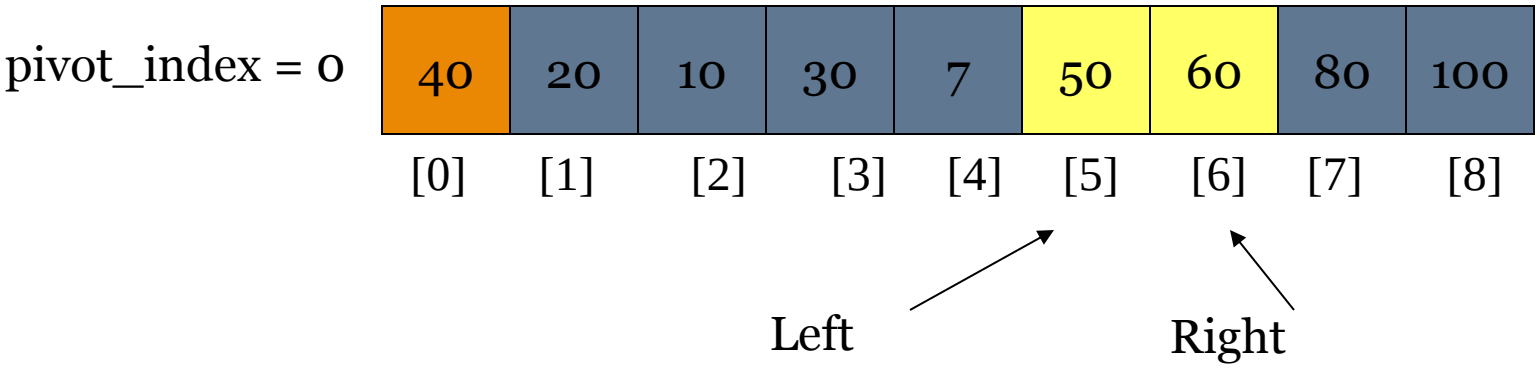
1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right
3. If Left > Right
   swap data[Left] and data[Right]
4. While Right > Left, go to 1.



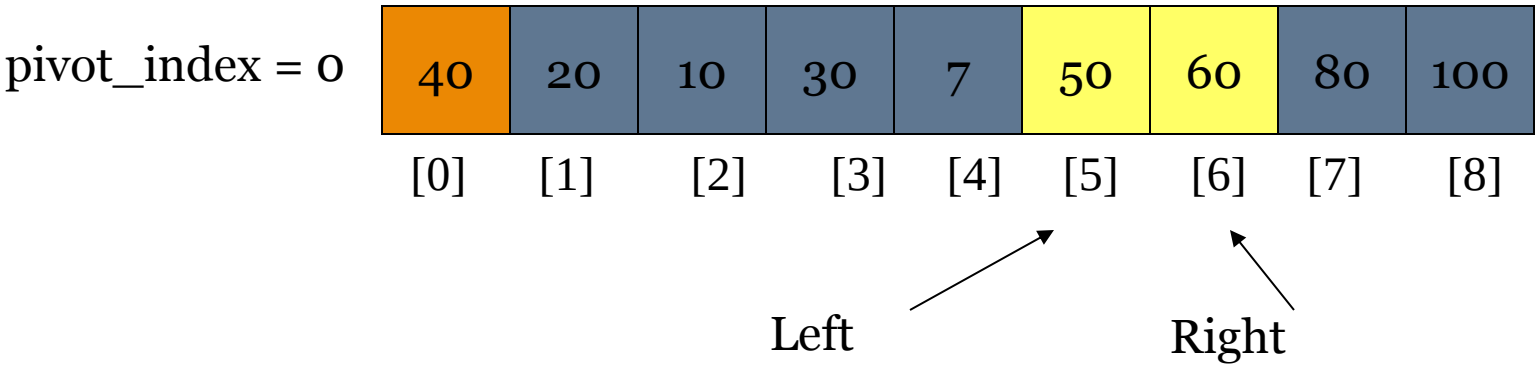
1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right
3. If Left > Right
   swap data[Left] and data[Right]
4. While Right > Left, go to 1.



1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right
3. If Left > Right
   swap data[Left] and data[Right]
4. While Right > Left, go to 1.

