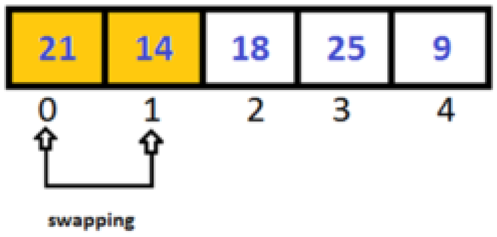
Bubble Sort Algorithm

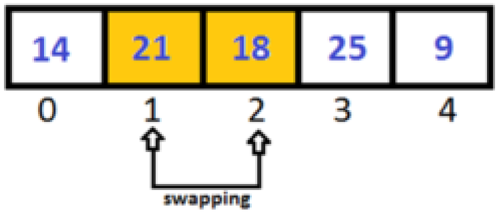
- Bubble sort algorithm starts by comparing the first two elements of an list and swapping if necessary,
- i.e., if you want to sort the elements of list in ascending order and if the first element is greater than second then, you need to swap the elements but, if the first element is smaller than second, you mustn't swap the element.
- Then, again second and third elements are compared and swapped if it is necessary and this process go on until last and second last element is compared and swapped.
- This completes the first step of bubble sort.

Example

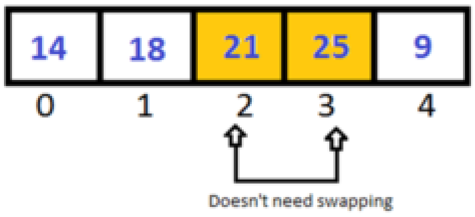
Let's consider we have an list:
21,14,18,25,9



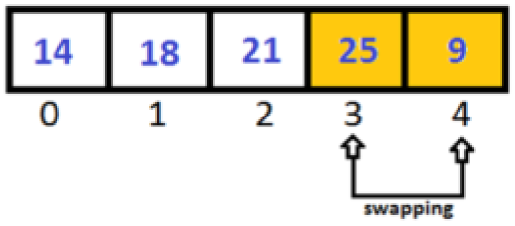
Pass-1:
As we know 14 is less than 21 so swap them and pick next two elements.



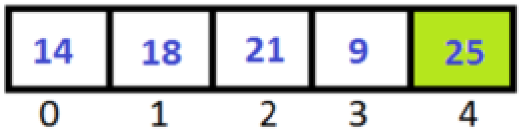
Again 18 is less than 21, so we have to swap them, and pick next two elements.



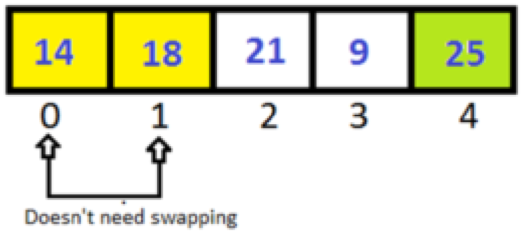
25 is greater than 21 so we doesn't need to swap them, now pick next two elements

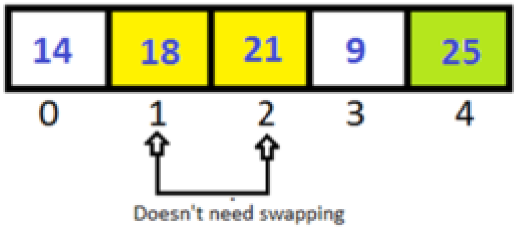


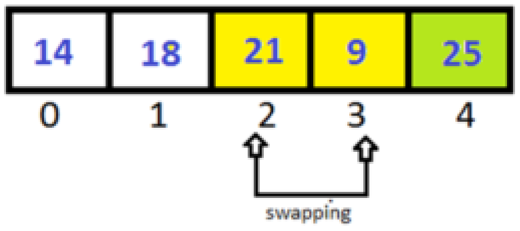
9 is less than 25, swap them.

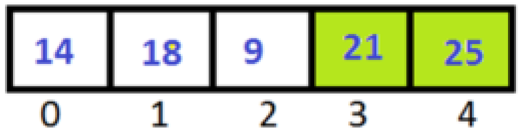


So here we have been traveling list once and the largest number is placed at last index. As we mentioned above that to sort the entire list we have to travel list (n-1) times, where n is the length of list.
So we name it **Pass-1.**
Let's start Pass-2.
Pass-2:
This time we doesn't need to check the last index, because it is sorted after Pass-1.



