

UNIT 5

Introduction to Searching
and Searching Techniques:
Sequential Search
Binary Search



Searching :

The process of locating target data.

Defination :

In computer science, a search algorithm is any algorithm which solves the search problem, namely, to retrieve information stored within some data structure, or calculated in the search space of a problem domain.

The Sequential Search

- When data items are stored in a collection such as a list, we say that they have a linear or sequential relationship.
- Each data item is stored in a position relative to the others.
- Starting at the first item in the list, we simply move from item to item, following the underlying sequential ordering until we either find what we are looking for or run out of items.
- If we run out of items, we have discovered that the item we were searching for was not present.

```
1 def sequentialSearch(alist, item):
2     pos = 0
3     found = False
4
5     while pos < len(alist) and not found:
6         if alist[pos] == item:
7             found = True
8         else:
9             pos = pos+1
10
11     return found
12
13 testlist = [1, 2, 32, 8, 17, 19, 42, 13, 0]
14 print(sequentialSearch(testlist, 3))
15 print(sequentialSearch(testlist, 13))
```

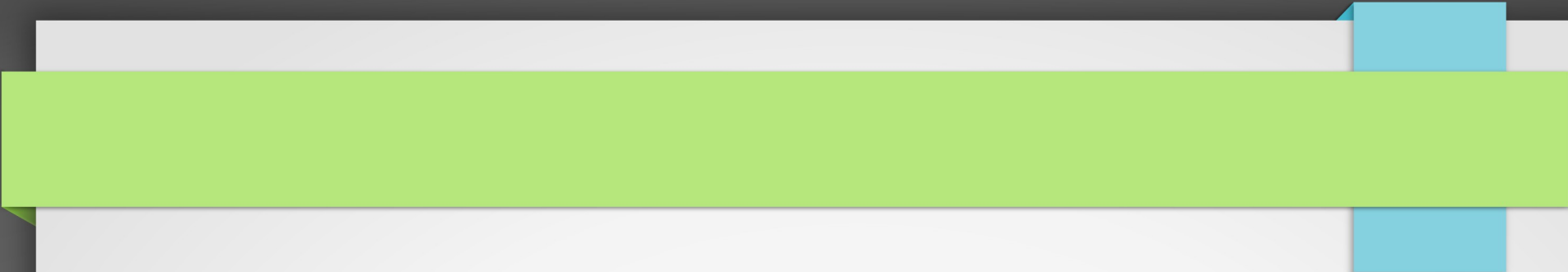
$O(n)$. *Table 1* summarizes these results.

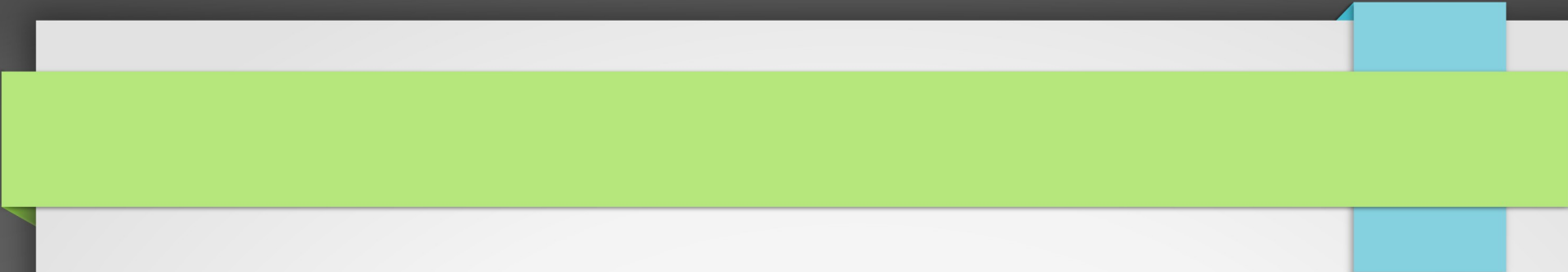
Table 1: Comparisons Used in a Sequential Search of an Unordered List

Case	Best Case	Worst Case	Average Case
item is present	1	n	$\frac{n}{2}$
item is not present	n	n	n

Binary Search

- In computer science, a binary search or half-interval search algorithm finds the position of a target value within a sorted array.
- The binary search algorithm can be classified as a dichotomic divide and conquer search algorithm and executes in logarithmic time.
-
- 23 45 6 7 89 100
- 100
- 6 7 23 45 89 100
- $\text{Mid} = (0 + 5)/2$ 45 100 > 45
- $\text{Mid} = 89$ 100 > 89
-

- 
- The binary search algorithm begins by comparing the target value to value of the middle element of the sorted array.
 - If the target value is equal to the middle element's value, the position is returned.
 - If the target value is smaller, the search continues on the lower half of the array, or if the target value is larger, the search continues on the upper half of the array.
 - This process continues until the element is found and its position is returned, or there are no more elements left to search for in the array and a "not found" indicator is returned.

