PW SKILLS

Lesson Plan

Cyclic Sort Algorithm

PW SKILLS

Cycle sort is an in-place, unstable sorting algorithm that is particularly useful when sorting arrays containing elements with a small range of values. The basic idea behind cycle sort is to divide the input array into cycles, where each cycle consists of elements that belong to the same position in the sorted output array. The algorithm then performs a series of swaps to place each element in its correct position within its cycle, until all cycles are complete and the array is sorted.

Here's a step-by-step explanation of the cycle sort algorithm:

1. Start with an unsorted array of n elements.
2. Initialize a variable, cycleStart, to 0.
3. For each element in the array, compare it with every other element to its right. If there are any elements that are smaller than the current element, increment cycleStart.
4. If cycleStart is still 0 after comparing the first element with all other elements, move to the next element and repeat step 3.
5. Once a smaller element is found, swap the current element with the first element in its cycle. The cycle is then continued until the current element returns to its original position.

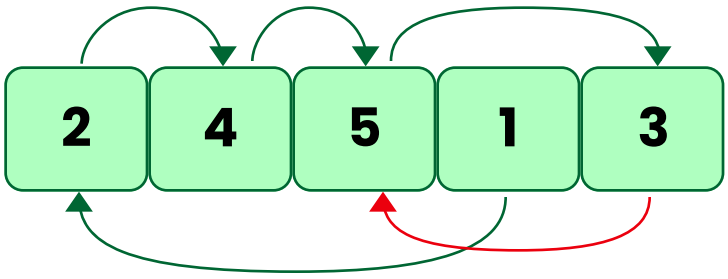
Repeat steps 3-5 until all cycles have been completed.

The array is now sorted.

One of the advantages of cycle sort is that it has a low memory footprint, as it sorts the array in-place and does not require additional memory for temporary variables or buffers. However, it can be slow in certain situations, particularly when the input array has a large range of values. Nonetheless, cycle sort remains a useful sorting algorithm in certain contexts, such as when sorting small arrays with limited value ranges.

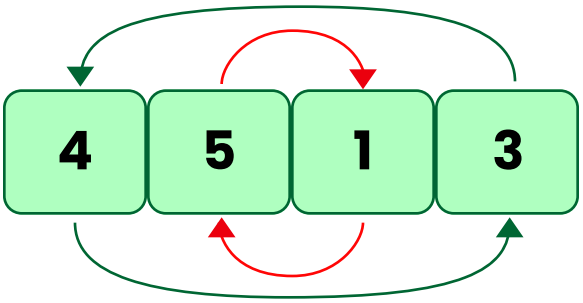
Cycle sort is an in-place sorting Algorithm, unstable sorting algorithm, and a comparison sort that is theoretically optimal in terms of the total number of writes to the original array.

- It is optimal in terms of the number of memory writes. It minimizes the number of memory writes to sort (Each value is either written zero times if it's already in its correct position or written one time to its correct position.)
- It is based on the idea that the array to be sorted can be divided into cycles. Cycles can be visualized as a graph. We have n nodes and an edge directed from node i to node j if the element at i-th index must be present at j-th index in the sorted array. Cycle in arr[] = {2, 4, 5, 1, 3}



Cycle in arr[] = {2, 4, 5, 1, 3}

- Cycle in arr[] = {4, 3, 2, 1}



Cycle in arr[] = {4, 3, 2, 1}
We consider all cycles one by one. We first consider the cycle that includes the first element. We find the correct position of the first element, and place it at its correct position, say j. We consider the old value of arr[j] and find its correct position, we keep doing this till all elements of the current cycle are placed at the correct position, i.e., we don't come back to the cycle starting point.

Pseudocode :

```
Begin
for
start:= 0 to n - 2 do
key := array[start]
location := start
for i:= start + 1 to n-1 do
  if array[i] < key then
    location: =location +1
done
if location = start then
   ignore lower part, go for next iteration
while key = array[location] do
  location: = location + 1
done
if location != start then
   swap array[location] with key
while location != start do
   location start

for i:= start + 1 to n-1 do
    if array[i] < key then
        location: =location +1
done
```

```
while key= array[location]
     location := location +1
 if key != array[location]
     Swap array[location] and key
   done
 done
End
```

Explanation :

arr[] = {10, 5, 2, 3}
index = 0 1 2 3
cycle_start = 0
item = 10 = arr[0]

Find position where we put the item

```
pos = cycle_start
i=pos+1
while(i<n)
if (arr[i] < item)
   pos++;
```

We put 10 at arr[3] and change item to
old value of arr[3].
arr[] = {10, 5, 2, 10}
item = 3

Again rotate rest cycle that start with index '0'
Find position where we put the item = 3
we swap item with element at arr[1] now
arr[] = {10, 3, 2, 10}
item = 5

Again rotate rest cycle that start with index '0' and item = 5
we swap items with elements at arr[2].
arr[] = {10, 3, 5, 10 }
item = 2

Again rotate rest cycle that start with index '0' and item = 2
arr[] = {2, 3, 5, 10}

Above is one iteration for cycle_stat = 0.
Repeat above steps for cycle_start = 1, 2, ..n-2
Below is the implementation of the above approach:

Code:

```
import java.util.*;
import java.lang.*;

class Main{
    // Function sort the array using Cycle sort
    public static void cycleSort(int arr[], int n)
    {
        // count number of memory writes
        int writes = 0;

        // traverse array elements and put it to on
        // the right place
        for (int cycle_start = 0; cycle_start ⩽ n - 2;
cycle_start++) {
            // initialize item as starting point
            int item = arr[cycle_start];

            // Find the position where we put the item. We
basically
            // count all smaller elements on the right side
of the item.
            int pos = cycle_start;
            for (int i = cycle_start + 1; i < n; i++)
                if (arr[i] < item)
                    pos++;

            // If item is already in correct position
            if (pos == cycle_start)
                continue;

            // ignore all duplicate elements
            while (item == arr[pos])
                pos += 1;

            // put the item to it's right position
            if (pos ≠ cycle_start) {
                int temp = item;
                item = arr[pos];
                arr[pos] = temp;
                writes++;
            }

            // Rotate rest of the cycle
            while (pos ≠ cycle_start) {
                pos = cycle_start;

                // Find position where we put the element
                for (int i = cycle_start + 1; i < n; i++)
                    if (arr[i] < item)
                        pos += 1;
```

```
                // ignore all duplicate elements
                while (item == arr[pos])
                    pos += 1;

                // put the item to it's right position
                if (item != arr[pos]) {
                    int temp = item;
                    item = arr[pos];
                    arr[pos] = temp;
                    writes++;
                }
            }
        }
    }
    // Driver program to test above function
    public static void main(String[] args)
    {
        int arr[] = { 1, 8, 3, 9, 10, 10, 2, 4 };
        int n = arr.length;
        cycleSort(arr, n);

        System.out.println("After sort : ");
        for (int i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
    }
}
```

Output

After sort :

1 2 3 4 8 9 10 10

Time Complexity Analysis:

- Worst Case: O(n2)
- Average Case: O(n2)
- Best Case: O(n2)

Auxiliary Space: O(1)

- The space complexity is constant because this algorithm is in place so it does not use any extra memory to sort.

Method 2: This method is only applicable when given array values or elements are in the range of 1 to N or 0 to N. In this method, we do not need to rotate an array

Approach : All the given array values should be in the range of 1 to N or 0 to N. If the range is 1 to N then every array element's correct position will be the index == value-1 i.e. means at the 0th index value will be 1 similarly at the 1st index position value will be 2 and so on till nth value.

Similarly for 0 to N values the correct index position of each array element or value will be the same as its value i.e. at 0th index 0 will be there 1st position 1 will be there.

Explanation :

arr[] = {5, 3, 1, 4, 2}
index = 0 1 2 3 4

i = 0;
while(i < arr.length)
correctposition = arr[i]-1;

find ith item correct position
for the first time i = 0 arr[0] = 5 correct index of 5 is 4 so arr[i] - 1 = 5-1 = 4

if(arr[i] <= arr.length && arr[i] != arr[correctposition])

arr[i] = 5 and arr[correctposition] = 4
so 5 <= 5 && 5 != 4 if condition true
now swap the 5 with 4

int temp = arr[i];
arr[i] = arr[correctposition];
arr[correctposition] = temp;

now resultant arr at this after 1st swap
arr[] = {2, 3, 1, 4, 5} now 5 is shifted at its correct position

now loop will run again check for i = 0 now arr[i] is = 2
after swapping 2 at its correct position
arr[] = {3, 2, 1, 4, 5}

now loop will run again check for i = 0 now arr[i] is = 3
after swapping 3 at its correct position
arr[] = {1, 2, 3, 4, 5}

now loop will run again check for i = 0 now arr[i] is = 1
this time 1 is at its correct position so else block will execute and i will increment i = 1;
once i exceeds the size of array will get array sorted.
arr[] = {1, 2, 3, 4, 5}

else

i++;
loop end;

once while loop end we get sorted array just print it
for(index = 0 ; index < arr.length; index++)
  print(arr[index] + " ")
sorted arr[] = {1, 2, 3, 4, 5}

```
// java program to check implement cycle sort
import java.util.*;
public class MissingNumber {
    public static void main(String[] args)
    {
        int[] arr = { 3, 2, 4, 5, 1 };
        int n = arr.length;
        System.out.println("Before sort :");
        System.out.println(Arrays.toString(arr));
        CycleSort(arr, n);

    }

    static void CycleSort(int[] arr, int n)
    {
        int i = 0;
        while (i < n) {
            // as array is of 1 based indexing so the
            // correct position or index number of each
            // element is element-1 i.e. 1 will be at 0th
            // index similarly 2 correct index will 1 so
            // on...
            int correctpos = arr[i] - 1;
            if (arr[i] < n && arr[i] != arr[correctpos]) {
                // if array element should be lesser than
                // size and array element should not be at
                // its correct position then only swap with
                // its correct position or index value
                swap(arr, i, correctpos);
            }
            else {
                // if element is at its correct position
                // just increment i and check for remaining
                // array elements
                i++;
            }
        }
            System.out.println("After sort :  ");
            System.out.print(Arrays.toString(arr));


    }

    static void swap(int[] arr, int i, int correctpos)
```

```
    {
    // swap elements with their correct indexes
        int temp = arr[i];
        arr[i] = arr[correctpos];
        arr[correctpos] = temp;
    }
}
```

Output
Before sorting array:

3 2 4 5 1

Sorted array:

1 2 3 4 5

Time Complexity Analysis:

- Worst Case : O(n)
- Average Case: O(n)
- Best Case : O(n)

Auxiliary Space: O(1)

Advantage of Cycle sort:

1. No additional storage is required.
2. in-place sorting algorithm.
3. A minimum number of writes to the memory
4. Cycle sort is useful when the array is stored in EEPROM or FLASH.

Disadvantage of Cycle sort:

1. It is not mostly used.
2. It has more time complexity o(n^2)
3. Unstable sorting algorithm.

Application of Cycle sort:

- This sorting algorithm is best suited for situations where memory write or swap operations are costly.
- Useful for complex problems.