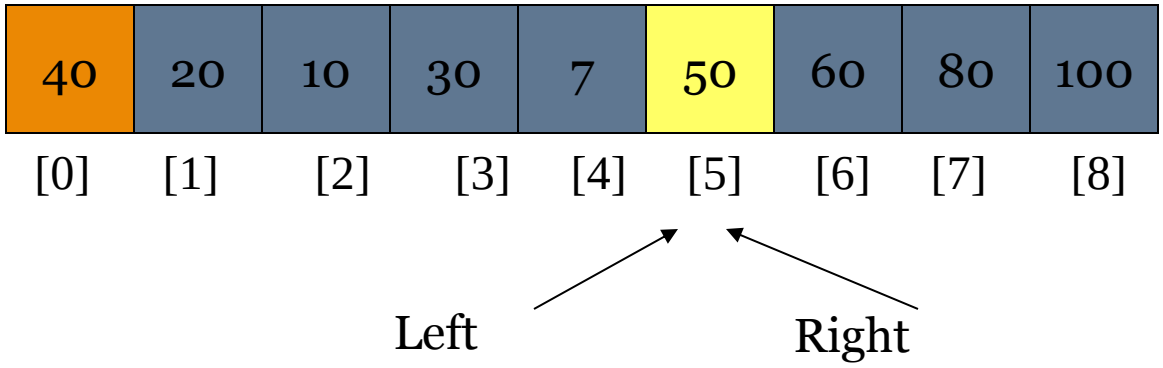


1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right
3. If Left > Right
   swap data[Left] and data[Right]
4. While Right > Left, go to 1.

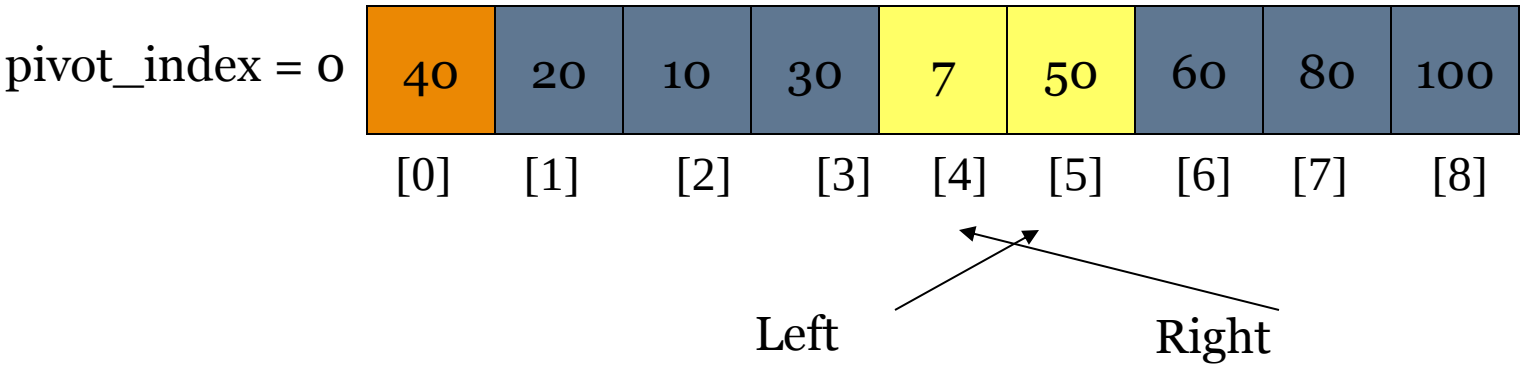


1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right
3. If Left > Right
   swap data[Left] and data[Right]
4. While Right > Left, go to 1.

