Bubble Sort

In Bubble sort, Each element of the array is compared with its adjacent element. The algorithm processes the list in passes. A list with n elements requires n-1 passes for sorting. Consider an array A of n elements whose elements are to be sorted by using Bubble sort. The algorithm processes like following.

1. In Pass 1, A[0] is compared with A[1], A[1] is compared with A[2], A[2] is compared with A[3] and so on. At the end of pass 1, the largest element of the list is placed at the highest index of the list.
2. In Pass 2, A[0] is compared with A[1], A[1] is compared with A[2] and so on. At the end of Pass 2 the second largest element of the list is placed at the second highest index of the list.
3. In pass n-1, A[0] is compared with A[1], A[1] is compared with A[2] and so on. At the end of this pass. The smallest element of the list is placed at the first index of the list.

Algorithm :

- **Step 1**: Repeat Step 2 For i = 0 to N-1
- **Step 2**: Repeat For J = i + 1 to N - I
- **Step 3**: IF A[J] > A[i] SWAP A[J] and A[i] [END OF INNER LOOP] [END OF OUTER LOOP
- **Step 4**: EXIT

Complexity

Scenario	Complexity
Space	O(1)
Worst case running time	$O(n^2)$
Average case running time	O(n)
Best case running time	$O(n^2)$

C Program

```c
#include<stdio.h>
void main ()
{
    int i, j,temp;
    int a[10] = { 10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
   for(i = 0; i<10; i++)
   {
      for(j = i+1; j<10; j++)
       {
          if(a[j] > a[i])
           {
              temp = a[i];
               a[i] = a[j];
              a[j] = temp;
           }
      }
   }
   printf("Printing Sorted Element List ...\n");
   for(i = 0; i<10; i++)
   {
       printf("%d\n",a[i]);
   }
```

```
23. }
```

Output:

```
Printing Sorted Element List . . .
7
9
10
12
23
34
34
44
78
101
```

C++ Program

```
1.  #include<iostream>
2.  using namespace std;
3.  int main ()
4.  {
5.      int i, j,temp;
6.      int a[10] = { 10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
7.      for(i = 0; i<10; i++)
8.      {
9.          for(j = i+1; j<10; j++)
10.         {
11.             if(a[j] < a[i])
12.             {
13.                 temp = a[i];
14.                 a[i] = a[j];
15.                 a[j] = temp;
16.             }
17.         }
18.     }
19.     cout <<"Printing Sorted Element List ...\n";
20.     for(i = 0; i<10; i++)
21.     {
22.         cout <<a[i]<<"\n";
23.     }
24.     return 0;
25. }
```

Output:

```
Printing Sorted Element List ...
7
9
10
12
23
23
34
44
78
101
```

Java Program

```
1.  public class BubbleSort {
2.      public static void main(String[] args) {
3.      int[] a = {10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
4.      for(int i=0;i<10;i++)
5.      {
6.          for (int j=0;j<10;j++)
7.          {
8.              if(a[i]<a[j])
```

```
9.              {
10.               int temp = a[i];
11.                a[i]=a[j];
12.               a[j] = temp;
13.             }
14.         }
15.     }
16.     System.out.println("Printing Sorted List ...");
17.     for(int i=0;i<10;i++)
18.     {
19.         System.out.println(a[i]);
20.     }
21. }
22. }
```

Output:

```
Printing Sorted List . . .
7
9
10
12
23
34
34
44
78
101
```

C# Program

```
1.  using System;
2.
3.  public class Program
4.  {
5.      public static void Main()
6.      {
7.          int i, j,temp;
8.      int[] a = {10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
9.      for(i = 0; i<10; i++)
10.     {
11.         for(j = i+1; j<10; j++)
12.         {
13.             if(a[j] > a[i])
14.             {
15.                 temp = a[i];
16.                a[i] = a[j];
17.                 a[j] = temp;
18.             }
19.         }
20.     }
21.     Console.WriteLine("Printing Sorted Element List ...\n");
22.     for(i = 0; i<10; i++)
23.     {
24.         Console.WriteLine(a[i]);
25.     }
26.     }
27. }
```

Output:

```
Printing Sorted Element List . . .
7
9
10
12
23
34
34
44
78
101
```

Python Program

```
a=[10, 9, 7, 101, 23, 44, 12, 78, 34, 23]
for i in range(0,len(a)):
    for j in range(i+1,len(a)):
        if a[j]<a[i]:
            temp = a[j]
            a[j]=a[i]
            a[i]=temp
print("Printing Sorted Element List...")
for i in a:
    print(i)
```

Output:

```
Printing Sorted Element List . . .
7
9
10
12
23
34
34
44
78
101
```

Rust Program

```
fn main()
{
    let mut temp;
 let mut a: [i32; 10] = [10, 9, 7, 101, 23, 44, 12, 78, 34, 23];
    for i in 0..10
    {
        for j in (i+1)..10
        {
            if a[j] < a[i]
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
 println!("Printing Sorted Element List ...\n");
 for i in &a
 {
     println!("{} ",i);
 }
}
```

Output:

```
Printing Sorted Element List . . .
7
9
10
```

```
12
23
34
34
44
78
101
4
```

JavaScript Program

```
<html>
<head>
<title>
Bubble Sort
</title>
</head>
<body>
<script>
    var a = [10, 9, 7, 101, 23, 44, 12, 78, 34, 23];
   for(i=0;i<10;i++)
   {
      for (j=0;j<10;j++)
      {
         if(a[i]<a[j])
          {
              temp = a[i];
              a[i]=a[j];
             a[j] = temp;
          }
      }
   }
   txt = "<br>";
   document.writeln("Printing Sorted Element List ..."+txt);
   for(i=0;i<10;i++)
   {
      document.writeln(a[i]);
       document.writeln(txt);
   }
   </script>
   </body>
</html>
```

Output:

```
Printing Sorted Element List ...
7
9
10
12
23
23
34
44
78
101
```

PHP Program

```
<html>
<head>
<title>Bubble Sort</title>
</head>
<body>
<?php
    $a = array(10, 9, 7, 101, 23, 44, 12, 78, 34, 23);
    for($i=0;$i<10;$i++)
```

```
9.      {
10.        for ($j=0;$j<10;$j++)
11.        {
12.           if($a[$i]<$a[$j])
13.           {
14.              $temp = $a[$i];
15.              $a[$i]=$a[$j];
16.             $a[$j] = $temp;
17.           }
18.        }
19.     }
20.     echo "Printing Sorted Element List ...\n";
21.     for($i=0;$i<10;$i++)
22.     {
23.         echo $a[$i];
24.        echo "\n";
25.     }
26. ?>
27. </body>
28. </html>
```

Output:

```
Printing Sorted Element List ...
7
9
10
12
23
23
34
44
78
101
```

Bucket Sort

Bucket sort is also known as bin sort. It works by distributing the element into the array also called buckets. Buckets are sorted individually by using different sorting algorithm.

Complexity of Bucket Sort

Algorithm	Complexity
Space	O(1)
Worst Case	$O(n^2)$
Best Case	Ω(n+k)
Average Case	θ(n+k)

Algorithm

- Step 1 START
- Step 2 Set up an array of initially empty "buckets".
- Step 3 Scatter: Go over the original array, putting each object in its bucket.
- Step 4 Sort each non-empty bucket.
- Step 5 Gather: Visit the buckets in order and put all elements back into the original array.
- Step 6 STOP

Program

```
1. #include <stdio.h>
```

```
2.  void Bucket_Sort(int array[], int n)
3.  {
4.      int i, j;
5.      int count[n];
6.      for (i = 0; i < n; i++)
7.          count[i] = 0;
8.
9.      for (i = 0; i < n; i++)
10.         (count[array[i]])++;
11.
12.     for (i = 0, j = 0; i < n; i++)
13.         for(; count[i] > 0; (count[i])--)
14.             array[j++] = i;
15. }
16. /* End of Bucket_Sort() */
17.
18. /* The main() begins */
19. int main()
20. {
21.     int array[100], i, num;
22.
23.     printf("Enter the size of array : ");
24.     scanf("%d", &num);
25.     printf("Enter the %d elements to be sorted:\n",num);
26.     for (i = 0; i < num; i++)
27.         scanf("%d", &array[i]);
28.     printf("\nThe array of elements before sorting : \n");
29.     for (i = 0; i < num; i++)
30.         printf("%d ", array[i]);
31.     printf("\nThe array of elements after sorting : \n");
32.     Bucket_Sort(array, num);
33.     for (i = 0; i < num; i++)
34.         printf("%d ", array[i]);
35.     printf("\n");
36.     return 0;
37. }
```

COMB SORT

Comb Sort is the advance form of Bubble Sort. Bubble Sort compares all the adjacent values while comb sort removes all the turtle values or small values near the end of the list.

Factors affecting comb sort are:

- It improves on bubble sort by using gap of size more than 1.
- Gap starts with large value and shrinks by the factor of 1.3.
- Gap shrinks till value reaches 1.

Complexity

Algorithm	Complexity
Worst Case Complexity	$O(n^2)$
Best Case Complexity	$\theta(n \log n)$
Average Case Complexity	$\Omega(n^2/2^p)$ where p is number of increments.
Worst Case Space Complexity	O(1)

Algorithm

- STEP 1 START
- STEP 2 Calculate the gap value if gap value==1 goto step 5 else goto step 3
- STEP 3 Iterate over data set and compare each item with gap item then goto step 4.
- STEP 4 Swap the element if require else goto step 2
- STEP 5 Print the sorted array.
- STEP 6 STOP

Program

```
#include <stdio.h>
#include <stdlib.h>
 int newgap(int gap)
{
    gap = (gap * 10) / 13;
    if (gap == 9 || gap == 10)
        gap = 11;
    if (gap < 1)
        gap = 1;
    return gap;
}

void combsort(int a[], int aSize)
{
   int gap = aSize;
   int temp, i;
   for (;;)
   {
      gap = newgap(gap);
      int swapped = 0;
      for (i = 0; i < aSize - gap; i++)
      {
         int j = i + gap;
         if (a[i] > a[j])
         {
            temp = a[i];
             a[i] = a[j];
            a[j] = temp;
             swapped  =  1;
         }
      }
      if (gap  ==  1 && !swapped)
         break;
   }
}
int main ()
{
   int n, i;
   int *a;
   printf("Please insert the number of elements to be sorted: ");
   scanf("%d", &n);       // The total number of elements
   a  =  (int *)calloc(n, sizeof(int));
   for (i = 0;i< n;i++)
   {
      printf("Input element %d :", i);
      scanf("%d", &a[i]); // Adding the elements to the array
```

```
47.    }
48.    printf("unsorted list");      // Displaying the unsorted array
49.    for(i = 0;i < n;i++)
50.    {
51.         printf("%d", a[i]);
52.    }
53.    combsort(a, n);
54.    printf("Sorted list:\n"); // Display the sorted array
55.    for(i = 0;i < n;i++)
56.    {
57.        printf("%d ", (a[i]));
58.    }
59.    return 0;
60. }
```

Counting Sort

It is a sorting technique based on the keys i.e. objects are collected according to keys which are small integers. Counting sort calculates the number of occurrence of objects and stores its key values. New array is formed by adding previous key elements and assigning to objects.

Complexity

Time Complexity: O(n+k) is worst case where n is the number of element and k is the range of input.

Space Complexity: O(k) k is the range of input.

Complexity	Best Case	Average Case	Worst Case
Time Complexity	Ω(n+k)	θ(n+k)	O(n+k)
Space Complexity			O(k)

Limitation of Counting Sort

- It is effective when range is not greater than number of object.
- It is not comparison based complexity.
- It is also used as sub-algorithm for different algorithm.
- It uses partial hashing technique to count the occurrence.
- It is also used for negative inputs.

Algorithm

- STEP 1 START
- STEP 2 Store the input array
- STEP 3 Count the key values by number of occurrence of object
- STEP 4 Update the array by adding previous key elements and assigning to objects
- STEP 5 Sort by replacing the object into new array and key= key-1
- STEP 6 STOP

Program

```
1. #include <stdio.h>
2. void counting_sort(int A[], int k, int n)
3. {
4.     int i, j;
5.     int B[15], C[100];
6.     for (i = 0; i <= k; i++)
```

```
7.          C[i] = 0;
8.      for (j = 1; j <= n; j++)
9.          C[A[j]] = C[A[j]] + 1;
10.     for (i = 1; i <= k; i++)
11.         C[i] = C[i] + C[i-1];
12.     for (j = n; j >= 1; j--)
13.     {
14.        B[C[A[j]]] = A[j];
15.         C[A[j]] = C[A[j]] - 1;
16.     }
17.     printf("The Sorted array is : ");
18.     for (i = 1; i <= n; i++)
19.         printf("%d ", B[i]);
20. }
21. /*  End of counting_sort()  */
22.
23. /*  The main() begins  */
24. int main()
25. {
26.     int n, k = 0, A[15], i;
27.     printf("Enter the number of input : ");
28.     scanf("%d", &n);
29.     printf("\nEnter the elements to be sorted :\n");
30.     for (i = 1; i <= n; i++)
31.     {
32.         scanf("%d", &A[i]);
33.         if (A[i] > k) {
34.             k = A[i];
35.         }
36.     }
37.     counting_sort(A, k, n);
38.     printf("\n");
39.     return 0;
40. }
```

Heap Sort

Heap sort processes the elements by creating the min heap or max heap using the elements of the given array. Min heap or max heap represents the ordering of the array in which root element represents the minimum or maximum element of the array. At each step, the root element of the heap gets deleted and stored into the sorted array and the heap will again be heapified.