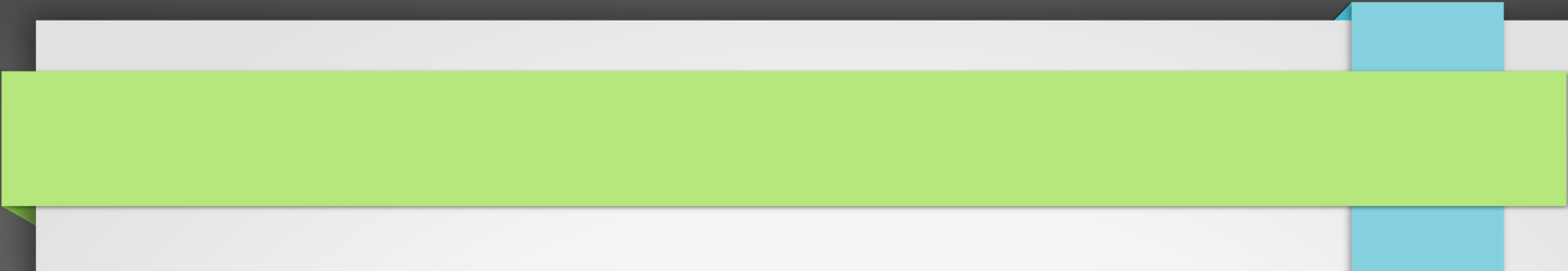


It is possible to take greater advantage of the ordered list if we are clever with our comparisons.

In the sequential search, when we compare against the first item, there are at most $n-1$ more items to look through if the first item is not what we are looking for.

Instead of searching the list in sequence, a **binary search** will start by examining the middle item. If that item is the one we are searching for, we are done. If it is not the correct item, we can use the ordered nature of the list to eliminate half of the remaining items.

If the item we are searching for is greater than the middle item, we know that the entire lower half of the list as well as the middle item can be eliminated from further consideration. The item, if it is in the list, must be in the upper half.



We can then repeat the process with the upper half. Start at the middle item and compare it against what we are looking for. Again, we either find it or split the list in half, therefore eliminating another large part of our possible search space.

```

5
6     while first<=last and not found:
7         midpoint = (first + last)//2
8         if alist[midpoint] == item:
9             found = True
10        else:
11            if item < alist[midpoint]:
12                last = midpoint-1
13            else:
14                first = midpoint+1
15
16        return found
17
18 testlist = [0, 1, 2, 8, 13, 17, 19, 32, 42,]
19 print(binarySearch(testlist, 3))
20 print(binarySearch(testlist, 13))

```

Sorting

- Introduction to Sorting
- Sorting Techniques:
- Bubble sort
- Selection sort
- Insertion sort
- Merge sort
- Quick Sort

Introduction to Sorting

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.

Bubble Sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Position wise comparison

Adjacent node swapping

$n-1$ steps

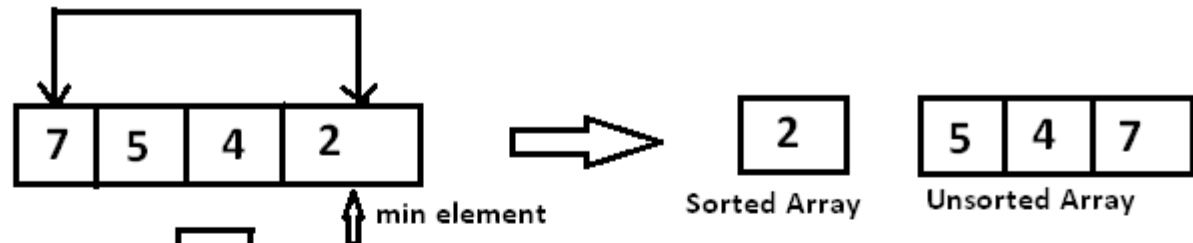
At the end of each step u will get targets element and that element going to secure position from end