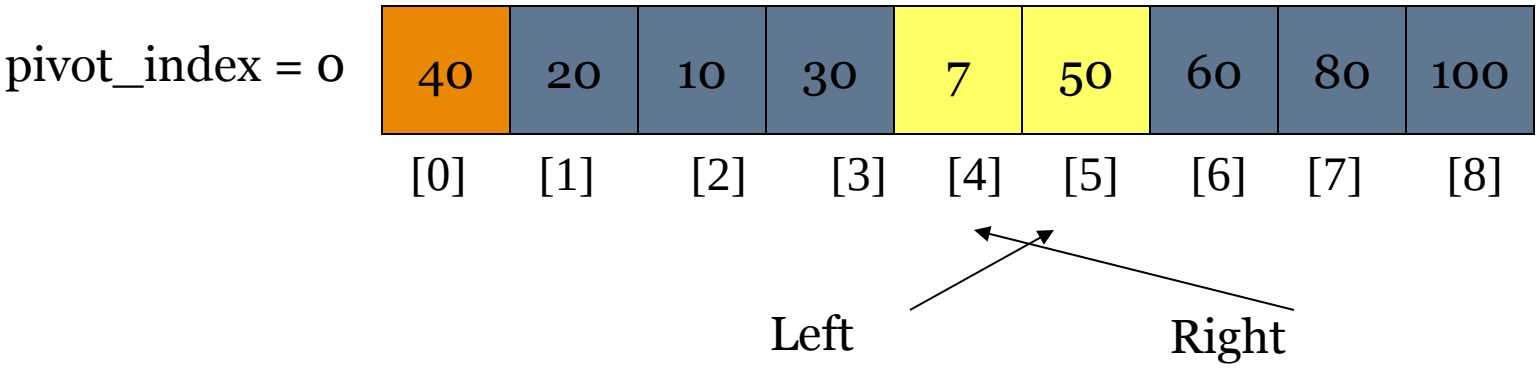
pivot_index = 0



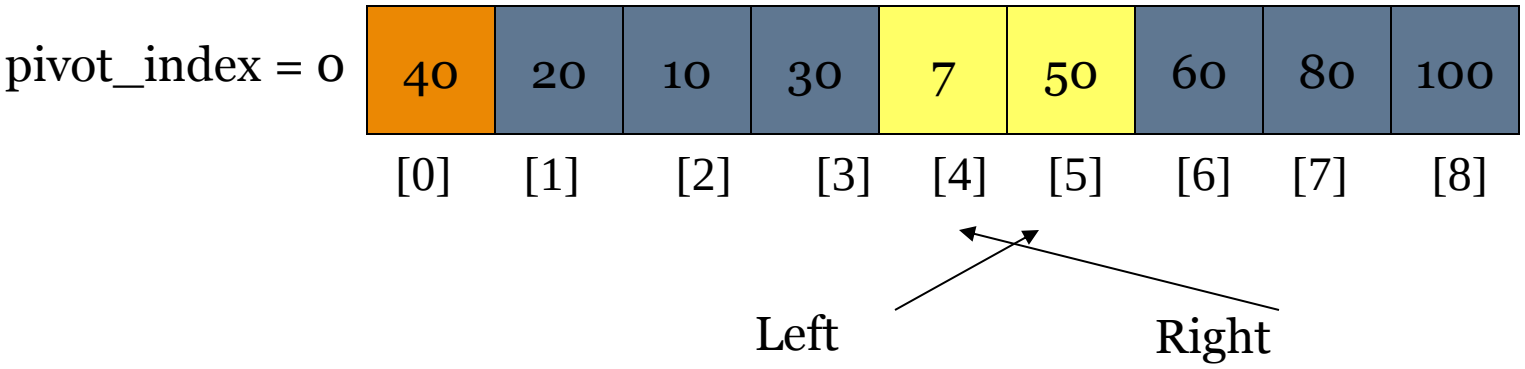
1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right
3. If Left > Right
   swap data[Left] and data[Right]
4. While Right > Left, go to 1.



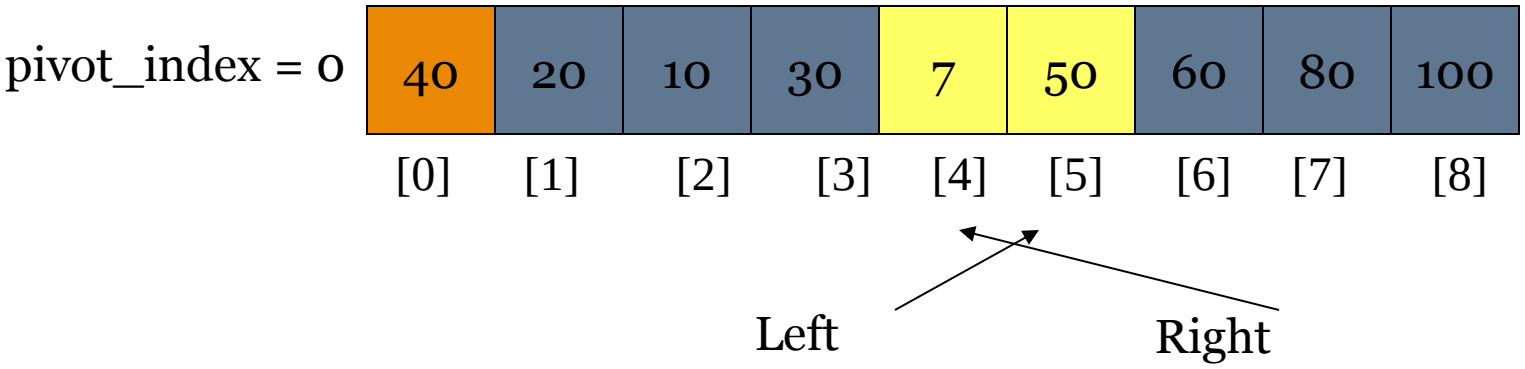
1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right
3. If Left > Right
   swap data[Left] and data[Right]
4. While Right > Left, go to 1.