The heap sort basically recursively performs two main operations.

- Build a heap H, using the elements of ARR.
- Repeatedly delete the root element of the heap formed in phase 1.

Complexity

Complexity	Best Case	Average Case	Worst case
Time Complexity	Ω(n log (n))	θ(n log (n))	O(n log (n))
Space Complexity			O(1)

Algorithm

HEAP_SORT(ARR, N)

- **Step 1**: [Build Heap H]
  Repeat for i=0 to N-1
  CALL INSERT_HEAP(ARR, N, ARR[i])
  [END OF LOOP]
- **Step 2**: Repeatedly Delete the root element
  Repeat while N > 0
  CALL Delete_Heap(ARR,N,VAL)
  SET N = N+1
  [END OF LOOP]
- **Step 3**: END

C Program

```
#include<stdio.h>
int temp;

void heapify(int arr[], int size, int i)
{
int largest = i;
int left = 2*i + 1;
int right = 2*i + 2;

if (left < size && arr[left] >arr[largest])
largest = left;

if (right < size && arr[right] > arr[largest])
largest = right;

if (largest != i)
{
temp = arr[i];
    arr[i]= arr[largest];
    arr[largest] = temp;
heapify(arr, size, largest);
}
}

void heapSort(int arr[], int size)
{
int i;
for (i = size / 2 - 1; i >= 0; i--)
heapify(arr, size, i);
for (i=size-1; i>=0; i--)
{
temp = arr[0];
    arr[0]= arr[i];
    arr[i] = temp;
heapify(arr, i, 0);
}
}

void main()
{
int arr[] = {1, 10, 2, 3, 4, 1, 2, 100,23, 2};
int i;
int size = sizeof(arr)/sizeof(arr[0]);

heapSort(arr, size);
```

```
46.
47. printf("printing sorted elements\n");
48. for (i=0; i<size; ++i)
49. printf("%d\n",arr[i]);
50. }
```

Output:

```
printing sorted elements

1
1
2
2
2
3
4
10
23
100
```

Java Program

```
1.  #include<stdio.h>
2.  int temp;
3.
4.  void heapify(int arr[], int size, int i)
5.  {
6.  int largest = i;
7.  int left = 2*i + 1;
8.  int right = 2*i + 2;
9.
10. if (left < size && arr[left] >arr[largest])
11. largest = left;
12.
13. if (right < size && arr[right] > arr[largest])
14. largest = right;
15.
16. if (largest != i)
17. {
18. temp = arr[i];
19.     arr[i]= arr[largest];
20.     arr[largest] = temp;
21. heapify(arr, size, largest);
22. }
23. }
24.
25. void heapSort(int arr[], int size)
26. {
27. int i;
28. for (i = size / 2 - 1; i >= 0; i--)
29. heapify(arr, size, i);
30. for (i=size-1; i>=0; i--)
31. {
32. temp = arr[0];
33.     arr[0]= arr[i];
34.     arr[i] = temp;
35. heapify(arr, i, 0);
36. }
37. }
38.
39. void main()
40. {
41. int arr[] = {1, 10, 2, 3, 4, 1, 2, 100, 23, 2};
```

```
42. int i;
43. int size = sizeof(arr)/sizeof(arr[0]);
44.
45. heapSort(arr, size);
46.
47. printf("printing sorted elements\n");
48. for (i=0; i<size; ++i)
49. printf("%d\n",arr[i]);
50. }
```

Output:

```
printing sorted elements

1
1
2
2
2
3
4
10
23
100
```

C# program

```
1.  using System;
2.  public class HeapSorting {
3.  static void heapify(int[] arr, int size, int i)
4.  {
5.  int largest = i;
6.  int left = 2*i + 1;
7.  int right = 2*i + 2;
8.  int temp;
9.  if (left < size && arr[left] > arr[largest])
10. largest = left;
11.
12. if (right < size && arr[right] > arr[largest])
13. largest = right;
14.
15. if (largest != i)
16. {
17. temp = arr[i];
18.     arr[i]= arr[largest];
19.     arr[largest] = temp;
20. heapify(arr, size, largest);
21. }
22. }
23.
24. static void heapSort(int[] arr, int size)
25. {
26. int i;
27. int temp;
28. for (i = size / 2 - 1; i >= 0; i--)
29. heapify(arr, size, i);
30. for (i=size-1; i>=0; i--)
31. {
32. temp = arr[0];
33.     arr[0]= arr[i];
34.     arr[i] = temp;
35. heapify(arr, i, 0);
36. }
37. }
```

```
38.
39. public void Main()
40. {
41. int[] arr = {1, 10, 2, 3, 4, 1, 2, 100, 23, 2};
42. int i;
43. heapSort(arr, 10);
44. Console.WriteLine("printing sorted elements");
45. for (i=0; i<10; ++i)
46. Console.WriteLine(arr[i]);
47. }
48. }
```

Output:

```
printing sorted elements

1
1
2
2
2
3
4
10
23
100
```

Insertion Sort

Insertion sort is the simple sorting algorithm which is commonly used in the daily lives while ordering a deck of cards. In this algorithm, we insert each element onto its proper place in the sorted array. This is less efficient than the other sort algorithms like quick sort, merge sort, etc.

Technique

Consider an array A whose elements are to be sorted. Initially, A[0] is the only element on the sorted set. In pass 1, A[1] is placed at its proper index in the array.

In pass 2, A[2] is placed at its proper index in the array. Likewise, in pass n-1, A[n-1] is placed at its proper index into the array.

To insert an element A[k] to its proper index, we must compare it with all other elements i.e. A[k-1], A[k-2], and so on until we find an element A[j] such that, A[j]<=A[k].

All the elements from A[k-1] to A[j] need to be shifted and A[k] will be moved to A[j+1].

Complexity

Complexity	Best Case	Average Case	Worst Case
Time	Ω(n)	θ(n^2)	o(n^2)
Space			o(1)

Algorithm

- **Step 1**: Repeat Steps 2 to 5 for K = 1 to N-1
- **Step 2**: SET TEMP = ARR[K]
- **Step 3**: SET J = K - 1
- **Step 4**: Repeat while TEMP <=ARR[J]
  SET ARR[J + 1] = ARR[J]
  SET J = J - 1
  [END OF INNER LOOP]
- **Step 5**: SET ARR[J + 1] = TEMP
  [END OF LOOP]
- **Step 6**: EXIT

C Program

```c
#include<stdio.h>
void main ()
{
    int i,j, k,temp;
    int a[10] = { 10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
    printf("\nprinting sorted elements...\n");
    for(k=1; k<10; k++)
    {
        temp = a[k];
       j= k-1;
        while(j>=0 && temp <= a[j])
        {
            a[j+1] = a[j];
           j = j-1;
        }
       a[j+1] = temp;
   }
    for(i=0;i<10;i++)
    {
        printf("\n%d\n",a[i]);
    }
}
```

Output:

```
Printing Sorted Elements . . .
7
9
10
12
23
23
34
44
78
101
```

C++ Program

```cpp
#include<iostream>
using namespace std;
int main ()
{
    int i,j, k,temp;
    int a[10] = { 10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
    cout<<"\nprinting sorted elements...\n";
    for(k=1; k<10; k++)
    {
        temp = a[k];
        j= k-1;
        while(j>=0 && temp <= a[j])
        {
            a[j+1] = a[j];
            j = j-1;
        }
        a[j+1] = temp;
    }
    for(i=0;i<10;i++)
    {
        cout <<a[i]<<"\n";
    }
}
```

Output:

```
printing sorted elements...
7
9
10
12
23
23
34
44
78
101
```

Java Program

```
1.  public class InsertionSort {
2.  public static void main(String[] args) {
3.      int[] a = {10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
4.      for(int k=1; k<10; k++)
5.      {
6.          int temp = a[k];
7.          int j= k-1;
8.          while(j>=0 && temp <= a[j])
9.          {
10.             a[j+1] = a[j];
11.             j = j-1;
12.         }
13.         a[j+1] = temp;
14.     }
15.     System.out.println("printing sorted elements ...");
16.     for(int i=0;i<10;i++)
17.     {
18.         System.out.println(a[i]);
19.     }
20. }
21. }
```

Output:

```
Printing sorted elements . . .
7
9
10
12
23
23
34
44
78
101
```

C# Program

```
1.  using System;
2.
3.  public class Program
4.  {
5.
6.      public static void Main()
7.      {
8.          int i,j, k,temp;
9.          int[] a = { 10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
10.     Console.WriteLine("\nprinting sorted elements...\n");
11.     for(k=1; k<10; k++)
12.     {
13.         temp = a[k];
14.         j= k-1;
```

```
15.         while(j>=0 && temp <= a[j])
16.         {
17.             a[j+1] = a[j];
18.            j = j-1;
19.         }
20.         a[j+1] = temp;
21.     }
22.     for(i=0;i<10;i++)
23.     {
24.         Console.WriteLine(a[i]);
25.     }
26. }
27. }
```

Output:

```
printing sorted elements . . .
7
9
10
12
23
23
34
44
78
101
```

Python Program

```
1.  a=[10, 9, 7, 101, 23, 44, 12, 78, 34, 23]
2.  for k in range(1,10):
3.      temp = a[k]
4.      j = k-1
5.      while j>=0 and temp <=a[j]:
6.          a[j+1]=a[j]
7.          j = j-1
8.      a[j+1] = temp
9.  print("printing sorted elements...")
10. for i in a:
11.     print(i)
```

Output:

```
printing sorted elements . . .
7
9
10
12
23
23
34
44
78
101
```

Swift Program

```
1.  import Foundation
2.  import Glibc
3.      var a = [10, 9, 7, 101, 23, 44, 12, 78, 34, 23];
4.      print("\nprinting sorted elements...\n");
5.      for k in 1...9
6.      {
7.          let temp = a[k];
8.          var j = k-1;
9.          while j>=0 && temp <= a[j]
10.         {
```

```
            a[j+1] = a[j];
          j = j-1;
       }

       a[j+1] = temp;
   }
   for i in a
   {
       print(i);
   }
```

Output:

```
printing sorted elements...

7
9
10
12
23
23
34
44
78
101
```

JavaScript Program

```
<html>
<head>
<title>
   Insertion Sort
</title>
   </head>
   <body>
      <script>
            var txt = "<br>";
        var a = [10, 9, 7, 101, 23, 44, 12, 78, 34, 23];
         document.writeln("printing sorted elements ... "+txt);
        for(k=0;k<10;k++)
         {
           var temp = a[k]
            j=k-1;
           while (j>=0 && temp <= a[j])
            {
              a[j+1] = a[j];
                jj = j-1;
           }
            a[j+1] = temp;
       }

       for(i=0;i<10;i++)
        {
           document.writeln(a[i]);
            document.writeln(txt);
       }
      </script>
   </body>
</html>
```

Output:

```
printing sorted elements ...
7
9
```

```
10
12
23
23
34
44
78
101
```

PHP Program

```
1.  <html>
2.  <head>
3.  <title>Insertion Sort</title>
4.  </head>
5.  <body>
6.  <?php
7.   $a = array(10, 9, 7, 101, 23, 44, 12, 78, 34, 23);
8.            echo("printing sorted elements ... \n");
9.             for($k=0;$k<10;$k++)
10.           {
11.               $temp = $a[$k];
12.              $j=$k-1;
13.               while ($j>=0 && $temp <= $a[$j])
14.              {
15.                  $a[$j+1] = $a[$j];
16.                $j = $j-1;
17.               }
18.              $a[$j+1] = $temp;
19.           }
20.          for($i=0;$i<10;$i++)
21.           {
22.              echo $a[$i];
23.               echo "\n";
24.          }
25. ?>
26. </body>
27. </html>
```

Output:

```
printing sorted elements ...
7
9
10
12
23
23
34
44
78
101
```

Merge sort

Merge sort is the algorithm which follows divide and conquer approach. Consider an array A of n number of elements. The algorithm processes the elements in 3 steps.

1. If A Contains 0 or 1 elements then it is already sorted, otherwise, Divide A into two sub-array of equal number of elements.
2. Conquer means sort the two sub-arrays recursively using the merge sort.
3. Combine the sub-arrays to form a single final sorted array maintaining the ordering of the array.

The main idea behind merge sort is that, the short list takes less time to be sorted.