$i = 0$	j	0	1	2	3	4	5	6	7
	0	5	3	1	9	8	2	4	7
	1	3	5	1	9	8	2	4	7
	2	3	1	5	9	8	2	4	7
	3	3	1	5	9	8	2	4	7
	4	3	1	5	8	9	2	4	7
	5	3	1	5	8	2	9	4	7
	6	3	1	5	8	2	4	9	7
$i = 1$	0	3	1	5	8	2	4	7	9
	1	1	3	5	8	2	4	7	
	2	1	3	5	8	2	4	7	
	3	1	3	5	8	2	4	7	
	4	1	3	5	2	8	4	7	
	5	1	3	5	2	4	8	7	
$i = 2$	0	1	3	5	2	4	7	8	
	1	1	3	5	2	4	7		
	2	1	3	5	2	4	7		
	3	1	3	2	5	4	7		
	4	1	3	2	4	5	7		
$i = 3$	0	1	3	2	4	5	7		
	1	1	3	2	4	5			
	2	1	2	3	4	5			
	3	1	2	3	4	5			
$i = 4$	0	1	2	3	4	5			
	1	1	2	3	4				
	2	1	2	3	4				
$i = 5$	0	1	2	3	4				
	1	1	2	3					
$i = 6$	0	1	2	3					
		1	2						

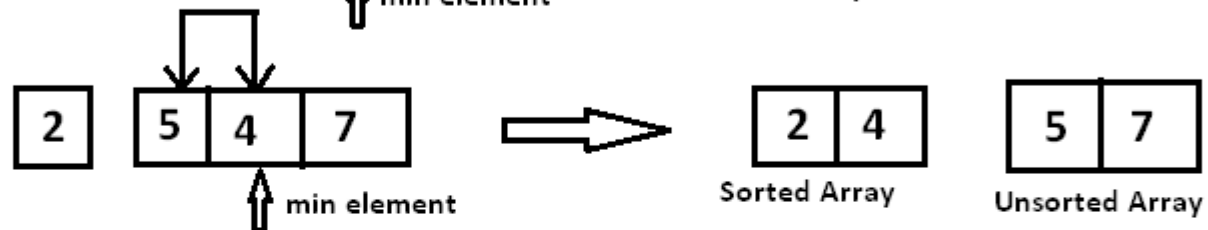
Selection Sort

- The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.
 - 1) The subarray which is already sorted.
 - 2) Remaining subarray which is unsorted.
- In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

STEP 1.



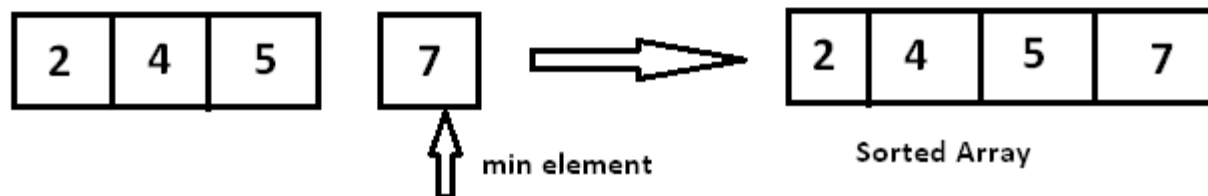
STEP 2.



STEP 3.



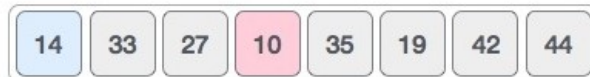
STEP 4.



Consider the following depicted array as an example.



For the first position in the sorted list, the whole list is scanned sequentially. The first position where 14 is stored presently, we search the whole list and find that 10 is the lowest value.



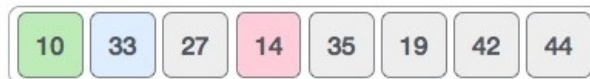
So we replace 14 with 10. After one iteration 10, which happens to be the minimum value in the list, appears in the first position of the sorted list.



For the second position, where 33 is residing, we start scanning the rest of the list in a linear manner.