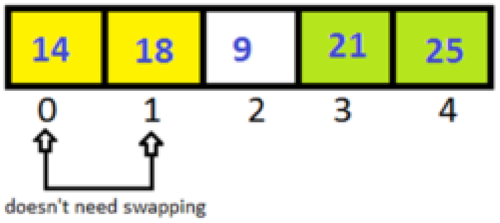


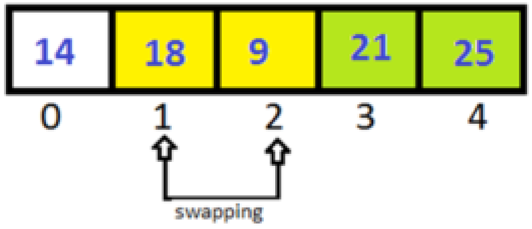


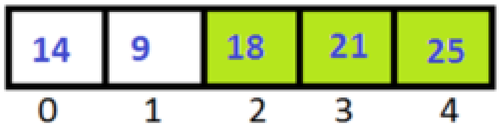


After completing Pass-2 the last two elements have been sorted. So in Pass-3 we don't have to check them.

Pass-3:

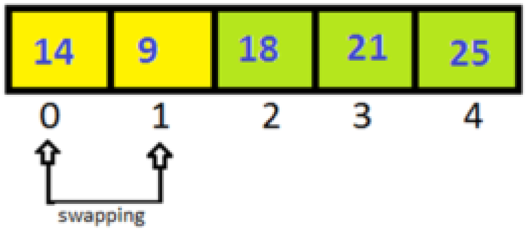


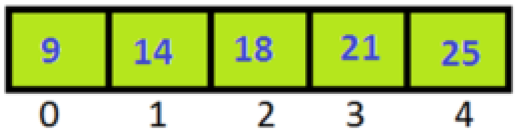




After completing Pass-3 , the last 3 elements of list are in sorted order.

Pass-4:





We have been traveled list (n-1) times. List has been sorted.

Python code:

```
L=[21,14,18,25,9]
n=len(L)
# Traverse through all list elements
for i in range(n):
     # Last i elements are already in place
    for j in range(0,n-i-1):
        # traverse the list from 0 to n-i-1
        # Swap if the element found is greater than the next element

        if L[j]>L[j+1]:
            temp=L[j]        #L[j],L[j+1]=L[j+1],L[j] in single line swap
            L[j]=L[j+1]
            L[j+1]=temp
print "sorted list", L
```

SELECTION SORT

Selection sort is one of the simplest sorting algorithms.

We follow the following steps to perform selection sort:

1. Start from the first element in the list and search for the smallest element in the list.

2. Swap the first element with the smallest element of the list.

3. Take a sublist (excluding the first element of the list as it is at its place) and search for the smallest number in the sublist (second smallest number of the entire list) and swap it with the first element of the list (second element of the entire list).

4. Repeat the steps 2 and 3 with new sublists until the list gets sorted.

Working of selection sort:

Initial list

16	19	11	15	10	12	14

Pass -1:

16	19	11	15	10	12	14
10	19	11	15	16	12	14

Pass -2:

10	19	11	15	16	12	14
10	11	19	15	16	12	14

Pass -3:

10	11	19	15	16	12	14
10	11	12	15	16	19	14

Pass -4:

10	11	12	15	16	19	14
10	11	12	14	16	19	15

Pass -5:

10	11	12	14	16	19	15
10	11	12	14	15	19	16

Pass -6:

10	11	12	14	15	19	16
10	11	12	14	15	16	19

Final list

10	11	12	14	15	16	19

Python Code:

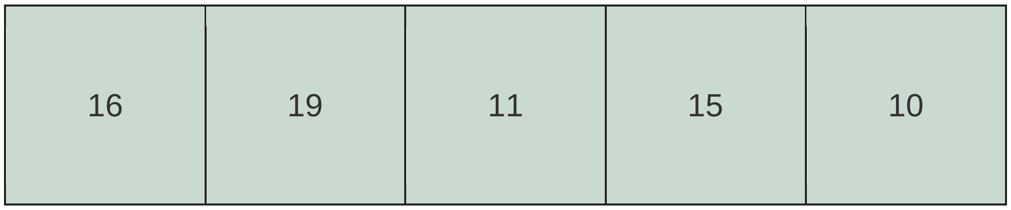
```
# Python program for implementation of Selection Sort
A = [16,19,11,15,10,12,14]
print "Unsorted list ",A
# Traverse through all list elements
for i in range(len(A)):
     # Find the minimum element in remaining  unsorted list
    min_idx = i
    for j in range(i+1, len(A)):
        if A[min_idx] > A[j]:
            min_idx = j

    # Swap the found minimum element with  the first element
    A[i], A[min_idx] = A[min_idx], A[i]

print "Sorted list ",A
```

Insertion Sort

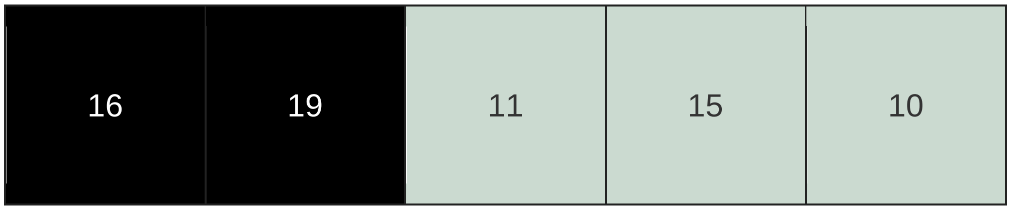
We follow the following steps to perform insertion sort:

Step 1 − If it is the first element, it is already sorted.
Step 2 − Pick next element
Step 3 − Compare with all elements in the sorted sub-list it means previous elements in list.
Step 4 − Shift all the elements in the sorted sub-list that is greater than the value to be sorted
Step 5 − Insert the value
Step 6 − Repeat until list is sorted

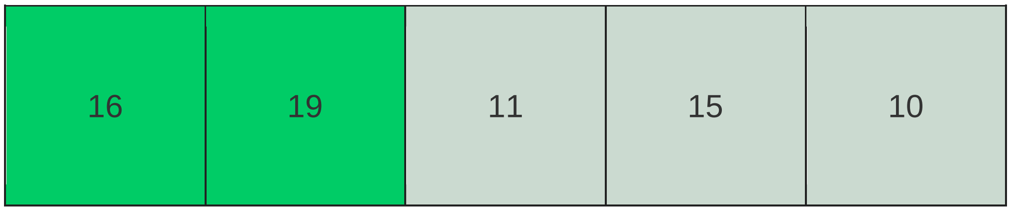
Initial list



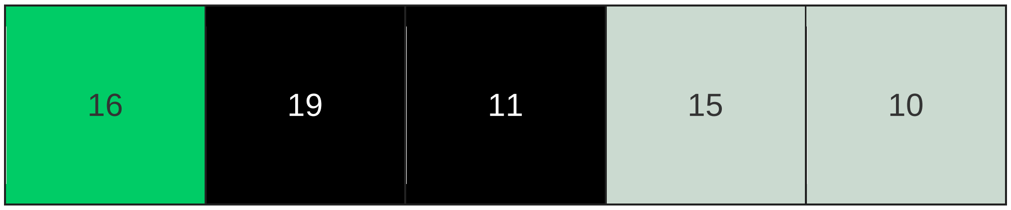
Pass -1:



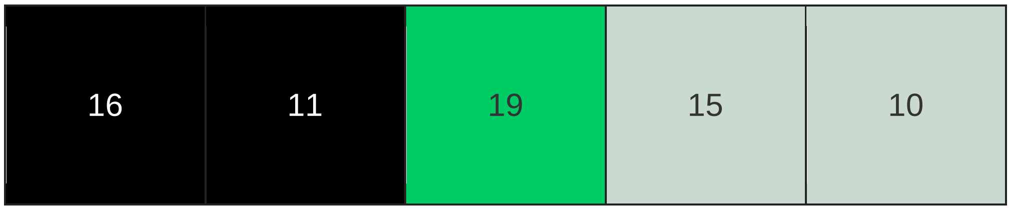
No swapping –



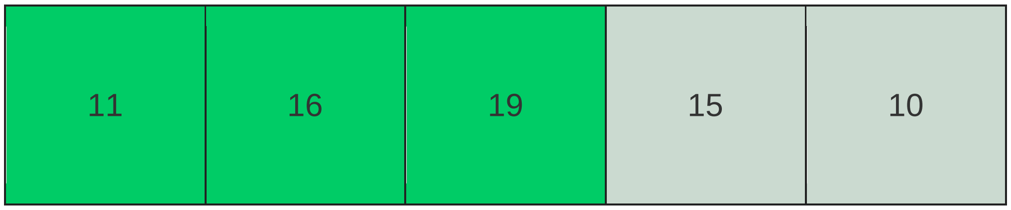
Pass -2:



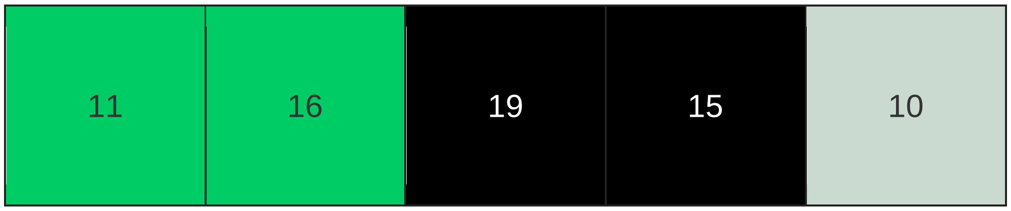
Swap



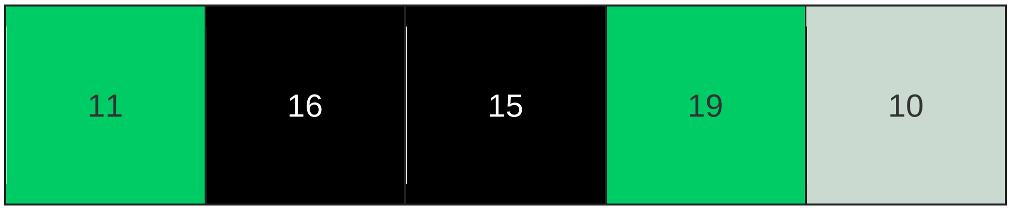
Swap



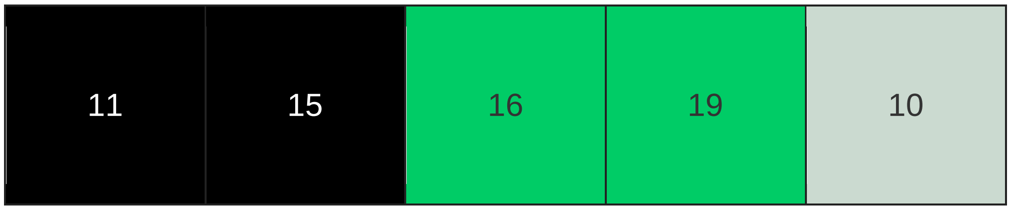
Pass -3:



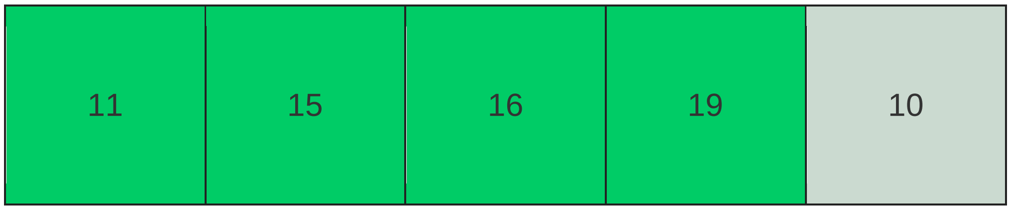
Swap



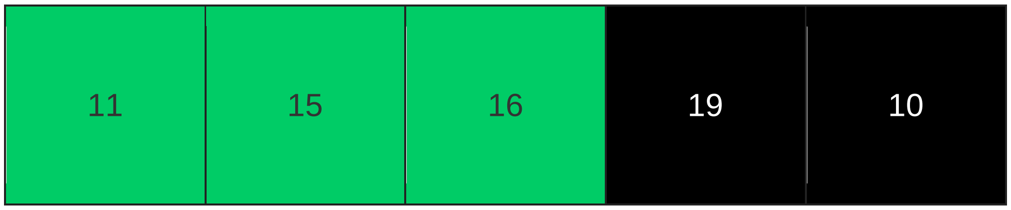
Swap



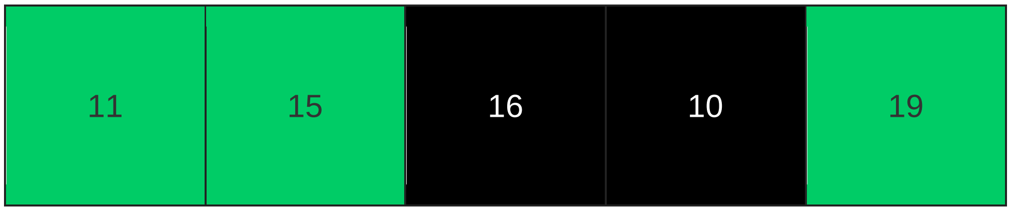
No swapping –



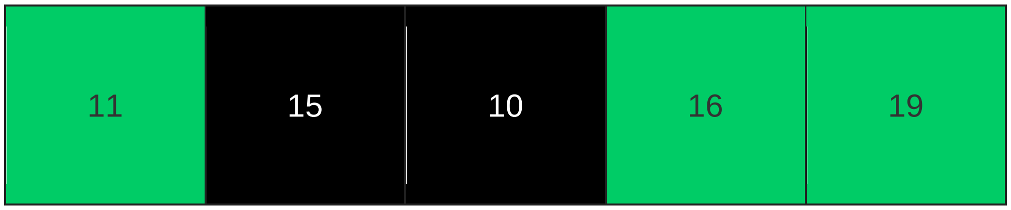
Pass -4:



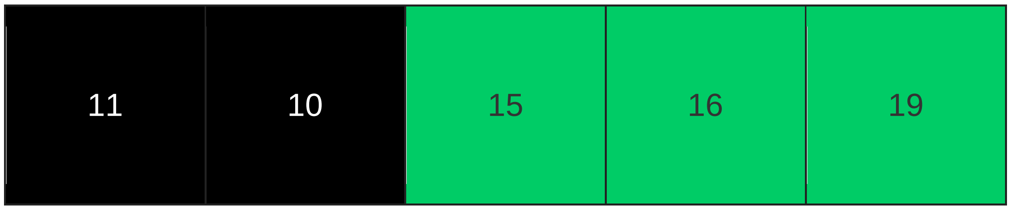
Swap



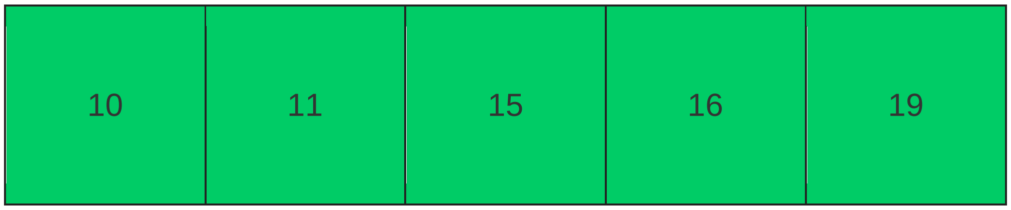
Swap



Swap



Swap



Python code:

```
arr = [16,19,11,15,10]
# Traverse through 1 to len(arr)
for i in range(1, len(arr)):
    key = arr[i]
    # Move elements of arr[0..i-1], that are
    # greater than key, to one position ahead
    # of their current position
    j = i-1
    while j >=0 and key < arr[j] :
        arr[j+1] = arr[j]
        j -= 1

    arr[j+1] = key
print "sorted list is :",arr
```

Merge Sort

Merge Sort is a kind of Divide and Conquer algorithm in computer programming. It is one of the most popular sorting algorithms and a great way to develop confidence in building recursive algorithms.