Complexity

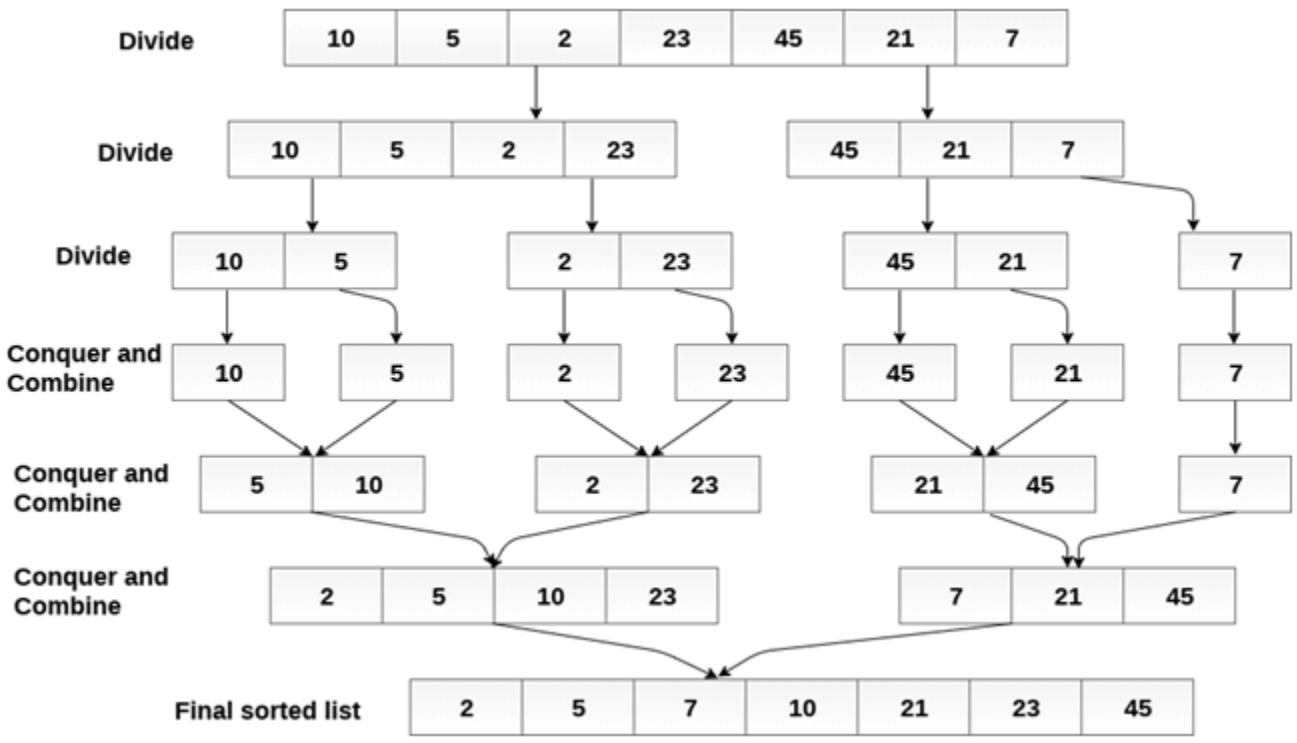
Complexity	Best case	Average Case	Worst Case

Time Complexity	O(n log n)	O(n log n)	O(n log n)
Space Complexity			O(n)

Example :

Consider the following array of 7 elements. Sort the array by using merge sort.

1. A = {10, 5, 2, 23, 45, 21, 7}



Algorithm

- **Step 1**: [INITIALIZE] SET I = BEG, J = MID + 1, INDEX = 0
- **Step 2**: Repeat while (I <= MID) AND (J<=END)
  IF ARR[I] < ARR[J]
  SET TEMP[INDEX] = ARR[I]
  SET I = I + 1
  ELSE
  SET TEMP[INDEX] = ARR[J]
  SET J = J + 1
  [END OF IF]
  SET INDEX = INDEX + 1
  [END OF LOOP]
  Step 3: [Copy the remaining elements of right sub-array, if any]
  IF I > MID
  Repeat while J <= END
  SET TEMP[INDEX] = ARR[J]
  SET INDEX = INDEX + 1, SET J = J + 1
  [END OF LOOP]
  [Copy the remaining elements of left sub-array, if any]
  ELSE
  Repeat while I <= MID
  SET TEMP[INDEX] = ARR[I]
  SET INDEX = INDEX + 1, SET I = I + 1
  [END OF LOOP]
  [END OF IF]
- **Step 4**: [Copy the contents of TEMP back to ARR] SET K = 0
- **Step 5**: Repeat while K < INDEX
  SET ARR[K] = TEMP[K]

SET K = K + 1
[END OF LOOP]

- **Step 6**: Exit

MERGE_SORT(ARR, BEG, END)

- **Step 1**: IF BEG < END
SET MID = (BEG + END)/2
CALL MERGE_SORT (ARR, BEG, MID)
CALL MERGE_SORT (ARR, MID + 1, END)
MERGE (ARR, BEG, MID, END)
[END OF IF]
- **Step 2**: END

C Program

```
1. #include<stdio.h>
2. void mergeSort(int[],int,int);
3. void merge(int[],int,int,int);
4. void main ()
5. {
6.     int a[10]= {10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
7.     int i;
8.     mergeSort(a,0,9);
9.     printf("printing the sorted elements");
10.    for(i=0;i<10;i++)
11.    {
12.        printf("\n%d\n",a[i]);
13.    }
14.
15. }
16. void mergeSort(int a[], int beg, int end)
17. {
18.    int mid;
19.    if(beg<end)
20.    {
21.        mid = (beg+end)/2;
22.        mergeSort(a,beg,mid);
23.        mergeSort(a,mid+1,end);
24.        merge(a,beg,mid,end);
25.    }
26. }
27. void merge(int a[], int beg, int mid, int end)
28. {
29.    int i=beg,j=mid+1,k,index = beg;
30.    int temp[10];
31.    while(i<=mid && j<=end)
32.    {
33.        if(a[i]<a[j])
34.        {
35.            temp[index] = a[i];
36.            i = i+1;
37.        }
38.        else
39.        {
40.            temp[index] = a[j];
41.            j = j+1;
42.        }
43.        index++;
```

```
44.    }
45.    if(i>mid)
46.    {
47.        while(j<=end)
48.        {
49.            temp[index] = a[j];
50.            index++;
51.            j++;
52.        }
53.    }
54.    else
55.    {
56.        while(i<=mid)
57.        {
58.            temp[index] = a[i];
59.            index++;
60.            i++;
61.        }
62.    }
63.    k = beg;
64.    while(k<index)
65.    {
66.        a[k]=temp[k];
67.        k++;
68.    }
69. }
```

Output:

```
printing the sorted elements

7
9
10
12
23
23
34
44
78
101
```

Java Program

```
1.  public class MyMergeSort
2.  {
3.  void merge(int arr[], int beg, int mid, int end)
4.  {
5.
6.  int l = mid - beg + 1;
7.  int r = end - mid;
8.
9.  intLeftArray[] = new int [l];
10. intRightArray[] = new int [r];
11.
12. for (int i=0; i<l; ++i)
13. LeftArray[i] = arr[beg + i];
14.
15. for (int j=0; j<r; ++j)
16. RightArray[j] = arr[mid + 1+ j];
17.
18.
19. int i = 0, j = 0;
20. int k = beg;
```

```
21. while (i<l&&j<r)
22. {
23. if (LeftArray[i] <= RightArray[j])
24. {
25. arr[k] = LeftArray[i];
26. i++;
27. }
28. else
29. {
30. arr[k] = RightArray[j];
31. j++;
32. }
33. k++;
34. }
35. while (i<l)
36. {
37. arr[k] = LeftArray[i];
38. i++;
39. k++;
40. }
41.
42. while (j<r)
43. {
44. arr[k] = RightArray[j];
45. j++;
46. k++;
47. }
48. }
49.
50. void sort(int arr[], int beg, int end)
51. {
52. if (beg<end)
53. {
54. int mid = (beg+end)/2;
55. sort(arr, beg, mid);
56. sort(arr , mid+1, end);
57. merge(arr, beg, mid, end);
58. }
59. }
60. public static void main(String args[])
61. {
62. intarr[] = {90,23,101,45,65,23,67,89,34,23};
63. MyMergeSort ob = new MyMergeSort();
64. ob.sort(arr, 0, arr.length-1);
65.
66. System.out.println("\nSorted array");
67. for(int i =0; i<arr.length;i++)
68. {
69.     System.out.println(arr[i]+"");
70. }
71. }
72. }
```

Output:

```
Sorted array
23
23
23
34
```

```
45
65
67
89
90
101
```

C# Program

```
1.  using System;
2.  public class MyMergeSort
3.  {
4.  void merge(int[] arr, int beg, int mid, int end)
5.  {
6.
7.  int l = mid - beg + 1;
8.  int r = end - mid;
9.          int i,j;
10.
11. int[] LeftArray = new int [l];
12. int[] RightArray = new int [r];
13.
14. for (i=0; i<l; ++i)
15. LeftArray[i] = arr[beg + i];
16.
17. for (j=0; j<r; ++j)
18. RightArray[j] = arr[mid + 1+ j];
19.
20.
21. i = 0; j = 0;
22. int k = beg;
23. while (i < l && j < r)
24. {
25. if (LeftArray[i] <= RightArray[j])
26. {
27. arr[k] = LeftArray[i];
28. i++;
29. }
30. else
31. {
32. arr[k] = RightArray[j];
33. j++;
34. }
35. k++;
36. }
37. while (i < l)
38. {
39. arr[k] = LeftArray[i];
40. i++;
41. k++;
42. }
43.
44. while (j < r)
45. {
46. arr[k] = RightArray[j];
47. j++;
48. k++;
49. }
50. }
51.
52. void sort(int[] arr, int beg, int end)
```

```
53. {
54. if (beg < end)
55. {
56. int mid = (beg+end)/2;
57. sort(arr, beg, mid);
58. sort(arr , mid+1, end);
59. merge(arr, beg, mid, end);
60. }
61. }
62. public static void Main()
63. {
64. int[] arr = {90,23,101,45,65,23,67,89,34,23};
65. MyMergeSort ob = new MyMergeSort();
66. ob.sort(arr, 0, 9);
67.
68. Console.WriteLine("\nSorted array");
69. for(int i =0; i<10;i++)
70. {
71.     Console.WriteLine(arr[i]+"");
72. }
73. }
74. }
```

Output:

```
Sorted array
23
23
23
34
45
65
67
89
90
101
```

Quick Sort

Quick sort is the widely used sorting algorithm that makes n log n comparisons in average case for sorting of an array of n elements. This algorithm follows divide and conquer approach. The algorithm processes the array in the following way.

1. Set the first index of the array to left and loc variable. Set the last index of the array to right variable. i.e. left = 0, loc = 0, en d = n - 1, where n is the length of the array.
2. Start from the right of the array and scan the complete array from right to beginning comparing each element of the array with the element pointed by loc.

   Ensure that, a[loc] is less than a[right].

   1. If this is the case, then continue with the comparison until right becomes equal to the loc.
   2. If a[loc] > a[right], then swap the two values. And go to step 3.
   3. Set, loc = right

1. start from element pointed by left and compare each element in its way with the element pointed by the variable loc. Ensure that a[loc] > a[left]
   1. if this is the case, then continue with the comparison until loc becomes equal to left.
   2. [loc] < a[right], then swap the two values and go to step 2.
   3. Set, loc = left.

