Sorting Applications

SWE2016-44

K’th Smallest Element in Unsorted Array

- Given an array and a number k where k is smaller than size of array, we need to find the k’th smallest element in the given array.

- Example

Input: *arr[] = {7, 10, 4, 3, 20, 15}*

k = 3

Output: *7*

Input: *arr[] = {7, 10, 4, 3, 20, 15}*

k = 4

Output: *10*

K’th Smallest Element in Unsorted Array

- Method 1: Simple Solution
  - A simple solution is to sort the given array using a O(N log N) sorting algorithm like Merge Sort, Heap Sort, etc and return the element at index k-1 in the sorted array.
  - Time Complexity of this solution is **O(N Log N)**

K’th Smallest Element in Unsorted Array

- Method 2: Using **Min Heap** – HeapSelect
  - Create Min Heap → O(N)
  - Call heapify() k times → O(k Log N)
  - Time Complexity of this solution is **O(N + k Log N)**

K’th Smallest Element in Unsorted Array

- Method 3: Using **Max Heap**

1. Build a Max-Heap MH of the first k elements (arr[0] to arr[k-1]) of the given array → O(k)
2. For each element, after the k’th element (arr[k] to arr[N-1]), compare it with root of MH. → O((N-k)*logk)
   - If the element is less than the root then make it root and call heapify for MH. Else ignore it.
3. Finally, root of the MH is the kth smallest element.

→Time Complexity of this solution is **O(k + (N-k)*logk)**.

K’th Smallest Element in Unsorted Array

- Method 4: QuickSelect
  - In **QuickSort**, we pick a pivot element, then move the pivot element to its correct position and partition the array around it. The idea is, not to do complete quicksort, but stop at the point where pivot itself is k’th smallest element. Also, not to recur for both left and right sides of pivot, but recur for one of them according to the position of pivot.
  - The worst case time complexity of this method is **$O(n^2)$.**
  - **Randomized QuickSelect**: expected worst case time complexity is **O(n). → Expected Linear Time**

Count Inversions in an array

- Inversion Count for an array indicates – how far (or close) the array is from being sorted. If array is already sorted then inversion count is 0. If array is sorted in reverse order that inversion count is the maximum. Formally speaking, two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j.

  - **Example:** The sequence 2, 4, 1, 3, 5 has three inversions (2, 1), (4, 1), (4, 3).

Count Inversions in an array

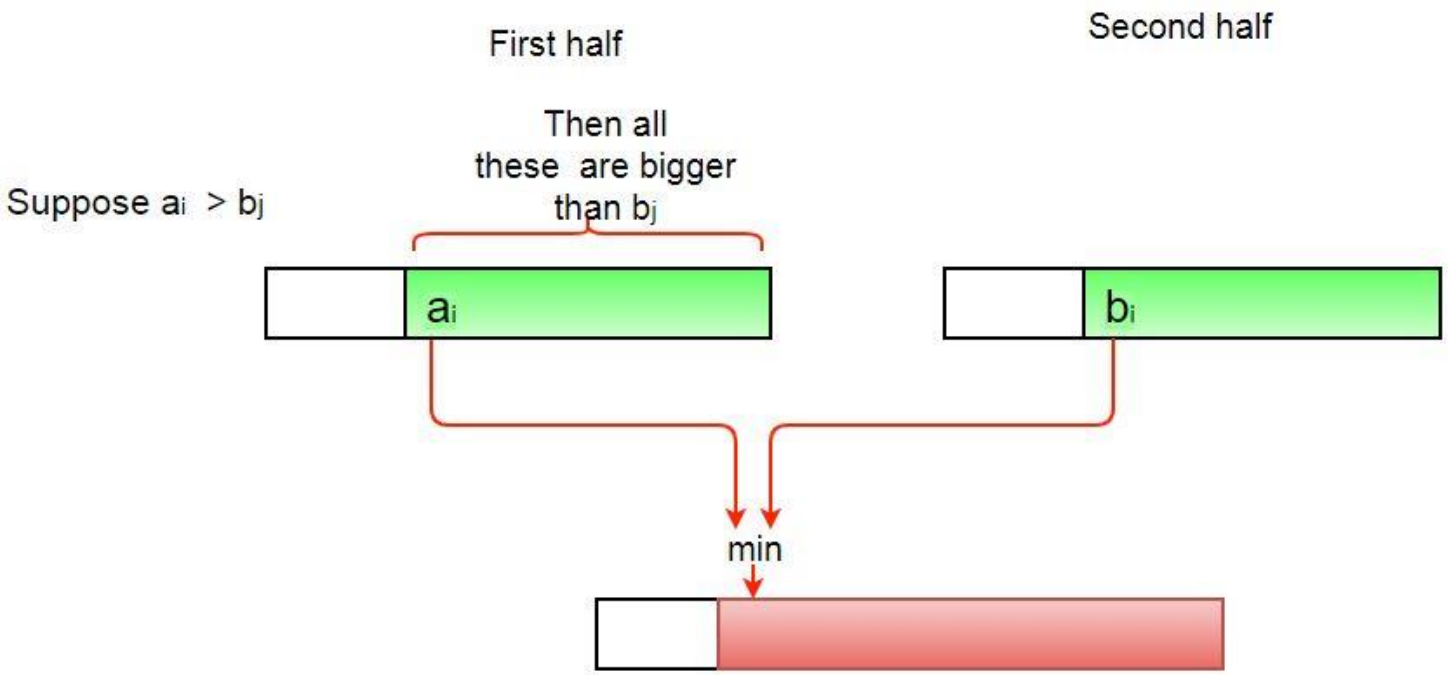
- **METHOD 1 (Simple)**
  - For each element, count number of elements which are on right side of it and are smaller than it.
  - **Time Complexity:** $\mathbf{O(n^2)}$