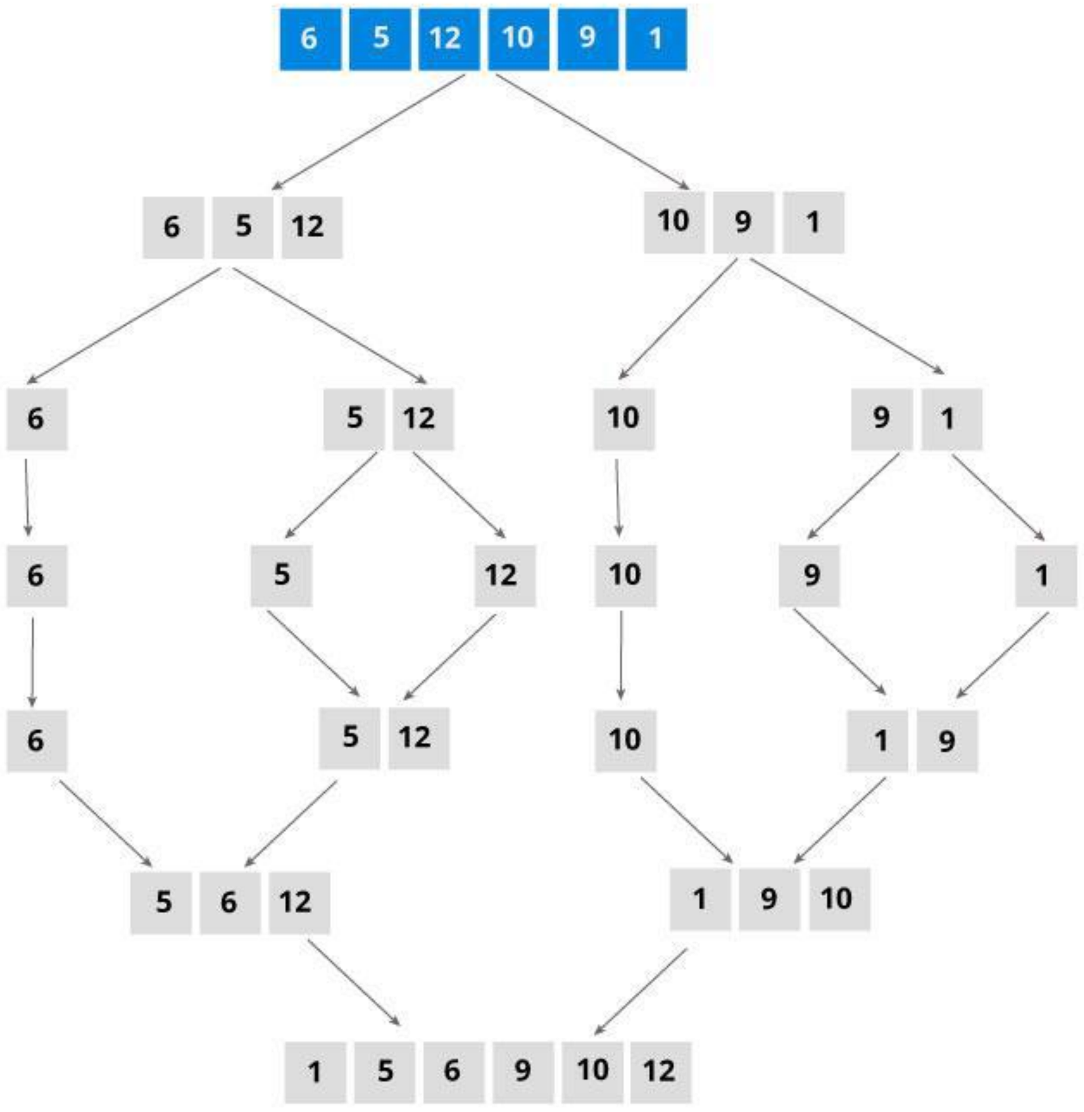
Divide and Conquer Strategy

Using the Divide and Conquer technique, we divide a problem into subproblems. When the solution to each subproblem is ready, we 'combine' the results from the subproblems to solve the main problem.

Merge Sort algorithm

Step 1 − if it is only one element in the list it is already sorted, return.
Step 2 − divide the list recursively into two halves until it can no more be divided.
Step 3 − merge the smaller lists into new list in sorted order.