Complexity

Complexity	Best Case	Average Case	Worst Case

Time Complexity	O(n) for 3 way partition or O(n log n) simple partition	O(n log n)	$O(n^2)$
Space Complexity			O(log n)

Algorithm

PARTITION (ARR, BEG, END, LOC)

- **Step 1**: [INITIALIZE] SET LEFT = BEG, RIGHT = END, LOC = BEG, FLAG =
- **Step 2**: Repeat Steps 3 to 6 while FLAG =
- **Step 3**: Repeat while ARR[LOC] <=ARR[RIGHT] AND LOC != RIGHT
  SET RIGHT = RIGHT - 1
  [END OF LOOP]
- **Step 4**: IF LOC = RIGHT
  SET FLAG = 1
  ELSE IF ARR[LOC] > ARR[RIGHT]
  SWAP ARR[LOC] with ARR[RIGHT]
  SET LOC = RIGHT
  [END OF IF]
- **Step 5**: IF FLAG = 0
  Repeat while ARR[LOC] >= ARR[LEFT] AND LOC != LEFT
  SET LEFT = LEFT + 1
  [END OF LOOP]
- **Step 6**:IF LOC = LEFT
  SET FLAG = 1
  ELSE IF ARR[LOC] < ARR[LEFT]
  SWAP ARR[LOC] with ARR[LEFT]
  SET LOC = LEFT
  [END OF IF]
  [END OF IF]
- **Step 7**: [END OF LOOP]
- **Step 8**: END

QUICK_SORT (ARR, BEG, END)

- **Step 1**: IF (BEG < END)
  CALL PARTITION (ARR, BEG, END, LOC)
  CALL QUICKSORT(ARR, BEG, LOC - 1)
  CALL QUICKSORT(ARR, LOC + 1, END)
  [END OF IF]
- **Step 2**: END

C Program

```c
#include <stdio.h>
int partition(int a[], int beg, int end);
void quickSort(int a[], int beg, int end);
void main()
{
    int i;
    int arr[10]={90,23,101,45,65,23,67,89,34,23};
    quickSort(arr, 0, 9);
    printf("\n The sorted array is: \n");
    for(i=0;i<10;i++)
    printf(" %d\t", arr[i]);
```

```
12. }
13. int partition(int a[], int beg, int end)
14. {
15.
16.    int left, right, temp, loc, flag;
17.    loc = left = beg;
18.    right = end;
19.    flag = 0;
20.    while(flag != 1)
21.    {
22.       while((a[loc] <= a[right]) && (loc!=right))
23.        right--;
24.       if(loc==right)
25.        flag =1;
26.       else if(a[loc]>a[right])
27.       {
28.          temp = a[loc];
29.           a[loc] = a[right];
30.          a[right] = temp;
31.           loc = right;
32.       }
33.        if(flag!=1)
34.       {
35.           while((a[loc] >= a[left]) && (loc!=left))
36.          left++;
37.           if(loc==left)
38.          flag =1;
39.           else if(a[loc] <a[left])
40.          {
41.              temp = a[loc];
42.             a[loc] = a[left];
43.              a[left] = temp;
44.             loc = left;
45.           }
46.       }
47.    }
48.    return loc;
49. }
50. void quickSort(int a[], int beg, int end)
51. {
52.
53.    int loc;
54.    if(beg<end)
55.    {
56.       loc = partition(a, beg, end);
57.        quickSort(a, beg, loc-1);
58.       quickSort(a, loc+1, end);
59.    }
60. }
```

Output:

```
The sorted array is:
23
23
23
34
45
65
67
89
90
```

Java Program

```
1.  public class QuickSort {
2.  public static void main(String[] args) {
3.          int i;
4.         int[] arr={90,23,101,45,65,23,67,89,34,23};
5.          quickSort(arr, 0, 9);
6.         System.out.println("\n The sorted array is: \n");
7.          for(i=0;i<10;i++)
8.         System.out.println(arr[i]);
9.      }
10.    public static int partition(int a[], int beg, int end)
11.    {
12.
13.         int left, right, temp, loc, flag;
14.        loc = left = beg;
15.         right = end;
16.        flag = 0;
17.       while(flag != 1)
18.      {
19.          while((a[loc] <= a[right]) && (loc!=right))
20.         right--;
21.          if(loc==right)
22.         flag =1;
23.          elseif(a[loc]>a[right])
24.         {
25.              temp = a[loc];
26.             a[loc] = a[right];
27.              a[right] = temp;
28.             loc = right;
29.          }
30.         if(flag!=1)
31.          {
32.             while((a[loc] >= a[left]) && (loc!=left))
33.              left++;
34.             if(loc==left)
35.              flag =1;
36.             elseif(a[loc] <a[left])
37.              {
38.                 temp = a[loc];
39.                  a[loc] = a[left];
40.                a[left] = temp;
41.                 loc = left;
42.             }
43.          }
44.      }
45.       returnloc;
46.   }
47.    static void quickSort(int a[], int beg, int end)
48.   {
49.
50.       int loc;
51.        if(beg<end)
52.       {
53.           loc = partition(a, beg, end);
54.          quickSort(a, beg, loc-1);
55.           quickSort(a, loc+1, end);
```

```
56.         }
57.     }
58. }
```

Output:

```
The sorted array is:
23
23
23
34
45
65
67
89
90
101
```

C# Program

```
1.  using System;
2.  public class QueueSort{
3.  public static void Main() {
4.          int i;
5.           int[] arr={90,23,101,45,65,23,67,89,34,23};
6.          quickSort(arr, 0, 9);
7.           Console.WriteLine("\n The sorted array is: \n");
8.          for(i=0;i<10;i++)
9.           Console.WriteLine(arr[i]);
10.    }
11.    public static int partition(int[] a, int beg, int end)
12.    {
13.
14.       int left, right, temp, loc, flag;
15.        loc = left = beg;
16.       right = end;
17.        flag = 0;
18.       while(flag != 1)
19.       {
20.          while((a[loc] <= a[right]) && (loc!=right))
21.           right--;
22.          if(loc==right)
23.           flag =1;
24.          else if(a[loc]>a[right])
25.          {
26.             temp = a[loc];
27.              a[loc] = a[right];
28.             a[right] = temp;
29.              loc = right;
30.          }
31.           if(flag!=1)
32.          {
33.              while((a[loc] >= a[left]) && (loc!=left))
34.             left++;
35.              if(loc==left)
36.             flag =1;
37.              else if(a[loc] <a[left])
38.             {
39.                 temp = a[loc];
40.                a[loc] = a[left];
41.                 a[left] = temp;
42.                loc = left;
43.              }
44.          }
```

```
45.         }
46.         return loc;
47.     }
48.     static void quickSort(int[] a, int beg, int end)
49.     {
50.
51.         int loc;
52.         if(beg<end)
53.         {
54.             loc = partition(a, beg, end);
55.             quickSort(a, beg, loc-1);
56.             quickSort(a, loc+1, end);
57.         }
58.     }
59. }
```

Output:

```
The sorted array is:
23
23
23
34
45
65
67
89
90
101
```

Radix Sort

Radix sort processes the elements the same way in which the names of the students are sorted according to their alphabetical order. There are 26 radix in that case due to the fact that, there are 26 alphabets in English. In the first pass, the names are grouped according to the ascending order of the first letter of names.

In the second pass, the names are grouped according to the ascending order of the second letter. The same process continues until we find the sorted list of names. The bucket are used to store the names produced in each pass. The number of passes depends upon the length of the name with the maximum letter.

In the case of integers, radix sort sorts the numbers according to their digits. The comparisons are made among the digits of the number from LSB to MSB. The number of passes depend upon the length of the number with the most number of digits.

Complexity

Complexity	Best Case	Average Case	Worst Case
Time Complexity	Ω(n+k)	θ(nk)	O(nk)
Space Complexity			O(n+k)

Example

Consider the array of length 6 given below. Sort the array by using Radix sort.

A = {10, 2, 901, 803, 1024}

Pass 1: (Sort the list according to the digits at 0's place)

10, 901, 2, 803, 1024.

Pass 2: (Sort the list according to the digits at 10's place)

02, 10, 901, 803, 1024

Pass 3: (Sort the list according to the digits at 100's place)

02, 10, 1024, 803, 901.

Pass 4: (Sort the list according to the digits at 1000's place)

02, 10, 803, 901, 1024

Therefore, the list generated in the step 4 is the sorted list, arranged from radix sort.

Algorithm

- **Step 1**:Find the largest number in ARR as LARGE
- **Step 2**: [INITIALIZE] SET NOP = Number of digits in LARGE
- **Step 3**: SET PASS =0
- **Step 4**: Repeat Step 5 while PASS <= NOP-1
- **Step 5**: SET I = 0 and INITIALIZE buckets
- **Step 6**:Repeat Steps 7 to 9 while I<n-1< li=""></n-1<>
- **Step 7**: SET DIGIT = digit at PASSth place in A[I]
- **Step 8**: Add A[I] to the bucket numbered DIGIT
- **Step 9**: INCREMENT bucket count for bucket numbered DIGIT [END OF LOOP]
- **Step 10**: Collect the numbers in the bucket [END OF LOOP]
- **Step 11**: END

C Program

```c
#include <stdio.h>
int largest(int a[]);
void radix_sort(int a[]);
void main()
{
    int i;
    int a[10]={90,23,101,45,65,23,67,89,34,23};
    radix_sort(a);
    printf("\n The sorted array is: \n");
    for(i=0;i<10;i++)
        printf(" %d\t", a[i]);
}

int largest(int a[])
{
    int larger=a[0], i;
    for(i=1;i<10;i++)
    {
        if(a[i]>larger)
        larger = a[i];
    }
    return larger;
}
void radix_sort(int a[])
{
    int bucket[10][10], bucket_count[10];
    int i, j, k, remainder, NOP=0, divisor=1, larger, pass;
    larger = largest(a);
    while(larger>0)
    {
        NOP++;
        larger/=10;
    }
```

```
for(pass=0;pass<NOP;pass++) // Initialize the buckets
{
   for(i=0;i<10;i++)
    bucket_count[i]=0;
   for(i=0;i<10;i++)
    {
      // sort the numbers according to the digit at passth place
       remainder = (a[i]/divisor)%10;
      bucket[remainder][bucket_count[remainder]] = a[i];
       bucket_count[remainder] += 1;
    }
    // collect the numbers after PASS pass
    i=0;
    for(k=0;k<10;k++)
    {
        for(j=0;j<bucket_count[k];j++)
       {
            a[i] = bucket[k][j];
            i++;
       }
    }
    divisor *= 10;

}
}
```

Output:

```
The sorted array is:
23
23
23
34
45
65
67
89
90
101
```

Java Program

```
1.  public class Radix_Sort {
2.  public static void main(String[] args) {
3.          int i;
4.          Scanner sc = new Scanner(System.in);
5.          int[] a = {90,23,101,45,65,23,67,89,34,23};
6.          radix_sort(a);
7.          System.out.println("\n The sorted array is: \n");
8.          for(i=0;i<10;i++)
9.              System.out.println(a[i]);
10.     }
11.
12.     static int largest(inta[])
13.     {
14.         int larger=a[0], i;
15.         for(i=1;i<10;i++)
16.         {
17.             if(a[i]>larger)
18.             larger = a[i];
19.         }
20.         returnlarger;
21.     }
22.     static void radix_sort(inta[])
```

```
23.    {
24.        int bucket[][]=newint[10][10];
25.         int bucket_count[]=newint[10];
26.        int i, j, k, remainder, NOP=0, divisor=1, larger, pass;
27.         larger = largest(a);
28.        while(larger>0)
29.         {
30.            NOP++;
31.             larger/=10;
32.        }
33.         for(pass=0;pass<NOP;pass++) // Initialize the buckets
34.        {
35.             for(i=0;i<10;i++)
36.            bucket_count[i]=0;
37.             for(i=0;i<10;i++)
38.            {
39.                 // sort the numbers according to the digit at passth place
40.                remainder = (a[i]/divisor)%10;
41.                 bucket[remainder][bucket_count[remainder]] = a[i];
42.                bucket_count[remainder] += 1;
43.            }
44.            // collect the numbers after PASS pass
45.             i=0;
46.            for(k=0;k<10;k++)
47.             {
48.                for(j=0;j<bucket_count[k];j++)
49.                 {
50.                    a[i] = bucket[k][j];
51.                     i++;
52.                }
53.            }
54.            divisor *= 10;
55.        }
56.    }
57. }
```

Output:

```
The sorted array is:
23
23
23
34
45
65
67
89
90
101
```

C# Program

```
1.  using System;
2.  public class Radix_Sort {
3.  public static void Main()
4.  {
5.          int i;
6.          int[] a = {90,23,101,45,65,23,67,89,34,23};
7.          radix_sort(a);
8.          Console.WriteLine("\n The sorted array is: \n");
9.          for(i=0;i<10;i++)
10.             Console.WriteLine(a[i]);
11.    }
12.
```

```
13.  static int largest(int[] a)
14.  {
15.      int larger=a[0], i;
16.     for(i=1;i<10;i++)
17.      {
18.         if(a[i]>larger)
19.          larger = a[i];
20.     }
21.      return larger;
22.  }
23.  static void radix_sort(int[] a)
24.  {
25.      int[,] bucket=new int[10,10];
26.     int[] bucket_count=new int[10];
27.      int i, j, k, remainder, NOP=0, divisor=1, larger, pass;
28.     larger = largest(a);
29.      while(larger>0)
30.     {
31.         NOP++;
32.        larger/=10;
33.     }
34.     for(pass=0;pass<NOP;pass++) // Initialize the buckets
35.     {
36.        for(i=0;i<10;i++)
37.         bucket_count[i]=0;
38.        for(i=0;i<10;i++)
39.         {
40.            // sort the numbers according to the digit at passth place
41.             remainder = (a[i]/divisor)%10;
42.            bucket[remainder,bucket_count[remainder]] = a[i];
43.             bucket_count[remainder] += 1;
44.        }
45.         // collect the numbers after PASS pass
46.        i=0;
47.         for(k=0;k<10;k++)
48.        {
49.             for(j=0;j<bucket_count[k];j++)
50.            {
51.                 a[i] = bucket[k,j];
52.                i++;
53.             }
54.         }
55.          divisor *= 10;
56.      }
57.   }
58. }
```

Output:

```
The sorted array is:
23
23
23
34
45
65
67
89
90
101
```

Selection Sort

In selection sort, the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array.