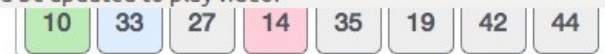


We find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.



After two iterations, two least values are positioned at the beginning in a sorted manner.

t supported, and should be updated to play video.

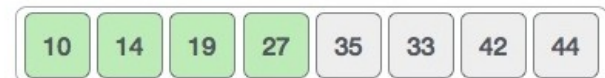


After two iterations, two least values are positioned at the beginning in a sorted manner.



The same process is applied to the rest of the items in the array.

Following is a pictorial depiction of the entire sorting process –



com/data_structures_algorithms/selection_sort_algorithm

Search

t supported, and should be updated to play video.

10 14 19 27 35 33 42 44

10 14 19 27 35 33 42 44

10 14 19 27 33 35 42 44

10 14 19 27 33 35 42 44

Now, let us learn some programming aspects of selection sort.

Algorithm

- Step 1 - Set MIN to location 0
- Step 2 - Search the minimum element in the list
- Step 3 - Swap with value at location MIN
- Step 4 - Increment MIN to point to next element
- Step 5 - Repeat until list is sorted

Pseudocode

```
procedure selection sort
  list : array of items
  n    : size of list

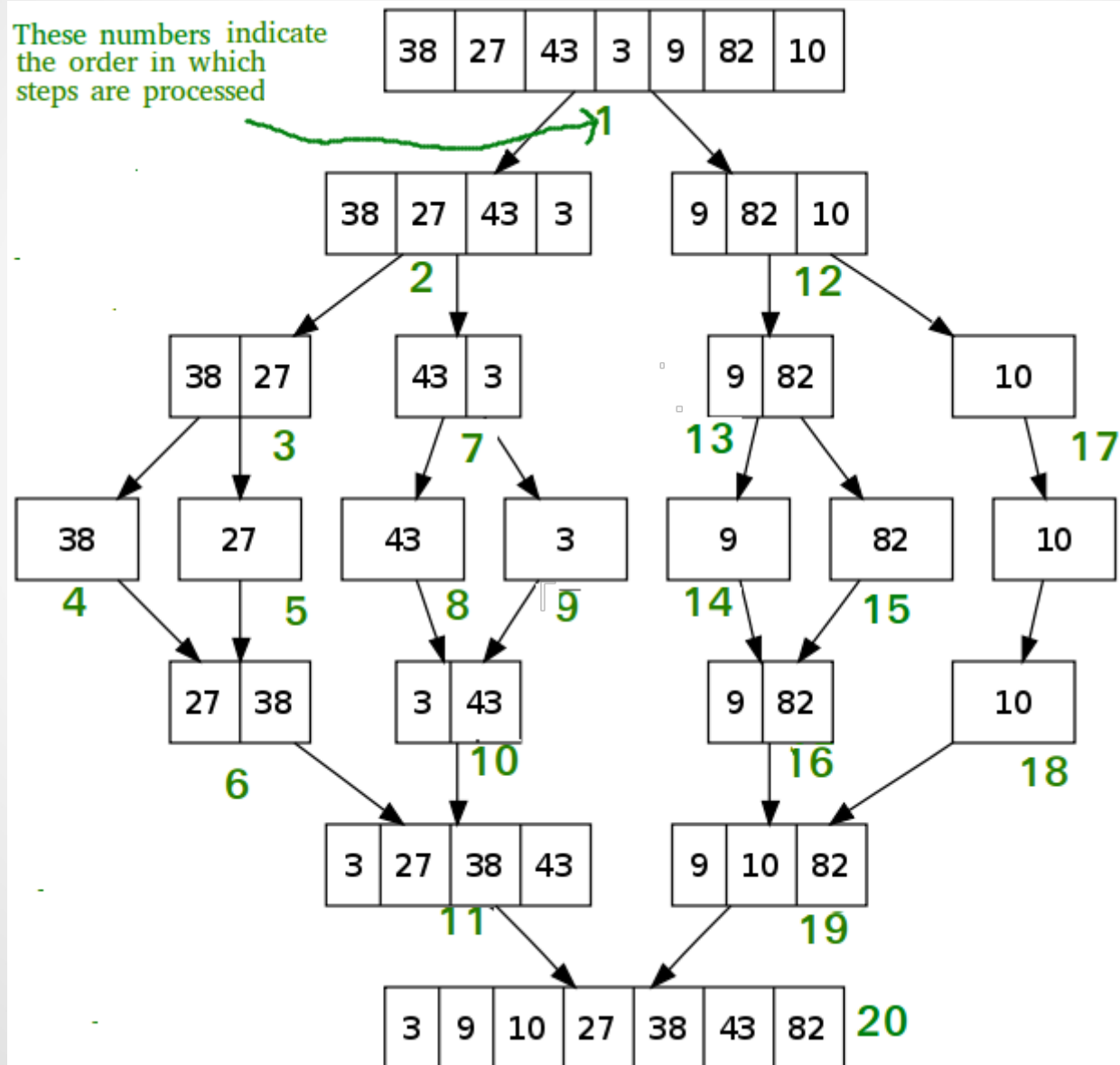
  for i = 1 to n - 1
    /* set current element as minimum */
    min = i

    /* check the element to be minimum */
```