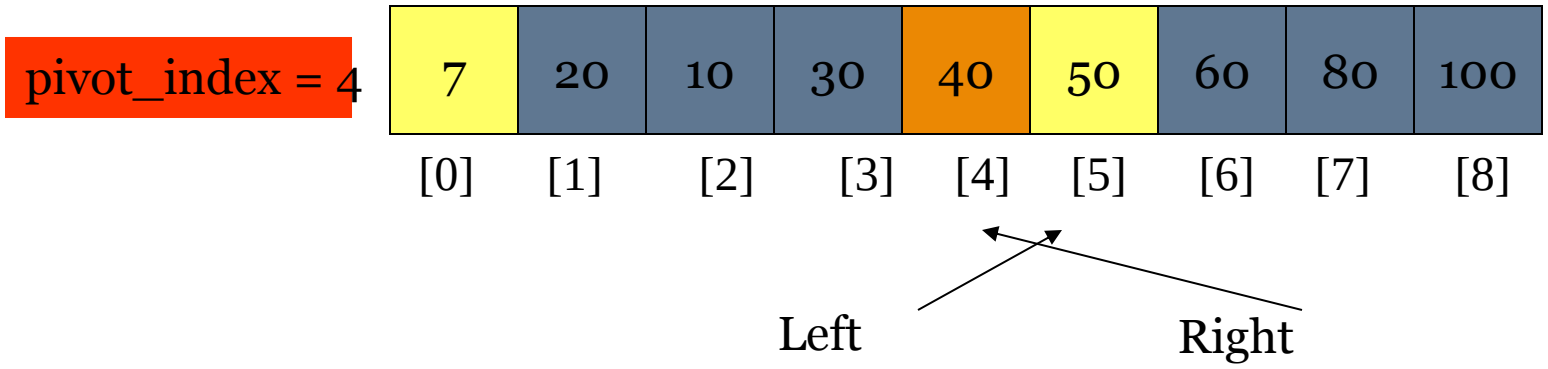
1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right
3. If Left > Right
   swap data[Left] and data[Right]
4. While Right > Left, go to 1.



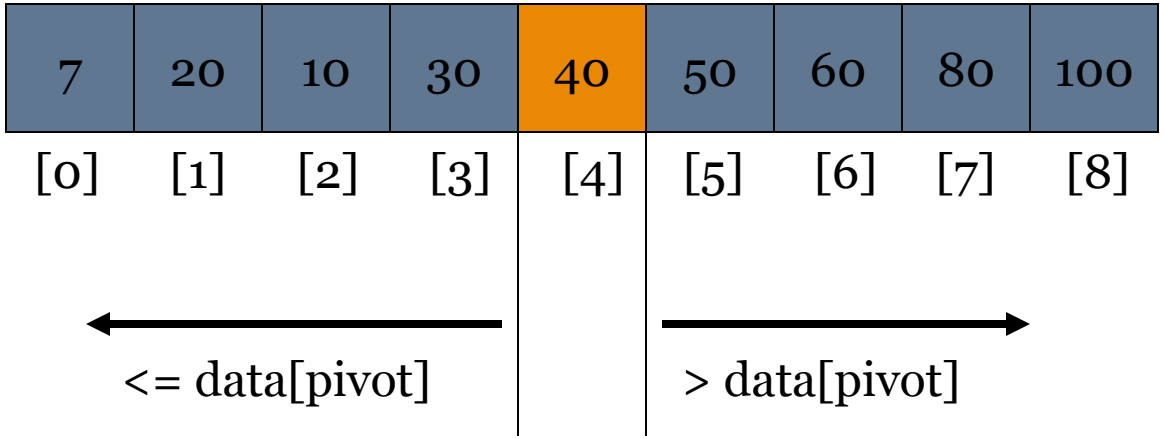
1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right
3. If Left > Right
   swap data[Left] and data[Right]
4. While Right > Left, go to 1.
5. Swap data[Right] and data[pivot_index]



1. While data[Left] <= data[pivot]
   ++Left
2. While data[Right] > data[pivot]
   --Right
3. If Left > Right
   swap data[Left] and data[Right]
4. While Right > Left, go to 1.
5. Swap data[Right] and data[pivot_index]