First, find the smallest element of the array and place it on the first position. Then, find the second smallest element of the array and place it on the second position. The process continues until we get the sorted array.

The array with n elements is sorted by using n-1 pass of selection sort algorithm.

- In 1st pass, smallest element of the array is to be found along with its index **pos**. then, swap A[0] and A[pos]. Thus A[0] is sorted, we now have n -1 elements which are to be sorted.
- In 2nd pas, position pos of the smallest element present in the sub-array A[n-1] is found. Then, swap, A[1] and A[pos]. Thus A[0] and A[1] are sorted, we now left with n-2 unsorted elements.
- In n-1th pass, position pos of the smaller element between A[n-1] and A[n-2] is to be found. Then, swap, A[pos] and A[n-1].

Therefore, by following the above explained process, the elements A[0], A[1], A[2],...., A[n-1] are sorted.

Example

Consider the following array with 6 elements. Sort the elements of the array by using selection sort.

A = {10, 2, 3, 90, 43, 56}.

Pass	Pos	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
1	1	2	10	3	90	43	56
2	2	2	3	10	90	43	56
3	2	2	3	10	90	43	56
4	4	2	3	10	43	90	56
5	5	**2**	**3**	**10**	**43**	**56**	**90**

Sorted A = {2, 3, 10, 43, 56, 90}

Complexity

Complexity	Best Case	Average Case	Worst Case
Time	Ω(n)	$\theta(n^2)$	$o(n^2)$
Space			o(1)

Algorithm

SELECTION SORT(ARR, N)

- **Step 1**: Repeat Steps 2 and 3 for K = 1 to N-1
- **Step 2**: CALL SMALLEST(ARR, K, N, POS)
- **Step 3**: SWAP A[K] with ARR[POS] [END OF LOOP]
- **Step 4**: EXIT

SMALLEST (ARR, K, N, POS)

- **Step 1**: [INITIALIZE] SET SMALL = ARR[K]
- **Step 2**: [INITIALIZE] SET POS = K
- **Step 3**: Repeat for J = K+1 to N -1 IF SMALL > ARR[J]

SET SMALL = ARR[J]
SET POS = J
[END OF IF]
[END OF LOOP]

- **Step 4**: RETURN POS

C Program

```c
#include<stdio.h>
int smallest(int[],int,int);
void main ()
{
    int a[10] = {10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
    int i,j,k,pos,temp;
    for(i=0;i<10;i++)
    {
        pos = smallest(a,10,i);
        temp = a[i];
        a[i]=a[pos];
        a[pos] = temp;
    }
    printf("\nprinting sorted elements...\n");
    for(i=0;i<10;i++)
    {
        printf("%d\n",a[i]);
    }
}
int smallest(int a[], int n, int i)
{
    int small,pos,j;
    small = a[i];
    pos = i;
    for(j=i+1;j<10;j++)
    {
        if(a[j]<small)
        {
            small = a[j];
            pos=j;
        }
    }
    return pos;
}
```

Output:

```
printing sorted elements...
7
9
10
12
23
23
34
44
78
101
```

C++ Program

```cpp
#include<iostream>
using namespace std;
int smallest(int[],int,int);
int main ()
{
```

```
6.      int a[10] = {10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
7.      int i,j,k,pos,temp;
8.      for(i=0;i<10;i++)
9.      {
10.         pos = smallest(a,10,i);
11.         temp = a[i];
12.         a[i]=a[pos];
13.         a[pos] = temp;
14.     }
15.     cout<<"\n printing sorted elements...\n";
16.     for(i=0;i<10;i++)
17.     {
18.         cout<<a[i]<<"\n";
19.     }
20.     return 0;
21. }
22. int smallest(int a[], int n, int i)
23. {
24.     int small,pos,j;
25.     small = a[i];
26.     pos = i;
27.     for(j=i+1;j<10;j++)
28.     {
29.         if(a[j]<small)
30.         {
31.             small = a[j];
32.             pos=j;
33.         }
34.     }
35.     return pos;
36. }
```

Output:

```
printing sorted elements...
7
9
10
12
23
23
34
44
78
101
```

Java Program

```
1.  public class SelectionSort {
2.  public static void main(String[] args) {
3.      int[] a = {10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
4.      int i,j,k,pos,temp;
5.      for(i=0;i<10;i++)
6.      {
7.          pos = smallest(a,10,i);
8.          temp = a[i];
9.          a[i]=a[pos];
10.         a[pos] = temp;
11.     }
12.     System.out.println("\nprinting sorted elements...\n");
13.     for(i=0;i<10;i++)
14.     {
15.         System.out.println(a[i]);
```

```
16.     }
17. }
18. public static int smallest(int a[], int n, int i)
19. {
20.     int small,pos,j;
21.     small = a[i];
22.     pos = i;
23.     for(j=i+1;j<10;j++)
24.     {
25.         if(a[j]<small)
26.         {
27.             small = a[j];
28.             pos=j;
29.         }
30.     }
31.     return pos;
32. }
33. }
```

Output:

```
printing sorted elements...
7
9
10
12
23
23
34
44
78
101
```

C# Program

```
1.  using System;
2.  public class Program
3.  {
4.
5.
6.  public void Main() {
7.      int[] a = {10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
8.      int i,pos,temp;
9.      for(i=0;i<10;i++)
10.     {
11.         pos = smallest(a,10,i);
12.         temp = a[i];
13.         a[i]=a[pos];
14.         a[pos] = temp;
15.     }
16.     Console.WriteLine("\nprinting sorted elements...\n");
17.     for(i=0;i<10;i++)
18.     {
19.         Console.WriteLine(a[i]);
20.     }
21. }
22. public static int smallest(int[] a, int n, int i)
23. {
24.     int small,pos,j;
25.     small = a[i];
26.     pos = i;
27.     for(j=i+1;j<10;j++)
28.     {
```

```
29.         if(a[j]<small)
30.        {
31.             small = a[j];
32.            pos=j;
33.         }
34.     }
35.     return pos;
36. }
37. }
```

Output:

```
printing sorted elements...
7
9
10
12
23
23
34
44
78
101
```

Python Program

```
1.  def smallest(a,i):
2.      small = a[i]
3.      pos=i
4.      for j in range(i+1,10):
5.          if a[j] < small:
6.              small = a[j]
7.              pos = j
8.      return pos
9.
10. a=[10, 9, 7, 101, 23, 44, 12, 78, 34, 23]
11. for i in range(0,10):
12.     pos = smallest(a,i)
13.     temp = a[i]
14.     a[i]=a[pos]
15.     a[pos]=temp
16. print("printing sorted elements...")
17. for i in a:
18.     print(i)
```

Output:

```
printing sorted elements...
7
9
10
12
23
23
34
44
78
101
```

Rust Program

```
1.  fn main ()
2.  {
3.      let mut a: [i32;10] = [10, 9, 7, 101, 23, 44, 12, 78, 34, 23];
4.
5.      for i in 0..10
6.      {
7.          let mut small = a[i];
```

```
8.          let mut pos = i;
9.          for j in (i+1)..10
10.         {
11.         if a[j]<small
12.         {
13.             small = a[j];
14.            pos=j;
15.         }
16.     }
17.         let mut temp = a[i];
18.         a[i]=a[pos];
19.         a[pos] = temp;
20.     }
21.     println!("\nprinting sorted elements...\n");
22.     for i in &a
23.     {
24.         println!("{}",i);
25.     }
26. }
```

Output:

```
printing sorted elements...

7
9
10
12
23
23
34
44
78
101
```

JavaScript Program

```
1. <html>
2. <head>
3. <title>
4.     Selection Sort
5. </title>
6.     </head>
7.     <body>
8.     <script>
9. function smallest(a,  n, i)
10. {
11.
12.     var small = a[i];
13.     var pos = i;
14.     for(j=i+1;j<10;j++)
15.     {
16.         if(a[j]<small)
17.         {
18.             small = a[j];
19.             pos=j;
20.         }
21.     }
22.     return pos;
23. }
24. var a = [10, 9, 7, 101, 23, 44, 12, 78, 34, 23];
25.     for(i=0;i<10;i++)
26.     {
```

```
27.         pos = smallest(a,10,i);
28.        temp = a[i];
29.         a[i]=a[pos];
30.        a[pos] = temp;
31.    }
32.    document.writeln("printing sorted elements ...\n"+"<br>");
33.    for(i=0;i<10;i++)
34.    {
35.        document.writeln(a[i]+"<br>");
36.    }
37.        </script>
38.    </body>
39. </html>
```

Output:

```
printing sorted elements ...
7
9
10
12
23
23
34
44
78
101
```

PHP Program

```
1.  <html>
2.  <head>
3.  <title>Selection sort</title>
4.  </head>
5.  <body>
6.  <?php
7.  function smallest($a,  $n, $i)
8.  {
9.
10.    $small = $a[$i];
11.    $pos = $i;
12.    for($j=$i+1;$j<10;$j++)
13.    {
14.        if($a[$j]<$small)
15.        {
16.            $small = $a[$j];
17.            $pos=$j;
18.        }
19.    }
20.    return $pos;
21. }
22. $a = array(10, 9, 7, 101, 23, 44, 12, 78, 34, 23);
23.    for($i=0;$i<10;$i++)
24.    {
25.        $pos = smallest($a,10,$i);
26.        $temp = $a[$i];
27.        $a[$i]=$a[$pos];
28.        $a[$pos] = $temp;
29.    }
30.    echo "printing sorted elements ...\n";
31.    for($i=0;$i<10;$i++)
32.    {
33.        echo $a[$i];
```

```
34.         echo "\n";
35.     }
36. ?>
37. </body>
38. </html>
```

Output:

```
printing sorted elements ...
7
9
10
12
23
23
34
44
78
101
```

Shell Sort

Shell sort is the generalization of insertion sort which overcomes the drawbacks of insertion sort by comparing elements separated by a gap of several positions. In general, Shell sort performs the following steps.

- **Step 1**: Arrange the elements in the tabular form and sort the columns by using insertion sort.
- **Step 2**: Repeat Step 1; each time with smaller number of longer columns in such a way that at the end, there is only one column of data to be sorted.

Complexity

Complexity	Best Case	Average Case	Worst Case
Time Complexity	$\Omega(n \log(n))$	$\theta(n \log(n)^2)$	$O(n \log(n)^2)$
Space Complexity			O(1)

Algorithm

Shell_Sort(Arr, n)

- **Step 1**: SET FLAG = 1, GAP_SIZE = N
- **Step 2**: Repeat Steps 3 to 6 while FLAG = 1 OR GAP_SIZE > 1
- **Step 3**:SET FLAG = 0
- **Step 4**:SET GAP_SIZE = (GAP_SIZE + 1) / 2
- **Step 5**:Repeat Step 6 for I = 0 to I < (N -GAP_SIZE)
- **Step 6**:IF Arr[I + GAP_SIZE] > Arr[I]
  SWAP Arr[I + GAP_SIZE], Arr[I]
  SET FLAG = 0
- **Step 7**: END

C Program

```
1. #include <stdio.h>
2. void shellsort(int arr[], int num)
3. {
4. int i, j, k, tmp;
5. for (i = num / 2; i > 0; i = i / 2)
6. {
7. for (j = i; j < num; j++)
8. {
9. for(k = j - i; k >= 0; k = k - i)
10. {
11. if (arr[k+i] >= arr[k])
12. break;
```

```
13. else
14. {
15. tmp = arr[k];
16. arr[k] = arr[k+i];
17. arr[k+i] = tmp;
18. }
19. }
20. }
21. }
22. }
23. int main()
24. {
25. int arr[30];
26. int k,  num;
27. printf("Enter total no. of elements : ");
28. scanf("%d", &num);
29. printf("\nEnter %d numbers: ", num);
30.
31. for (k =  0 ; k < num; k++)
32. {
33. scanf("%d", &arr[k]);
34. }
35. shellsort(arr, num);
36. printf("\n Sorted array is: ");
37. for (k = 0; k < num; k++)
38. printf("%d ", arr[k]);
39. return 0;
40. }
```

Output:

```
Enter total no. of elements : 6

Enter 6 numbers: 3
2
4
10
2
1

Sorted array is: 1 2 2 3 4 10
```

Java Program

```
1. import java.util.*;
2. public class Shell_Sort
3. {
4. static void shellsort(int[] arr, intnum)
5. {
6. int i, j, k, tmp;
7. for (i = num / 2; i> 0; i = i / 2)
8. {
9. for (j = i; j<num; j++)
10. {
11. for(k = j - i; k>= 0; k = k - i)
12. {
13. if (arr[k+i] >= arr[k])
14. break;
15. else
16. {
17. tmp = arr[k];
18. arr[k] = arr[k+i];
19. arr[k+i] = tmp;
```

```
20. }
21. }
22. }
23. }
24. }
25. public static void main(String[] args)
26. {
27. Scanner sc = new Scanner(System.in);
28. int arr[]= newint[30];
29. intk,  num;
30. System.out.println("Enter total no. of elements : ");
31. num = sc.nextInt();
32. for (k =  0 ; k<num; k++)
33. {
34. arr[k]=sc.nextInt();
35. }
36. shellsort(arr, num);
37. System.out.println("\n Sorted array is: ");
38. for (k = 0; k<num; k++)
39. System.out.println(arr[k]);
40. }
41. }
```