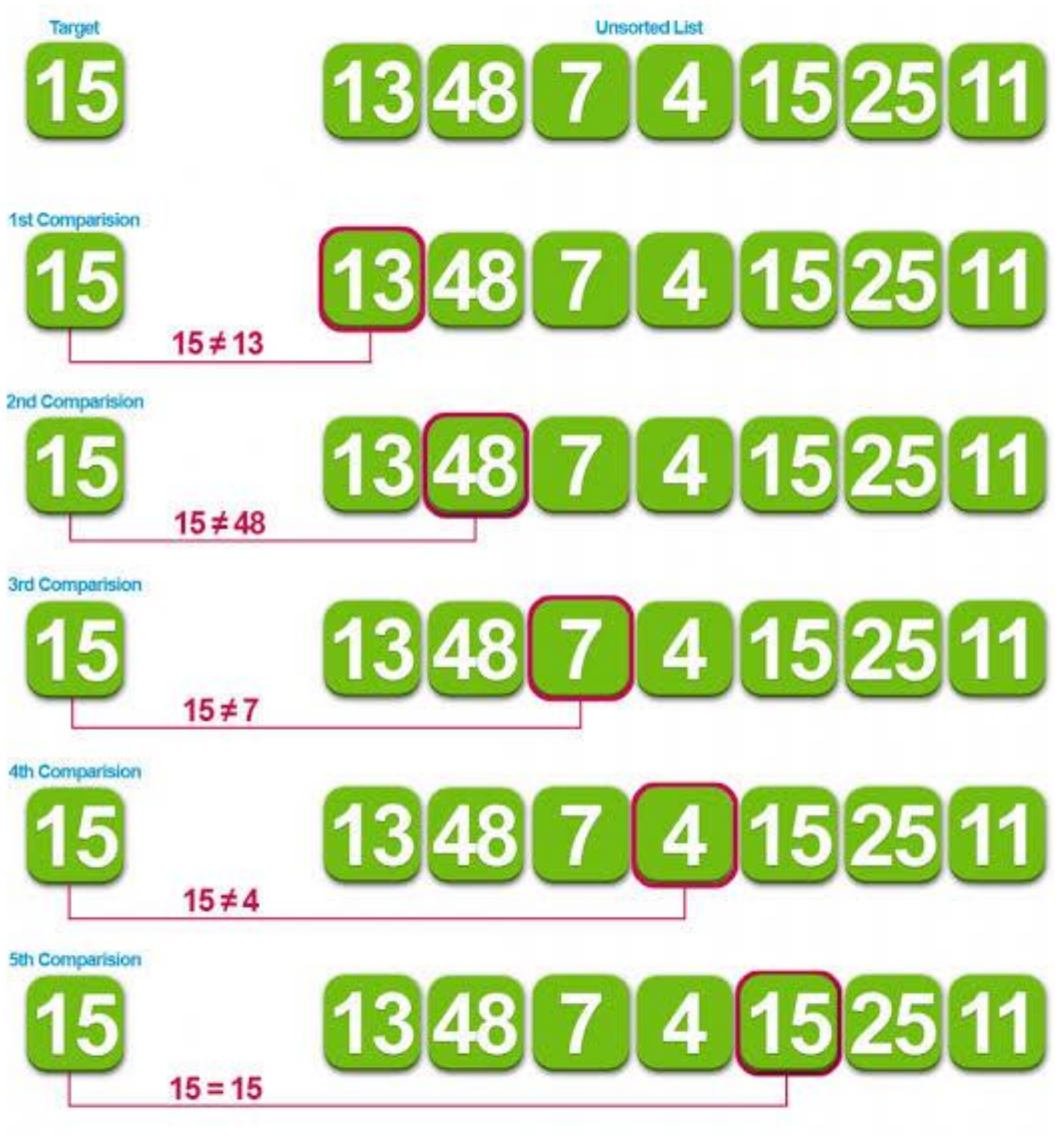
Merge Sort Example



Linear Search Algorithm

- Start from the leftmost element of given list and one by one compare element x(searching element) with each element of list.
- If x matches with any of the element, return the index value.
- If x doesn't match with any of elements in list , return -1 or element not found.