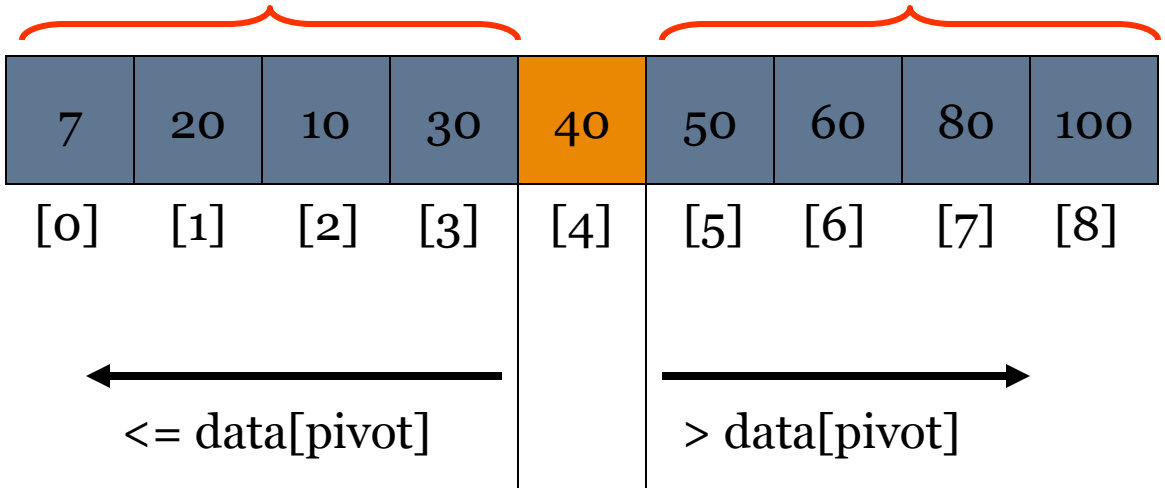


Partition Result



MIT-ADT
UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Recursion: Quicksort Sub-arrays



Quicksort Analysis

- Assume that keys are random, uniformly distributed.
- Best case running time: $O(n \log_2 n)$

MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

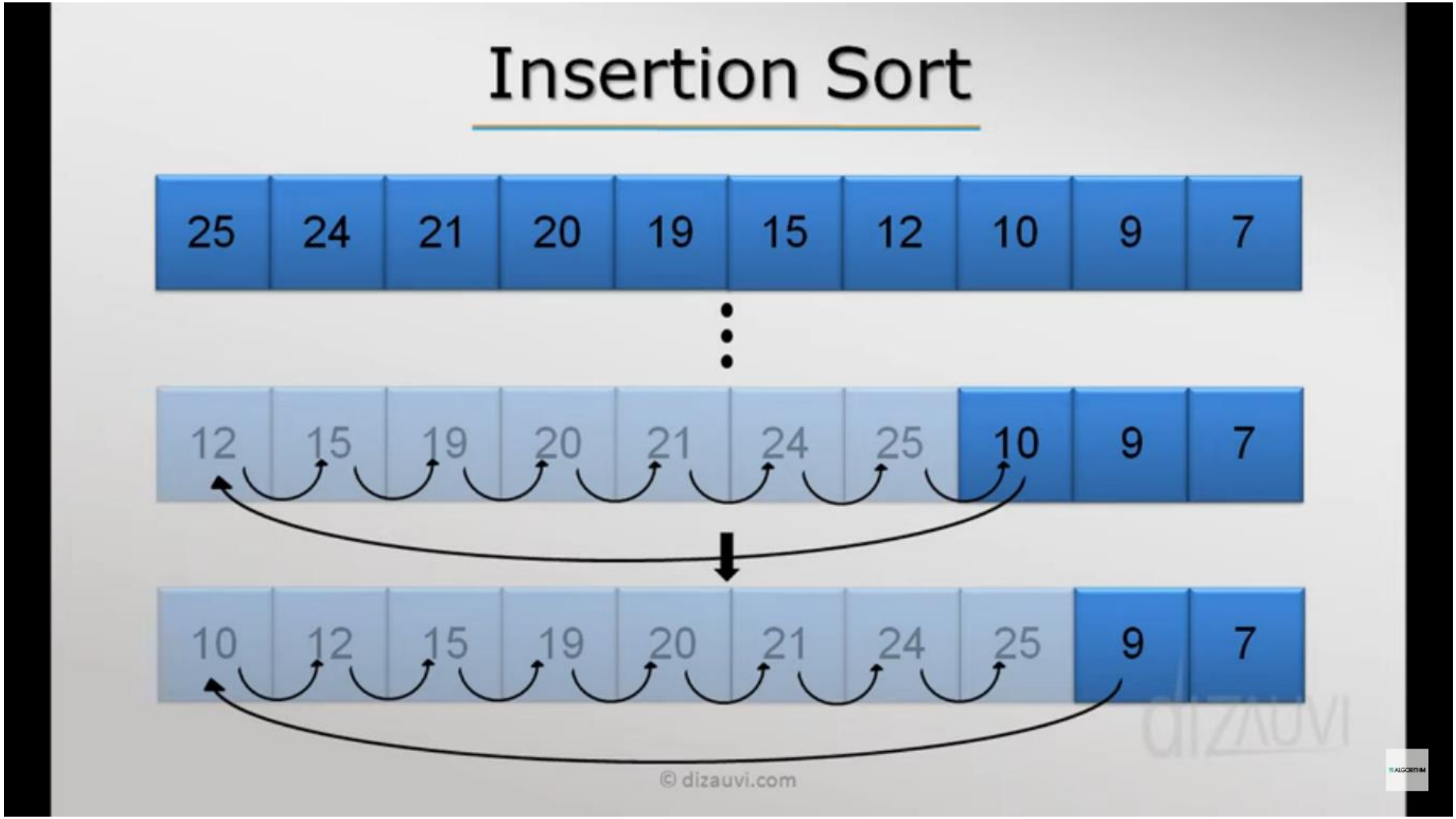
Quicksort Analysis

- Assume that keys are random, uniformly distributed.
- Best case running time: $O(n \log_2 n)$
- Worst case running time?

Quicksort Analysis

- Assume that keys are random, uniformly distributed.
- Best case running time: $O(n \log_2 n)$
- Worst case running time: $O(n^2)$!!!

Shell Sort

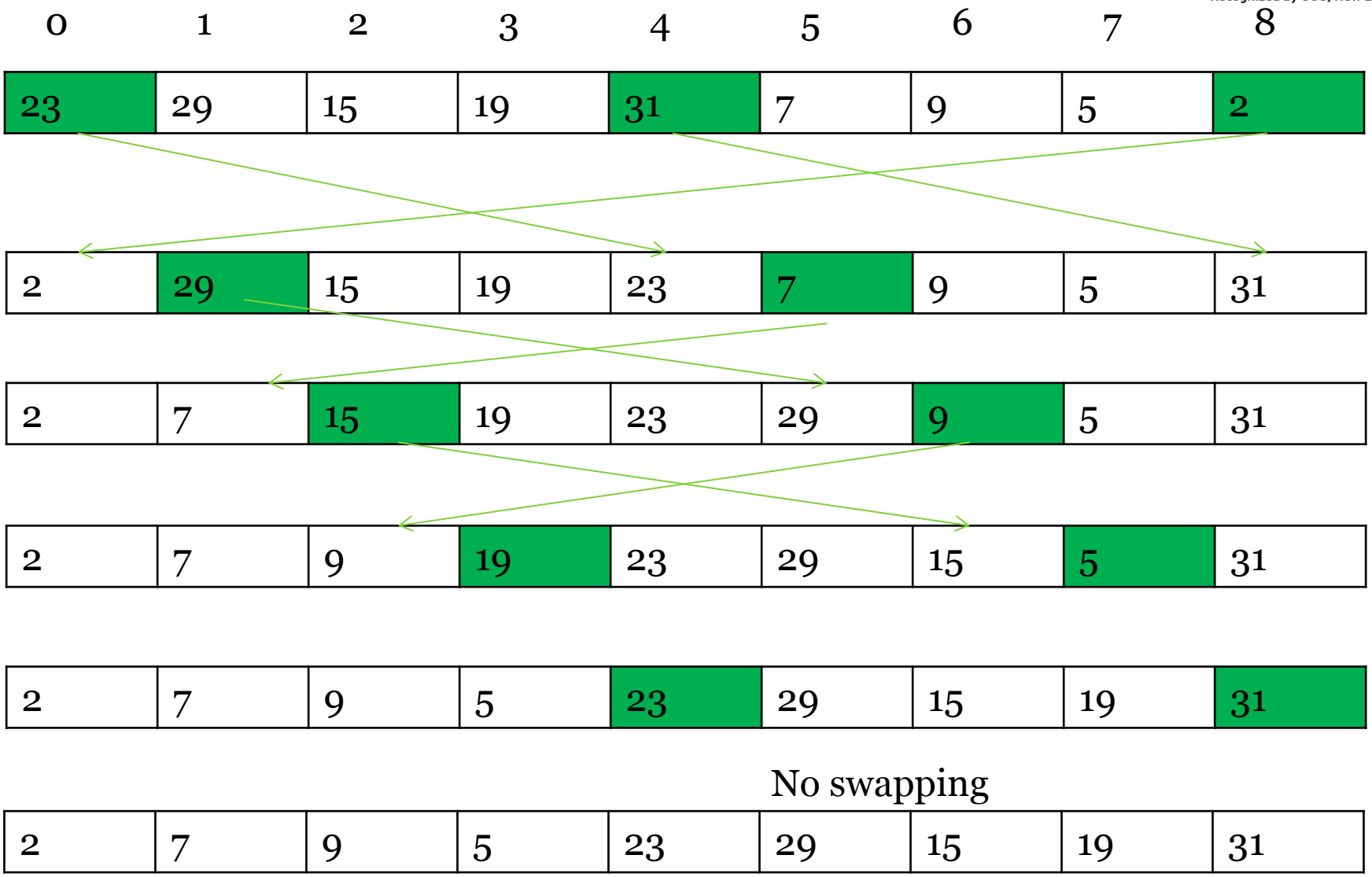

Insertion Sort
25 24 21 20 19 15 12 10 9 7
12 15 19 20 21 24 25 10 9 7
10 12 15 19 20 21 24 25 9 7
© dizauvi.com