Output:

```
Enter total no. of elements : 6
3
2
4
10
2
1

Sorted array is: 1 2 2 3 4 10
```

C# Program

```
1. using System;
2. public class Shell_Sort
3. {
4. static void shellsort(int[] arr, int num)
5. {
6. int i, j, k, tmp;
7. for (i = num / 2; i > 0; i = i / 2)
8. {
9. for (j = i; j < num; j++)
10. {
11. for(k = j - i; k >= 0; k = k - i)
12. {
13. if (arr[k+i] >= arr[k])
14. break;
15. else
16. {
17. tmp = arr[k];
18. arr[k] = arr[k+i];
19. arr[k+i] = tmp;
20. }
21. }
22. }
23. }
```

```
24. }
25. public void Main()
26. {
27.     int[] arr=new int[10];
28. int k,  num;
29. Console.WriteLine("Enter total no. of elements : ");
30. num = Convert.ToInt32(Console.ReadLine());
31. for (k =  0 ; k < num; k++)
32. {
33. arr[k]=Convert.ToInt32(Console.ReadLine());
34. }
35. shellsort(arr, num);
36. Console.WriteLine("\n Sorted array is: ");
37. for (k = 0; k < num; k++)
38. Console.WriteLine(arr[k]);
39. }
40. }
```

Output:

```
Enter total no. of elements : 6

3
2
4
10
2
1

Sorted array is: 1 2 2 3 4 10
```

Bitonic Sort

Bitonic sort is a parallel sorting algorithm which performs O(n2 log n) comparisons. Although, the number of comparisons are more than that in any other popular sorting algorithm, It performs better for the parallel implementation because elements are compared in predefined sequence which must not be depended upon the data being sorted. The predefined sequence is called Bitonic sequence.

What is Bitonic Sequence ?

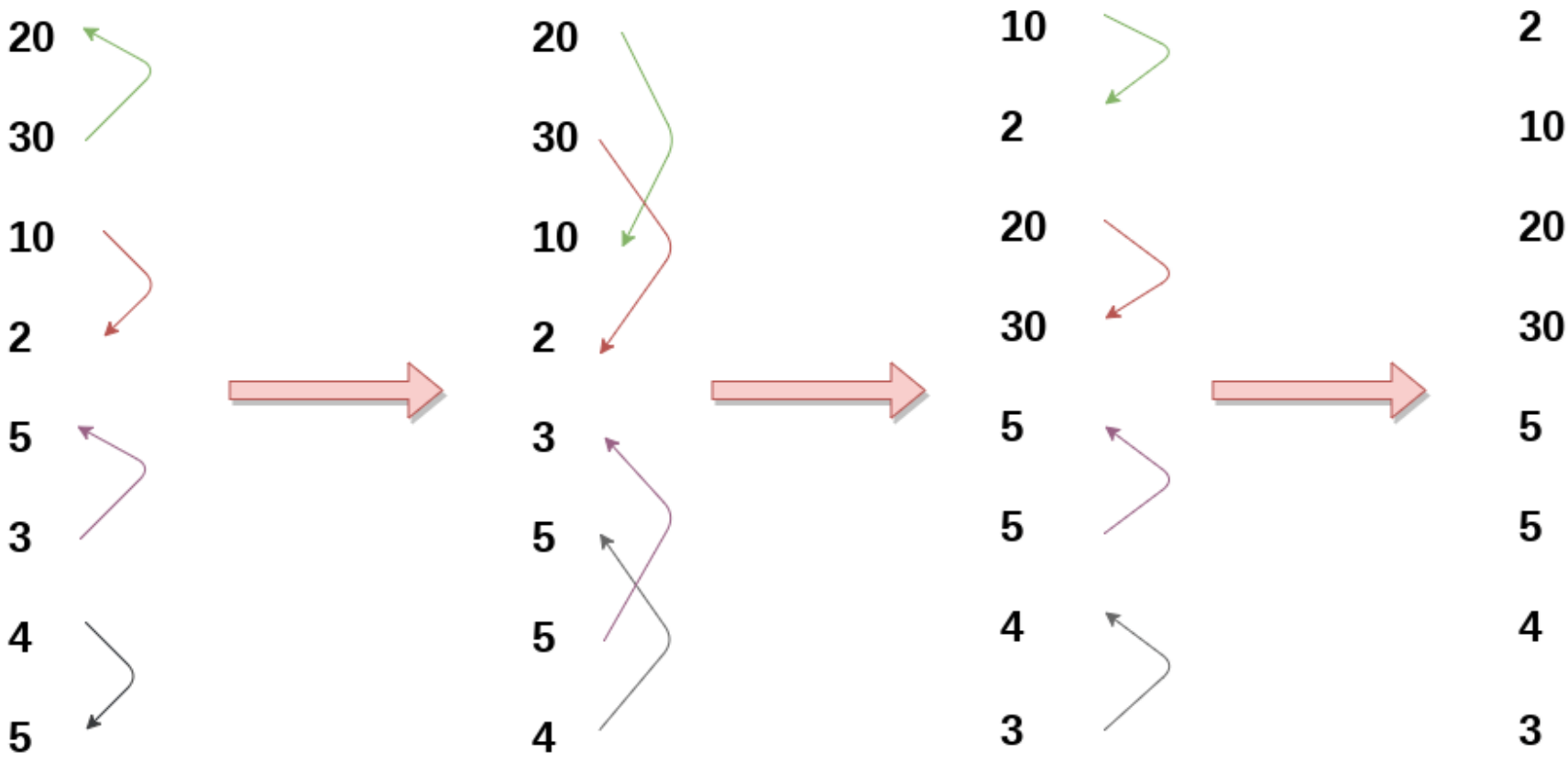
In order to understand Bitonic sort, we must understand Bitonic sequence. Bitonic sequence is the one in which, the elements first comes in increasing order then start decreasing after some particular index. An array A[0... i ... n-1] is called Bitonic if there exist an index i such that,

1. A[0] < A[1] < A[2] .... A[i-1] < A[i] > A[i+1] > A[i+2] > A[i+3] > ... >A[n-1]

where, 0 <= i <= n-1. A rotation of Bitonic sort is also Bitonic.

How to convert Random sequence to Bitonic sequence ?

Consider a sequence A[0 ... n-1] of n elements. First start constructing Bitonic sequence by using 4 elements of the sequence. Sort the first 2 elements in ascending order and the last 2 elements in descending order, concatenate this pair to form a Bitonic sequence of 4 elements. Repeat this process for the remaining pairs of element until we find a Bitonic sequence.



Constructing Bitonic Sequence

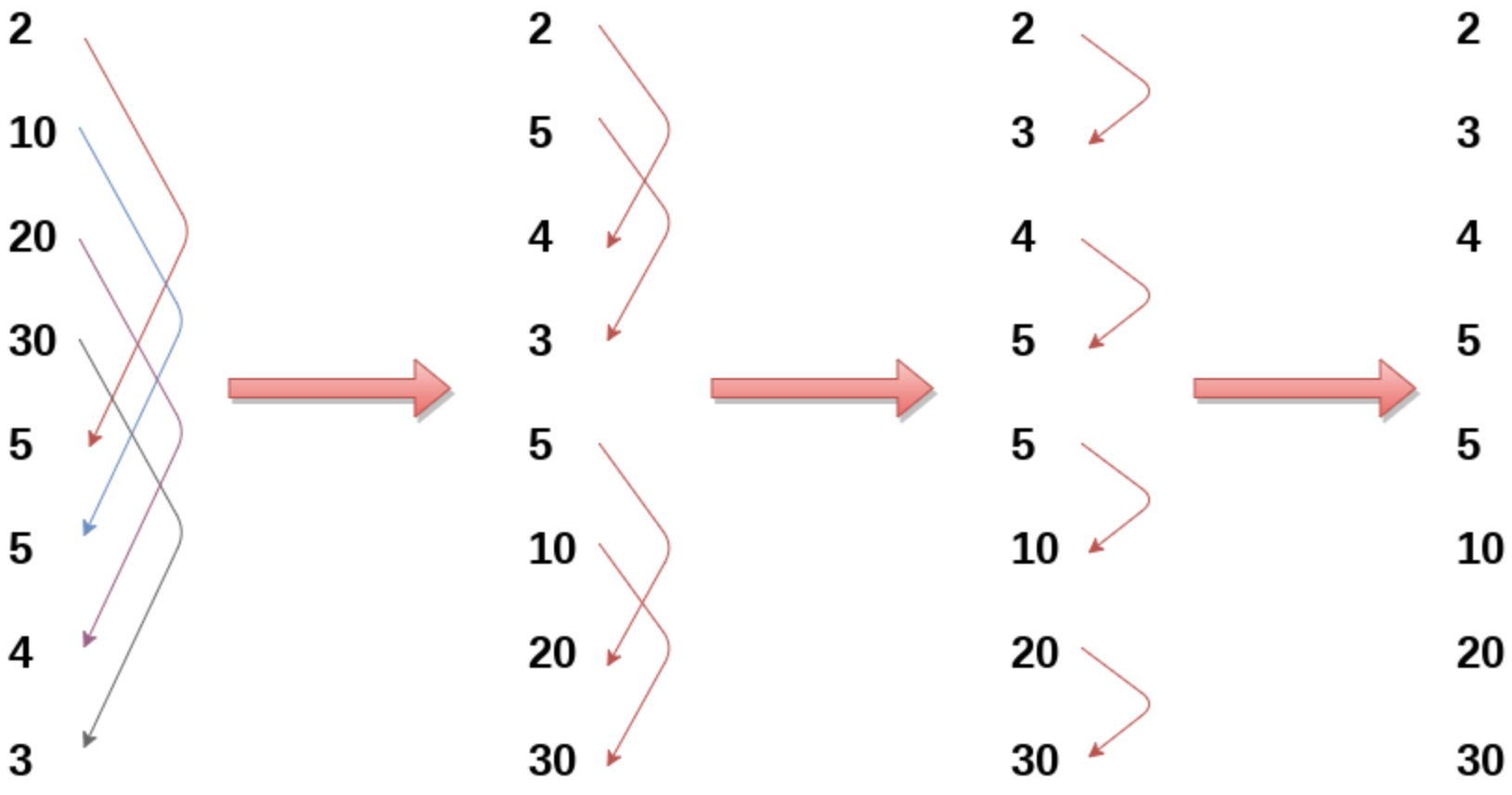
After this step, we get the Bitonic sequence for the given sequence as 2, 10, 20, 30, 5, 5, 4, 3.

Bitonic Sorting :

Bitonic sorting mainly involves the following basic steps.

1. Form a Bitonic sequence from the given random sequence which we have formed in the above step. We can consider it as the first step in our procedure. After this step, we get a sequence whose first half is sorted in ascending order while second step is sorted in descending order.
2. Compare first element of first half with the first element of the second half, then second element of the first half with the second element of the second half and so on. Swap the elements if any element in the second half is found to be smaller.
3. After the above step, we got all the elements in the first half smaller than all the elements in the second half. The compare and swap results into the two sequences of n/2 length each. Repeat the process performed in the second step recursively onto every sequence until we get single sorted sequence of length n.

The whole procedure involved in Bitonic sort is described in the following image.