Merge Sort

Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves.

These numbers indicate the order in which steps are processed



Quick Sort

Like Merge Sort, Quick Sort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

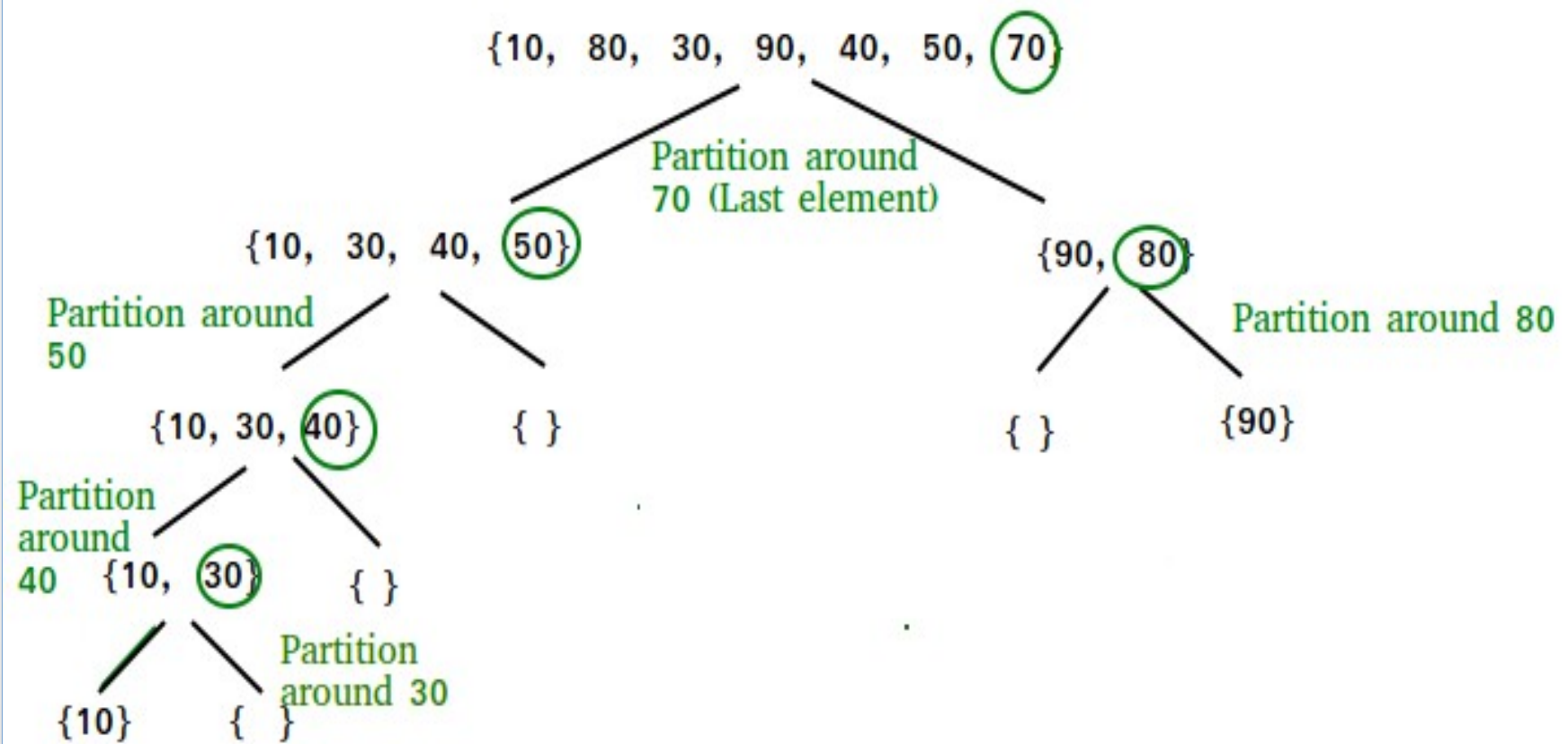
The key process in quick Sort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Short view of logic.

```
/* low --> Starting index,
   high --> Ending index
   Pi --> pivot element
   Arr --> Array to be sorted*/
quickSort(arr[], low, high)
{
    if (low < high)
    {

        /* pi is partitioning index, arr[pi] is now
        at right place */
        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1); // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
```