Shell Sort

- Shell sort is a sorting algorithm, which is an improved version of insertion sort.
- It makes repetitive use of insertion sort.
- In this technique, the elements at a fixed distance are compared.
- Later, this distance is decremented in the next pass by some value and again the comparisons are made.
- The fixed distance is called as gap. The algorithm begins with the initial gap as n/2, where n is the total number of elements to be sorted.
- Later, in the next pass, the gap is modified as n/4, n/8, and so on till it becomes 1.
- When gap is 1, it becomes an ordinary insertion sort

MIT UNIVERSITY
ART, DESIGN & TECHNOLOGY
MIT-ADT
UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Gap = 8/2 = 4



MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Pass 2

Gap = 4/2 = 2

0	1	2	3	4	5	6	7	8
2	7	9	5	23	29	15	19	31

No swapping

2	7	9	5	23	29	15	19	31

2	5	9	7	23	29	15	19	31

No swapping

2	5	9	7	23	29	15	19	31

If No swapping then no need to compare with previous element with gap value

2	5	9	7	23	29	15	19	31

2	5	9	7	15	29	23	19	31

2	5	9	7	15	19	23	29	31

Pass 3

Gap = 2/2 = 1

0	1	2	3	4	5	6	7	8
2	5	9	7	15	19	23	29	31
2	5	9	7	15	19	23	29	31
2	5	9	7	15	19	23	29	31
2	5	7	9	15	19	23	29	31
2	5	7	9	15	19	23	29	31
2	5	7	9	15	19	23	29	31
2	5	7	9	15	19	23	29	31
2	5	7	9	15	19	23	29	31

Bucket Sort

- IN this sorting algorithm we create buckets and put elements into them.
- Then we apply some sorting algorithm (insertion algorithm) to sort elements in each bucket.
- Finally we take the elements out and oin them to get the sorted result.

MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Bucket Sort

22	45	12	8	10	6	72	81	33	18	50	14

B
0
1
2
3
4
5
6
7
8
9

To sort elements we will take 10 buckets

N = 12
Max = 81
Min = 6

Now we need to find a divider which will be used to Put the elements in the buckets.
divider = ceil ((max+1)/buckets))
= ceil ((81+1)/10)
= ceil(82/10)
= ceil(8.2)
= 9

Bucket Sort

22	45	12	8	10	6	72	81	33	18	50	14

B

0	8 ,6
1	12, 10,14
2	22,18
3	33
4	
5	45, 50
6	
7	
8	72
9	81

put the elements in bucket by dividing each element by 9

Arr[0] = floor(22/9) = 2
Arr[1] = floor(45/9) = 5
Arr[2] = floor(12/9) = 1
Arr[3] = floor(8/9) = 0
Arr[4] = floor(10/9) = 1
Arr[5] = floor(6/9) = 0
Arr[6] = floor(72/9) = 8
Arr[8] = floor(81/9) = 9
Arr[9] = floor(33/9) = 3
Arr[10] = floor(18/9) = 2
Arr[11] = floor(50/9) = 5
Arr[12] = floor(14/9) = 1

N = 12
Max = 81
Min = 6

Bucket Sort

22	45	12	8	10	6	72	81	33	18	50	14

B

0	8 ,6
1	12, 10, 14
2	22,18
3	33
4	
5	45, 50
6	
7	
8	72
9	81

Now we will sort each bucket using insertion sort

8, 6
is 8 > 6 ----- yes
Swap
6,8

N = 12
Max = 81
Min = 6

Bucket Sort

22	45	12	8	10	6	72	81	33	18	50	14

B

0	6, 8
1	12, 10,14
2	22,18
3	33
4	
5	45, 50
6	
7	
8	72
9	81

Now we will sort each bucket using insertion sort

12, 10,14
is 12 > 10 ----- yes
Swap
10,12,14

N = 12
Max = 81
Min = 6

Bucket Sort

22	45	12	8	10	6	72	81	33	18	50	14

B

0	6, 8
1	10, 12,14
2	22, 18
3	33
4	
5	45, 50
6	
7	
8	72
9	81

Now we will sort each bucket using insertion sort

22, 18
is 22 > 18 ----- yes
Swap
18,22

N = 12
Max = 81
Min = 6

Bucket Sort

22	45	12	8	10	6	72	81	33	18	50	14

B

0	6, 8
1	10, 12,14
2	18,22
3	33
4	
5	45, 50
6	
7	
8	72
9	81

Now we will sort each bucket using insertion sort

Bucket 3 conation one element only

N = 12
Max = 81
Min = 6

MIT-ADT UNIVERSITY
PUNE, INDIA
A Leap Towards The World Class Education
Approved by Govt. of Maharashtra
Recognized by UGC, New Delhi