Constructing Sorted Sequence

Complexity

Complexity	Best Case	Average Case	Worst Case
Time Complexity	$O(\log^2 n)$	$O(\log^2 n)$	$O(\log^2 n)$
Space Complexity			$O(n \log^2 n)$

C Program

```
1. //this program works when size of input is power of 2.
2. #include<stdio.h>
3. void exchange(int arr[], int i, int j, int d)
4. {
5.     int temp;
6.     if (d==(arr[i]>arr[j]))
7.     {
8.         temp = arr[i];
9.         arr[i] = arr[j];
10.        arr[j] = temp;
11.    }
12. }
13. void merge(int arr[], int l, int c, int d)
14. {
15.    int k,i;
16.    if (c>1)
17.    {
18.        k = c/2;
19.        for (i=l; i<l+k; i++)
20.            exchange(arr, i, i+k, d);
21.        merge(arr, l, k, d);
22.        merge(arr, l+k, k, d);
23.    }
24. }
25. void bitonicSort(int arr[],int l, int c, int d)
```

```
{
    int k;
    if (c>1)
    {
        k = c/2;
        bitonicSort(arr, l, k, 1);
        bitonicSort(arr, l+k, k, 0);
        merge(arr,l, c, d);
    }
}

void sort(int arr[], int n, int order)
{
    bitonicSort(arr,0, n, order);
}
int main()
{
    int arr[]= {1, 10, 2, 3, 1, 23, 45, 21};
    int n = sizeof(arr)/sizeof(arr[0]);
    int i;
    int order = 1;
    sort(arr, n, order);

    printf("Sorted array: \n");
    for (i=0; i<n; i++)
        printf("%d ", arr[i]);
}
```

Output:

```
Sorted array:
1 1 2 3 10 21 23 45
```

Java

```
1. //this program works when size of input is power of 2.
2. public class BitonicSort
3. {
4. static void exchange(int arr[], int i, int j, boolean d)
5. {
6.     int temp;
7.     if (d==(arr[i]>arr[j]))
8.     {
9.         temp = arr[i];
10.        arr[i] = arr[j];
11.        arr[j] = temp;
12.    }
13. }
14. static void merge(int arr[], int l, int c, boolean d)
15. {
16.    int k,i;
17.    if (c>1)
18.    {
19.        k = c/2;
20.        for (i=l; i<l+k; i++)
21.            exchange(arr, i, i+k, d);
22.        merge(arr, l, k, d);
23.        merge(arr, l+k, k, d);
24.    }
```

```
25. }
26. static void bitonicSort(int arr[],int l, int c, boolean d)
27. {
28.     int k;
29.     if (c>1)
30.     {
31.         k = c/2;
32.         bitonicSort(arr, l, k, true);
33.         bitonicSort(arr, l+k, k, false);
34.         merge(arr,l, c, d);
35.     }
36. }
37.
38. static void sort(int arr[], int n, boolean order)
39. {
40.     bitonicSort(arr,0, n, order);
41. }
42. public static void main(String[] args)
43. {
44.     int arr[]= {1, 10, 2, 3, 1, 23, 45, 21};
45.     int n = arr.length;
46.     int i;
47.     boolean order = true;
48.     sort(arr, n, order);
49.
50.     System.out.println("Sorted array: \n");
51.     for (i=0; i<n; i++)
52.         System.out.println(arr[i]);
53. }
54. }
```

Output:

```
Sorted array:

1       1       2       3       10      21      23      45
```

C#

```
1.  //this program works when size of input is power of 2.
2.  using System;
3.  public class BitonicSort
4.  {
5.  static void exchange(int[] arr, int i, int j, bool d)
6.  {
7.      int temp;
8.      if (d==(arr[i]>arr[j]))
9.      {
10.         temp = arr[i];
11.         arr[i] = arr[j];
12.         arr[j] = temp;
13.     }
14. }
15. static void merge(int[] arr, int l, int c, bool d)
16. {
17.     int k,i;
18.     if (c>1)
19.     {
20.         k = c/2;
21.         for (i=l; i<l+k; i++)
```

```
22.             exchange(arr, i, i+k, d);
23.         merge(arr, l, k, d);
24.         merge(arr, l+k, k, d);
25.     }
26. }
27. static void bitonicSort(int[] arr,int l, int c, bool d)
28. {
29.     int k;
30.     if (c>1)
31.     {
32.         k = c/2;
33.         bitonicSort(arr, l, k, true);
34.         bitonicSort(arr, l+k, k, false);
35.         merge(arr,l, c, d);
36.     }
37. }
38.
39. static void sort(int[] arr, int n, bool order)
40. {
41.     bitonicSort(arr,0, n, order);
42. }
43. public void Main()
44. {
45.     int[] arr= {1, 10, 2, 3, 1, 23, 45, 21};
46.     int n = arr.Length;
47.     int i;
48.     bool order = true;
49.     sort(arr, n, order);
50.
51.     Console.WriteLine("Sorted array: \n");
52.     for (i=0; i<n; i++)
53.         Console.WriteLine(arr[i]+"   ");
54. }
55. }
```

Cocktail Sort

Cocktail sort is the variation of Bubble Sort which traverses the list in both directions alternatively. It is different from bubble sort in the sense that, bubble sort traverses the list in forward direction only, while this algorithm traverses in forward as well as backward direction in one iteration.

Algorithm

In cocktail sort, one iteration consists of two stages:

1. The first stage loop through the array like bubble sort from left to right. The adjacent elements are compared and if left element is greater than the right element, then we swap those elements. The largest element of the list is placed at the end of the array in the forward pass.
2. The second stage loop through the array from the right most unsorted element to the left. The adjacent elements are compared and if right element is smaller than the left element then, we swap those elements. The smallest element of the list is placed at the beginning of the array in backward pass.

The process continues until the list becomes unsorted.

Example

Consider the array A : {8, 2, 3, 1, 9}. Sort the elements of the array using Cocktail sort.

Iteration 1:

Forward pass :

1. compare 8 with 2; 8>2 → swap(8,2)
2.
3. A={2,8,3,1,9}
4.
5. Compare 8 with 3; 8>3 → swap(8,3)
6.
7. A={2,3,8,1,9}
8.
9. Compare 8 with 1; 8 > 1 → swap(8,1)
10.
11. A = {2,3,1,8,9}
12.
13. Compare 8 with 9; 8<9 → **do** not swap

At the end of first forward pass: the largest element of the list is placed at the end.

1. A={2, 3, 1, 8, 9 }

Backward pass:

1. compare 8 with 1; 8> 1 → **do** not swap
2.
3. A={2, 3, 1, 8, 9 }
4. compare 1 with 3 ; 3>1 → swap(1,3)
5.
6. A={2, 1, 3, 8, 9 }
7.
8. compare 1 with 2 ; 2> 1 → swap(1,2)
9.
10. A={1, 2, 3, 8, 9}

At the end of first backward pass; the smallest element has been placed at the first index of the array.

If we have a look at the list produced in the first step; we can find that the list has already been sorted in ascending order but the algorithm will process itself completely.

Complexity

Complexity	Best Case	Average Case	Worst Case
Time Complexity	$O(n^2)$	$O(n^2)$	$O(n^2)$
Space Complexity			O(1)

C Program

```c
#include <stdio.h>
int temp;
void Cocktail(int a[], int n)
{
    int is_swapped = 1;
    int begin = 0,i;
    int end = n - 1;

    while (is_swapped) {
        is_swapped = 0;
     for (i = begin; i < end; ++i) {
            if (a[i] > a[i + 1]) {
             temp = a[i];
        a[i]=a[i+1];
         a[i+1]=temp;
```

```
        is_swapped = 1;
            }
        }
if (!is_swapped)
            break;
is_swapped = 0;
for (i = end - 1; i >= begin; --i) {
    if (a[i] > a[i + 1])
   {
          temp = a[i];
     a[i]=a[i+1];
     a[i+1]=temp;
     is_swapped = 1;
        }
     }
       ++begin;
   }
}

int main()
{
    int arr[] = {0, 10, 2, 8, 23, 1, 3, 45},i;
    int n = sizeof(arr) / sizeof(arr[0]);
    Cocktail(arr, n);
    printf("printing sorted array :\n");
     for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
    return 0;
}
```

Output:

```
printing sorted array :
0 1 2 3 8 10 23 45
```

C++ Program

```
#include <iostream>
using namespace std;
int temp;
void Cocktail(int a[], int n)
{
    int is_swapped = 1;
    int begin = 0,i;
    int end = n - 1;

    while (is_swapped) {
        is_swapped = 0;
     for (i = begin; i < end; ++i) {
            if (a[i] > a[i + 1]) {
           temp = a[i];
       a[i]=a[i+1];
      a[i+1]=temp;
       is_swapped = 1;
          }
       }
if (!is_swapped)
            break;
```

```
22.  is_swapped = 0;
23.  for (i = end - 1; i >= begin; --i) {
24.      if (a[i] > a[i + 1])
25.     {
26.          temp = a[i];
27.       a[i]=a[i+1];
28.       a[i+1]=temp;
29.       is_swapped = 1;
30.         }
31.       }
32.         ++begin;
33.     }
34. }
35.
36. int main()
37. {
38.     int arr[] = {0, 10, 2, 8, 23, 1, 3, 45},i;
39.     int n = sizeof(arr) / sizeof(arr[0]);
40.     Cocktail(arr, n);
41.     cout <<"printing sorted array :\n";
42.      for (i = 0; i < n; i++)
43.         cout<<arr[i]<<" ";
44.     cout<<"\n";
45.     return 0;
46. }
```

Output:

```
printing sorted array :
0 1 2 3 8 10 23 45
```

Java Program

```
1.  public class CocktailSort
2.  {
3.  static int temp;
4.  static void Cocktail(int a[], int n)
5.  {
6.      boolean is_swapped = true;
7.      int begin = 0,i;
8.      int end = n - 1;
9.
10.     while (is_swapped) {
11.         is_swapped = false;
12.      for (i = begin; i < end; ++i) {
13.             if (a[i] > a[i + 1]) {
14.             temp = a[i];
15.         a[i]=a[i+1];
16.         a[i+1]=temp;
17.         is_swapped = true;
18.             }
19.         }
20.  if (!is_swapped)
21.             break;
22.  is_swapped = false;
23.  for (i = end - 1; i >= begin; --i) {
24.      if (a[i] > a[i + 1])
25.     {
26.          temp = a[i];
```

```
27.       a[i]=a[i+1];
28.       a[i+1]=temp;
29.       is_swapped = true;
30.         }
31.       }
32.         ++begin;
33.     }
34. }
35. public static void main(String[] args) {
36.
37.     int arr[] = {0, 10, 2, 8, 23, 1, 3, 45},i;
38.     int n = arr.length;
39.     Cocktail(arr, n);
40.     System.out.println("printing sorted array :\n");
41.      for (i = 0; i < n; i++)
42.         System.out.print(arr[i]+" ");
43.     System.out.println();
44.     }
45. }
```

Output:

```
printing sorted array :

0 1 2 3 8 10 23 45
```

C# Program

```
1.  using System;
2.  public class CocktailSort
3.  {
4.  static int temp;
5.  static void Cocktail(int[] a, int n)
6.  {
7.      Boolean is_swapped = true;
8.      int begin = 0,i;
9.      int end = n - 1;
10.
11.     while (is_swapped) {
12.         is_swapped = false;
13.      for (i = begin; i < end; ++i) {
14.             if (a[i] > a[i + 1]) {
15.                 temp = a[i];
16.         a[i]=a[i+1];
17.          a[i+1]=temp;
18.         is_swapped = true;
19.             }
20.         }
21. if (!is_swapped)
22.             break;
23. is_swapped = false;
24. for (i = end - 1; i >= begin; --i) {
25.     if (a[i] > a[i + 1])
26.    {
27.           temp = a[i];
28.       a[i]=a[i+1];
29.       a[i+1]=temp;
30.       is_swapped = true;
31.          }
32.       }
```

```
        ++begin;
    }
}
public void Main() {

    int[] arr = {0, 10, 2, 8, 23, 1, 3, 45};
    int n = arr.Length,i;
    Cocktail(arr, n);
    Console.WriteLine("printing sorted array :\n");
     for (i = 0; i < n; i++)
        Console.Write(arr[i]+" ");

    }
```

Output:

```
printing sorted array :

0 1 2 3 8 10 23 45
```

Python Program

```
def Cocktail(a,n):
    is_swapped = True;
    begin = 0
    end = n - 1;
    while is_swapped:
        is_swapped = False;
        for i in range(begin,end):
            if a[i] > a[i + 1]:
                temp = a[i];
                a[i]=a[i+1];
                a[i+1]=temp;
                is_swapped = True;
        if not(is_swapped):
            break;
        is_swapped = False;
        for i in range(end-1,begin-1,-1):
            if a[i] > a[i + 1]:
                temp = a[i];
                a[i]=a[i+1];
                a[i+1]=temp;
                is_swapped = True;
        ++begin;
arr = [0, 10, 2, 8, 23, 1, 3, 45];
n = len(arr);
Cocktail(arr, n);
print("printing sorted array :\n");
for i in range(0,n):
    print(arr[i]),
```

Output:

```
printing sorted array :

0 1 2 3 8 10 23 45
```

Rust Program

```
fn main()
{
    let mut a :[i32;8] = [0, 10, 2, 8, 23, 1, 3, 45];
```

```
4.      let mut is_swapped = true;
5.       let mut  begin = 0;
6.       let end = 7;
7.
8.       while is_swapped {
9.           is_swapped = false;
10.       for i in begin..end {
11.             if a[i] > a[i + 1] {
12.             let mut temp = a[i];
13.              a[i]=a[i+1];
14.             a[i+1]=temp;
15.              is_swapped = true;
16.             }
17.         }
18.     if !is_swapped
19.     {
20.             break;
21.     }
22.     is_swapped = false;
23.     for i in (begin..(end-1)).rev()
24.     {
25.         if a[i] > a[i + 1]
26.         {
27.           let mut temp = a[i];
28.          a[i]=a[i+1];
29.           a[i+1]=temp;
30.          is_swapped =true;
31.          }
32.       }
33.      begin=begin+1;
34.     }
35.     print!("printing sorted array :\n");
36.     for i in 0..7
37.     {
38.         print!("{}",a[i]);
39.     }
40. }
```