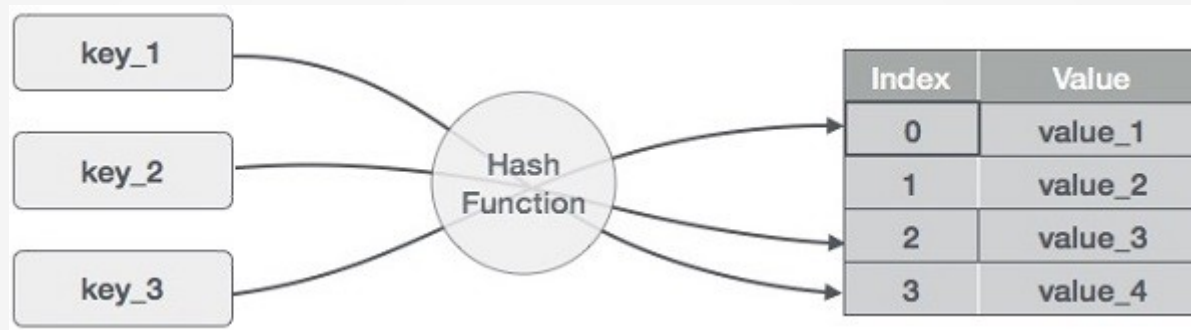
Hashing

- Hash Table: It is a data structure which stores data in an associative manner. In a hash table, data is stored in an array format, where each data value has its own unique index value. Access of data becomes very fast if we know the index of the desired data.
-
- Thus, it becomes a data structure in which insertion and search operations are very fast irrespective of the size of the data. Hash Table uses an array as a storage medium and uses hash technique to generate an index where an element is to be inserted or is to be located from.

Hashing

- Hashing is a technique to convert a range of key values into a range of indexes of an array.
- We're going to use modulo operator to get a range of key values.
- Consider an example of hash table of size 20, and the following items are to be stored. Item are in the (key,value) format.

- Hash table 20 size 0 to 19
- (1,20)
- (2,70)
- (42,80)
- (4,25)
- (12,44)
- (14,32)
- (17,11)
- (13,78)
- (37,98)



Sr.No.	Key	Hash	Array Index
1	1	(1,20) $1 \% 20 = 1$	1
2	2	(2,70) $2 \% 20 = 2$	2
3	42	(42,80) $42 \% 20 = 2$	2
4	4	(4,25) $4 \% 20 = 4$	4
5	12	(12,44) $12 \% 20 = 12$	12
6	14	(14,32) $14 \% 20 = 14$	14
7	17	(17,11) $17 \% 20 = 17$	17
8	13	(13,78) $13 \% 20 = 13$	13
9	37	(37,98) $37 \% 20 = 17$	17

Linear Probing

- As we can see, it may happen that the hashing technique is used to create an already used index of the array. In such a case, we can search the next empty location in the array by looking into the next cell until we find an empty cell. This technique is called linear probing.

Sr.No.	Key	Hash	Array Index	After Linear Probing, Array Index
1	1	$1 \% 20 = 1$	1	1
2	2	$2 \% 20 = 2$	2	2
3	42	$42 \% 20 = 2$	2	3
4	4	$4 \% 20 = 4$	4	4
5	12	$12 \% 20 = 12$	12	12
6	14	$14 \% 20 = 14$	14	14
7	17	$17 \% 20 = 17$	17	17
8	13	$13 \% 20 = 13$	13	13
9	37	$37 \% 20 = 17$	17	18