Bucket Sort

22	45	12	8	10	6	72	81	33	18	50	14

B

0	6, 8
1	10, 12,14
2	18,22
3	33
4	
5	45, 50
6	
7	
8	72
9	81

Now we will sort each bucket using insertion sort

45, 50
is 45 > 50 ----- no

N = 12
Max = 81
Min = 6

Bucket 8 and 9 conation one element only

6	8	10	12	14	18	22	33	45	50	72	81

Radix Sort

- Radix sort is a generalization of bucket sort and works in three steps:

1. Distribute all elements into m buckets. Here m is a suitable integer, for example, to sort decimal numbers with radix 10. We take 10 buckets numbered as 0, 1, 2, …, 9. For sorting strings, we may need 26 buckets, and so on.

2. Sort each bucket individually.

3. Finally, combine all buckets.

Pass 1:

Assume the input array is: 010, 021, 017, 034, 044, 011, 654, 123

(Maximum number is 654 which contain 3 digits so make all number of length 3)

Step 1: Put all elements into buckets according to the **one's digit** (least significant digit).

Step 2: Take out elements from bucket

Step 3: 010, 021, 011,123, 034, 044, 654, 017

0	010
1	021, 011
2	
3	123
4	034, 044, 654
5	
6	
7	017
8	
9	

Pass 2:

The input array is: 010, 021, 011,123, 034, 044, 654, 017

Step 1: Put all elements into buckets according to the **ten's digit**.

Step 2: Take out elements from bucket

Step 3: 010, 011, 017, 021, 123, 034, 044, 654

0	
1	010, 011, 017
2	021, 123
3	034
4	044
5	654
6	
7	
8	
9	

Pass 3:

The input array is: 010, 011, 017, 021, 123, 034, 044, 654

Step 1: Put all elements into buckets according to the **hundred's digit**.

Step 2: Take out elements from bucket

Step 3: 010, 011, 017, 021, 034, 044, 123, 654

0	010, 011,017, 021,034, 044
1	123
2	
3	
4	
5	
6	654
7	
8	
9	

Complexity radix sort

- O(d*(n+b))
- Where d=digits in max number
- n=total numbers to be sorted
- b=bases (0 to 9)

Which algorithm to use		
Algorithm Name	Description	When to use
Merge Sort	First, the values split until single-element groups And sorts sub-group while gradually merging	Used for sorting linked list
Bubble Sort	Compares adjacent element pairs and swaps if they are not ascending	Good for understanding sorting and sort small data set
Selection Sort	For each location, it checks whether it's less than the elements to its right	Beneficial to learn sorting algorithms
Quick Sort	Repeatedly splits elements based on chosen pivot number	Used for sorting arrays and medium-sized data set
Heap Sort	Proceeds by heapify, swap and insert	Ideal for big data sets
Insertion Sort	Sorts by comparing, moving, and placing	Used to sort small data set and check already sorted list
Intro Sort	Chooses between other sorting algorithm based on data set	Uses insertion sort to sort small dataset, quicksort when the pivot is the median and heap sort for sorting large dataset

Sorting Algorithm	Advantage	Disadvantage
Selection Sort	Performs well on a small list and no additional temporary storage is required.	Poor efficiency when dealing with a huge list of items
Bubble Sort	Popular and Easy to implement	Does not deal well with a list containing a huge number of items
Insertion Sort	Simple, performs well on a small list and minimal space is required	Does not deal well with the huge list
Merge Sort	It can be applied to files of any size.	Requires extra space =N
Quick Sort	Best sorting algorithm and deals very well with large list	Disadvantage with the worst case efficiency

iq.opengenus.org

Sorting Algorithms	Time Complexity			Space Complexity
	Best Case	Average Case	Worst Case	Worst Case
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Quick Sort	O(nlogn)	O(nlogn)	O(n^2)	O(n)
Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(1)
Counting Sort	O(n + k)	O(n + k)	O(n + k)	O(k)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n + k)
Bucket Sort	O(n + k)	O(n + k)	O(n^2)	O(n)