Count Inversions in an array

- **METHOD 2 (Enhance Merge Sort)**

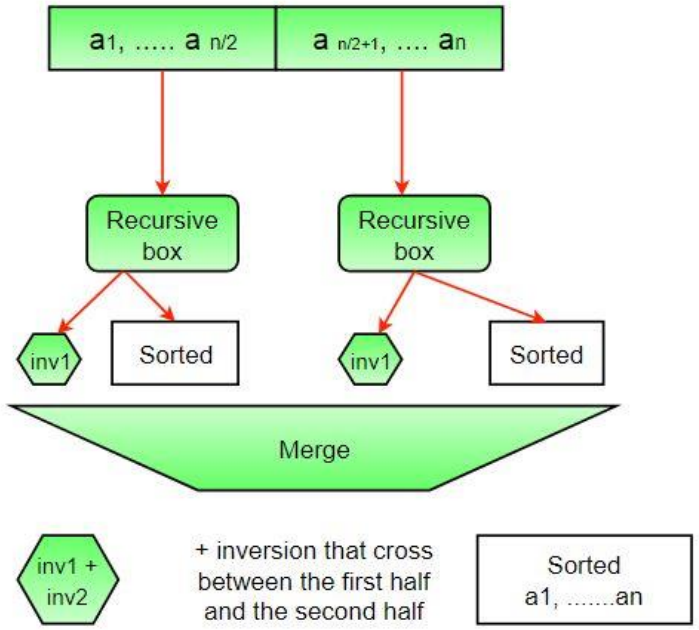
Count Inversions in an array

- **METHOD 2 (Enhance Merge Sort)**
  - In merge process,



Count Inversions in an array

- **METHOD 2 (Enhance Merge Sort)**
  - The complete picture:



→ **Time Complexity:** O(N log N)
→ **Algorithmic Paradigm:** Divide and Conquer

Find maximum meetings in one room

- There is one meeting room in a firm. There are N meetings in the form of (S[i], F[i]) where S[i] is the start time of meeting i and F[i] is finish time of meeting i. The task is to find the maximum number of meetings that can be accommodated in the meeting room. Print all meeting numbers.

Input : *s[] = {1, 3, 0, 5, 8, 5}, f[] = {2, 4, 6, 7, 9, 9}*
Output : *1 2 4 5*
First meeting [1, 2]
Second meeting [3, 4]
Fourth meeting [5, 7]
Fivth meeting [8, 9]

Input : *s[] = {75250, 50074, 43659, 8931, 11273, 27545, 50879, 77924},*
f[] = {112960, 114515, 81825, 93424, 54316, 35533, 73383, 160252 }
Output : *6 7 1*

Find maximum meetings in one room

1. Sort all pairs(Meetings) in increasing order of second number(Finish time) of each pair.

Find maximum meetings in one room

1. Sort all pairs(Meetings) in increasing order of second number(Finish time) of each pair.
2. Select first meeting of sorted pair as the first Meeting in the room and push it into result vector and set a variable time_limit(say) with the second value(Finishing time) of the first selected meeting.

Find maximum meetings in one room

1. Sort all pairs(Meetings) in increasing order of second number(Finish time) of each pair.
2. Select first meeting of sorted pair as the first Meeting in the room and push it into result vector and set a variable time_limit(say) with the second value(Finishing time) of the first selected meeting.
3. Iterate from the second pair to last pair of the array and if the value of the first element(Starting time of meeting) of the current pair is greater then previously selected pair finish time (time_limit) then select the current pair and update the result vector (push selected meeting number into vector) and variable time_limit.

Find maximum meetings in one room

1. Sort all pairs(Meetings) in increasing order of second number(Finish time) of each pair.
2. Select first meeting of sorted pair as the first Meeting in the room and push it into result vector and set a variable time_limit(say) with the second value(Finishing time) of the first selected meeting.
3. Iterate from the second pair to last pair of the array and if the value of the first element(Starting time of meeting) of the current pair is greater then previously selected pair finish time (time_limit) then select the current pair and update the result vector (push selected meeting number into vector) and variable time_limit.
4. Print the Order of meeting from vector. → O(N log N)

Reference

- https://www.geeksforgeeks.org