Linear search example:



Python code:

```
L=[13,48,7,4,15,25,11]
print "list of elements\t",L
key=input(raw_input("Enter the searching element\t"))
c=0
for i in range(0,len(L)):
        if L[i]==key:
                c=1
                break
if c==1:
        print "element found at index position",i
else:
        print "element not found"
```

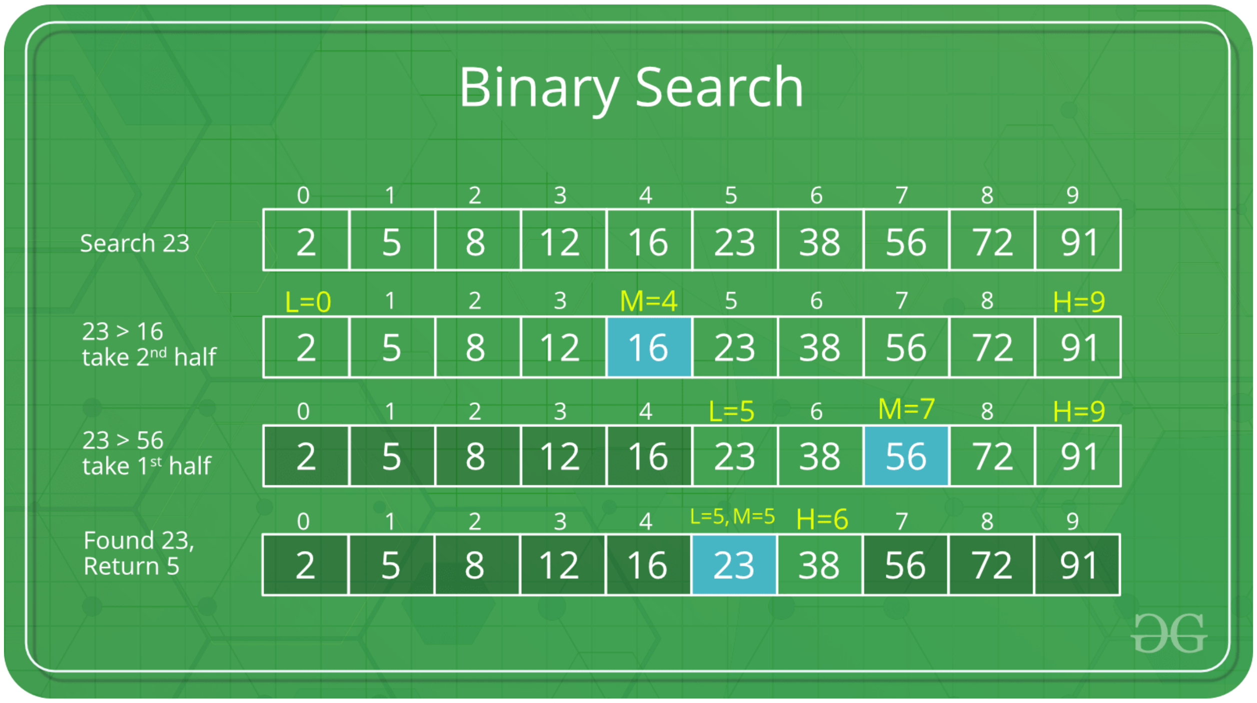
Binary Search

Search a sorted list by repeatedly dividing the search interval in half. Begin with an interval covering the whole list. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

We follow the following steps to perform binary search:

1. Compare x (search element) with the middle element.
2. If x matches with middle element, we return the mid index.
3. Else If x is greater than the mid element, then x can only lie in right half sub list after the mid element. So we recur for right half.
4. Else (x is smaller) recur for the left half.

Binary search example:



Python Code:

```
L=[2,5,8,12,16,23,28,56,72,91]
x=int(raw_input("Enter the number to search: "));
low=0
high=len(L)-1
c=0
while(low<=high):
        mid=(low+high)//2
        if(x==L[mid]):
                c=1;
                break;
        elif(x<L[mid]):
                high=mid-1
        else:
                low=mid+1
if c==1:
        print x,"is found at posotion",mid
else:
        print x,"is not found "
```