Output:

```
printing sorted array :
0 1 2 3 8 10 23 45
```

JavaScript Program

```
1.  <html>
2.  <head>
3.      <title>Cocktail Sort</title>
4.      </head>
5.          <body>
6.              <script>
7.
8.  function Cocktail( a, n)
9.  {
10.     var temp=0;
11.     var is_swapped = 1;
12.     var begin = 0;
13.     var end = n - 1;
14.
```

```
15.    while (is_swapped) {
16.       is_swapped = 0;
17.     for (i = begin; i < end; ++i) {
18.           if (a[i] > a[i + 1]) {
19.            temp = a[i];
20.       a[i]=a[i+1];
21.        a[i+1]=temp;
22.       is_swapped = 1;
23.           }
24.       }
25.    if (!is_swapped)
26.           break;
27.    is_swapped = 0;
28.    for (i = end - 1; i >= begin; --i) {
29.        if (a[i] > a[i + 1])
30.    {
31.          temp = a[i];
32.     a[i]=a[i+1];
33.        a[i+1]=temp;
34.       is_swapped = 1;
35.          }
36.       }
37.        beginbegin=begin+1;
38.     }
39. }
40.
41.     var txt = "<br>";
42.     var arr = [0, 10, 2, 8, 23, 1, 3, 45];
43.     var n = arr.length;
44.     Cocktail(arr, n);
45.     document.writeln("printing sorted array :\n"+txt);
46.      for (i = 0; i < n; i++)
47.      {
48.          document.writeln(arr[i]+"/xa0");
49.      }
50. </script>
51. </body>
52. </html>
```

Output:

```
printing sorted array :
0
1 2 3 8 10 23 45
```

Cycle Sort

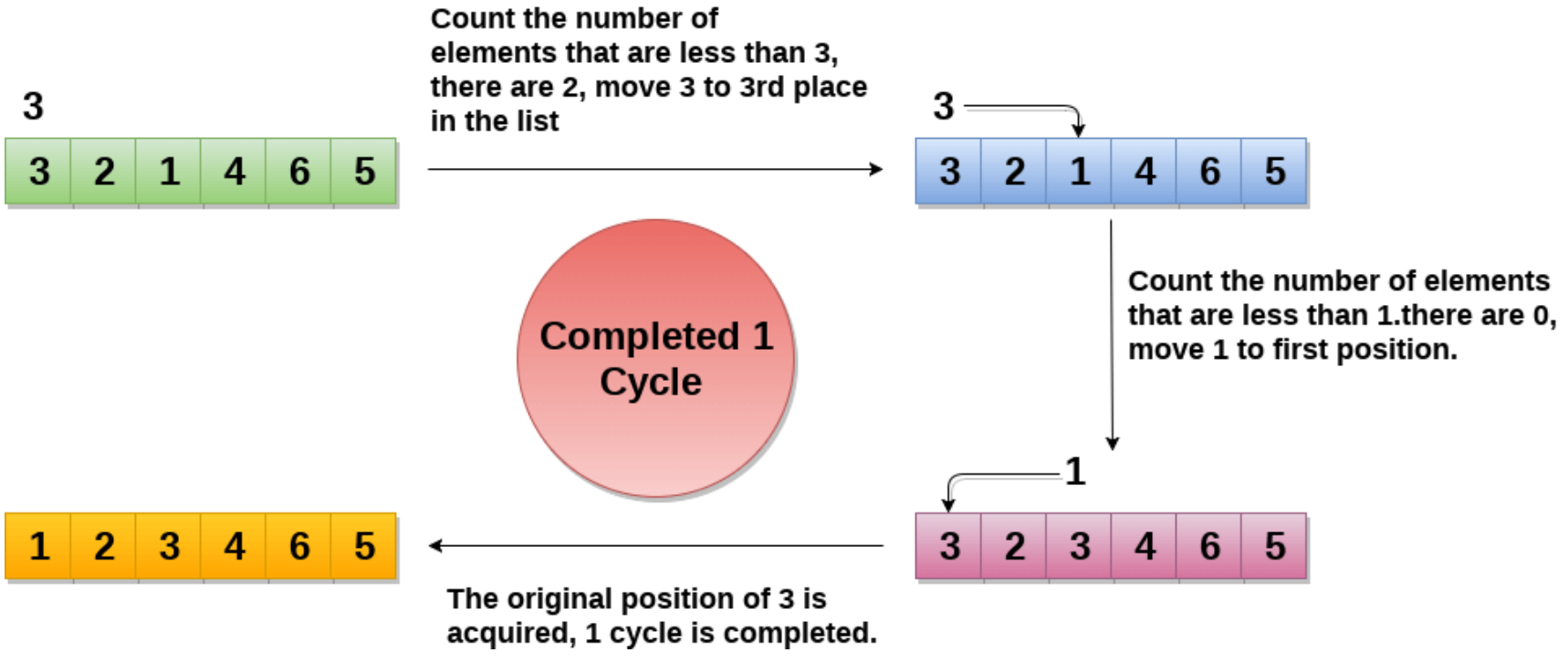
Cycle sort is a comparison sorting algorithm which forces array to be factored into the number of cycles where each of them can be rotated to produce a sorted array. It is theoretically optimal in the sense that it reduces the number of writes to the original array.

Algorithm

Consider an array of n distinct elements. An element a is given, index of a can be calculated by counting the number of elements that are smaller than a.

1. if the element is found to be at its correct position, simply leave it as it is.
2. Otherwise, find the correct position of **a** by counting the total number of elements that are less than **a**. where it must be present in the sorted array. The other element b which is replaced is to be moved to its correct position. This process continues until we got an element at the original position of **a**.

The illustrated process constitutes a cycle. Repeating this cycle for each element of the list. The resulting list will be sorted.



C program

```c
#include<stdio.h>
void sort(int a[], int n)
{
    int writes = 0,start,element,pos,temp,i;

    for (start = 0; start <= n - 2; start++) {
        element = a[start];
        pos = start;
        for (i = start + 1; i < n; i++)
            if (a[i] < element)
                pos++;
        if (pos == start)
            continue;
        while (element == a[pos])
            pos += 1;
        if (pos != start) {
            //swap(element, a[pos]);
        temp = element;
        element = a[pos];
            a[pos] = temp;
            writes++;
        }
        while (pos != start) {
            pos = start;
            for (i = start + 1; i < n; i++)
                if (a[i] < element)
                    pos += 1;
            while (element == a[pos])
                pos += 1;
            if (element != a[pos]) {
              temp = element;
          element = a[pos];
              a[pos] = temp;
                writes++;
            }
        }
    }

}
```

```
40.
41. int main()
42. {
43.     int a[] = {1, 9, 2, 4, 2, 10, 45, 3, 2};
44.     int n = sizeof(a) / sizeof(a[0]),i;
45.     sort(a, n);
46.     printf("After sort, array : ");
47.     for (i = 0; i < n; i++)
48.         printf("%d  ",a[i]);
49.     return 0;
50. }
```

Output:

```
After sort, array :
1
2
2
2
3
4
9
10
45
```

Java Program

```
1.  public class CycleSort
2.  {
3.  static void sort(int a[], int n)
4.  {
5.      int writes = 0,start,element,pos,temp,i;
6.
7.      for (start = 0; start <= n - 2; start++) {
8.          element = a[start];
9.          pos = start;
10.         for (i = start + 1; i < n; i++)
11.             if (a[i] < element)
12.                 pos++;
13.         if (pos == start)
14.             continue;
15.         while (element == a[pos])
16.             pos += 1;
17.       if (pos != start) {
18.           //swap(element, a[pos]);
19.       temp = element;
20.       element = a[pos];
21.           a[pos] = temp;
22.           writes++;
23.       }
24.       while (pos != start) {
25.           pos = start;
26.           for (i = start + 1; i < n; i++)
27.               if (a[i] < element)
28.                   pos += 1;
29.           while (element == a[pos])
30.               pos += 1;
31.           if (element != a[pos]) {
32.             temp = element;
```

```
        element = a[pos];
            a[pos] = temp;
              writes++;
          }
      }
  }
}
public static void main(String[] args)
{
  int a[] = { 1, 9, 2, 4, 2, 10, 45, 3, 2 };
  int n = a.length,i;
  sort(a, n);
  System.out.println("After sort, array : ");
  for (i = 0; i < n; i++)
      System.out.println(a[i]);

}
}
```

Output:

```
After sort, array :
1
2
2
2
3
4
9
10
45
```

C# Program

```
using System;
public class CycleSort
{
static void sort(int[] a, int n)
{
    int writes = 0,start,element,pos,temp,i;

    for (start = 0; start <= n - 2; start++) {
        element = a[start];
        pos = start;
        for (i = start + 1; i < n; i++)
            if (a[i] < element)
                pos++;
        if (pos == start)
            continue;
        while (element == a[pos])
            pos += 1;
        if (pos != start) {
            //swap(element, a[pos]);
        temp = element;
        element = a[pos];
            a[pos] = temp;
            writes++;
        }
        while (pos != start) {
            pos = start;
            for (i = start + 1; i < n; i++)
                if (a[i] < element)
```

```
                pos += 1;
          while (element == a[pos])
             pos += 1;
          if (element != a[pos]) {
            temp = element;
        element = a[pos];
            a[pos] = temp;
              writes++;
          }
      }
  }
}
public void Main()
{
   int[] a = { 1, 9, 2, 4, 2, 10, 45, 3, 2  };
   int n = a.Length,i;
   sort(a, n);
   Console.WriteLine("After sort, array : ");
   for (i = 0; i < n; i++)
       Console.WriteLine(a[i]);

}
}
```

Output:

```
After sort, array :
1
2
2
2
3
4
9
10
45
```

Tim-sort

Tim-sort is a sorting algorithm derived from insertion sort and merge sort. It was designed to perform in an optimal way on different kind of real world data. It is an adaptive sorting algorithm which needs O(n log n) comparisons to sort an array of n elements. It was designed and implemented by Tim Peters in 2002 in python programming language. It has been python's standard sorting algorithm since version 2.3.

Technique

Consider an array of n elements which needs to be sorted. In Tim sort, the array is divided into several parts where each of them is called a Run. These Runs are sorted by using insertion sort one by one and then the sorted runs get combined using a combine function. The idea of Tim sort is originated from the fact that, insertion sort works more optimally on the short lists rather than working on the larger lists.

Complexity

Complexity	Best Case	Average Case	Worst Case
Time Complexity	O(n)	O(n log n)	O(n log n)
Space Complexity			n

Steps :

1. Divide the array into the number of blocks known as run.
2. Consider size of run either 32 or 64(in the below implementation, size of run is 32.)
3. Sort the individual elements of every run one by one using insertion sort.
4. Merge the sorted runs one by one using merge function of merge sort.
5. Double the size of merged sub-arrays after every iteration.

C program

```
#include<stdio.h>
const int run = 32;
int minimum(int a, int b)
{
    if(a<b)
    return a;
    else
    return b;
}
void insertionSort(int a[], int beg, int end)
{
   int temp, i, j;
   for (i = beg + 1; i <= end; i++)
   {
       temp = a[i];
      j = i - 1;
       while (a[j] > temp && j >= beg)
      {
          a[j+1] = a[j];
         j--;
      }
      a[j+1] = temp;
   }
}

void merge(int a[], int left, int mid, int right)
{
   int len1 = mid - left + 1, len2 = right - mid;
   int beg[len1], end[len2];
   int i,j,k;
   for (i = 0; i < len1; i++)
      beg[i] = a[left + i];
   for (i = 0; i < len2; i++)
      end[i] = a[mid + 1 + i];

i = 0;
j = 0;
k = left;

while (i < len1 && j < len2)
{
   if (beg[i] <= end[j])
   {
      a[k] = beg[i];
       i++;
   }
   else
   {
       a[k] = end[j];
```

```
            j++;
        }
        k++;
    }
    while (i < len1)
    {
        a[k] = beg[i];
        k++;
        i++;
    }

    while (j < len2)
    {
        a[k] = end[j];
        k++;
        j++;
    }
}
void timSort(int a[], int n)
{
    int i,size,beg,mid,end;
    for (i = 0; i < n; i+=run)
        insertionSort(a, i, minimum((i+31), (n-1)));
    for (size = run; size < n; size = 2*size)
    {
        for (beg = 0; beg < n; beg += 2*size)
        {
            mid = beg + size - 1;
            end = minimum((beg + 2*size - 1), (n-1));

            merge(a, beg, mid, end);
        }
    }
}

int main()
{
    int a[] = {12,1,20,2,3,123,54,332},i;
    int n = sizeof(a)/sizeof(a[0]);
    printf("Printing Array elements \n");
    for (i = 0; i < n; i++)
    printf("%d  ", a[i]);
    printf("\n");
    timSort(a, n);

    printf("Printing sorted array elements \n");
    for (i = 0; i < n; i++)
        printf("%d  ", a[i]);
    printf("\n");
    return 0;
        }
```

Output:

```
Printing Array elements
12  1  20  2  3  123  54  332

Printing sorted array elements
1  2  3  12  20  54  123  332
```