**1. Problem Discussion :**

Here you are given an array(arr) of integers. You have to sort the given array in increasing order using selection sort. Here we are using word selection. It means you have to select some integers in an array. For example: Sample Input: (5 7 -2 4 1 3)

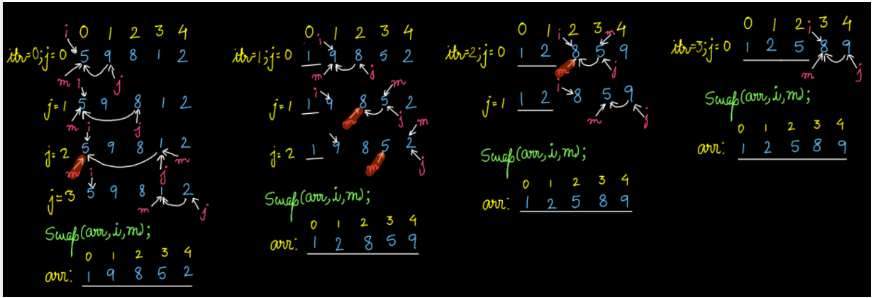
It's quite obvious that size of array is 5 and output for bubble sort on inputting any array will be the sorted array, i.e. (-2 1 3 4 7) But in order to make sure that you use the Selection Sort Algorithm in this problem, our IDE will accept the output of the following format. Because we want to know that all steps use in this algorithm which differ from other normal sorting algorithm.

● Sample Output: Comparing -2 and 7 Comparing 4 and -2 Comparing 1 and -2 Comparing 3 and -2 Swapping 7 and -2 Comparing 4 and 7 Comparing 1 and 4 Comparing 3 and 1 Swapping 7 and 1 Comparing 7 and 4 Comparing 3 and 4 Swapping 4 and 3 Comparing 4 and 7 Swapping 7 and 4 -2 1 3 4 7(in different lines)

So an advice for you is that you use the same function for swapping and comparing which is already on the IDE of this problem . for swapping we are using other function which makes function modular

**2. Aprroach-**

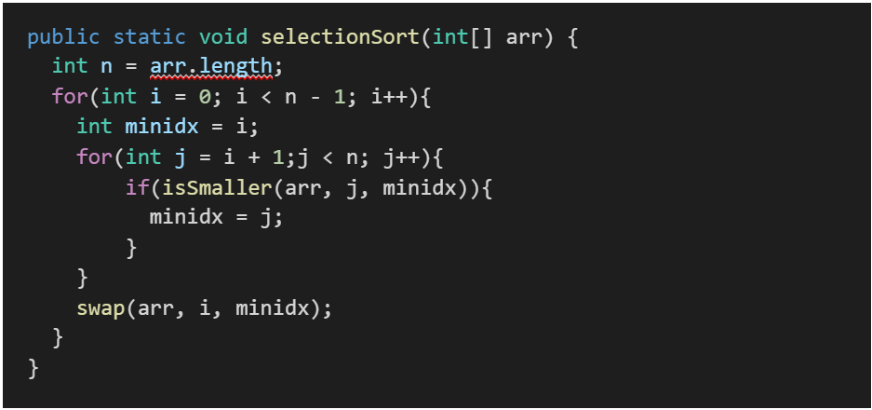
Let's say that elements of our input array are: (5, 9, 8, 2 and 1). As the length of the array is 5 that means there will be a total of four iterations. What we basically do is, find the minimum value in the array and swap it with the first element. And at the end of each iteration, the smallest value will get placed at the correct position of the array. Let's take a look at that.

****

n the first iteration, our i remains fixed at 0 index that is element 5, min is also fixed at 0 index. But j is initialized with i+1 i.e 1, then we compare 9 with 5, but 9 is not smaller than 5 therefore m will not update. Then j is incremented to 2 and 8 is compared with 5 but 8 is not smaller than 5 therefore m will not update. Then j is incremented to 3 and now 1 is compared with 5, since 1 is smaller than 5 therefore m gets updated to index 3.

Then j is again incremented to 4 and 2 is compared with 1, since 2 is greater than 1 therefore m will remain the same. After getting out of the for loop of j, m will be pointing towards the index of minimum value, i.e. 3 in this case. And then we swap value at index i and value at index m.

Since the smallest element is placed at the right position, that is at the first index. So, next time, 'for' loop of j will run for one iteration less. Let's try to code this.

****

First loop of itr will run only for n-1 times because for n-1 iterations, n-1 smallest elements of array will be placed at right positions, leaving only one option for left value. Second loop of j, will only run for the last n-1 elements of the array, comparing the value of mth index element with the jth index element. While comparing values inside the 'for' loop, if the value of (j+1)th index element is smaller than the jth index element then we simply swap the elements.

And yes, we are done! You can also watch part of the video for better understanding.

**3. Code-**

ConsoleJava

public class Main {

public static void selectionSort(int[] arr) {

int n = arr.length;

for (int i = 0; i < n - 1; i++) {

int minidx = i;

for (int j = i + 1; j < n; j++) {

if (isSmaller(arr, j, minidx)) {

minidx = j;

}

}

swap(arr, i, minidx);

}

}

// used for swapping ith and jth elements of array

public static void swap(int[] arr, int i, int j) {

System.out.println("Swapping " + arr[i] + " and " + arr[j]);

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

// return true if ith element is smaller than jth element

public static boolean isSmaller(int[] arr, int i, int j) {

System.out.println("Comparing " + arr[i] + " and " + arr[j]);

if (arr[i] < arr[j]) {

return true;

} else {

return false;

}

}

public static void print(int[] arr) {

for (int i = 0; i < arr.length; i++) {

System.out.println(arr[i]);

}

}

public static void main(String[] args) throws Exception {

Scanner scn = new Scanner(System.in);

int n = scn.nextInt();

int[] arr = new int[n];

for (int i = 0; i < n; i++) {

arr[i] = scn.nextInt();

}

selectionSort(arr);

print(arr);

}

}

**4. Analysis**

Time Complexity : Worst Case Time Complexity: O(n\*n). Worst case is when the array is in reverse order. Best Case Time Complexity: O(n). Best case is when the input array is already sorted. Average Case Time Complexity: O(n\*n) The inner loop does O(n) work on each iteration, and the outer loop runs for O(n) iterations, so the total work is O(n 2). SPACE COMPLEXITY : Auxiliary Space: O(1), as no extra space is used. We hope that this article was helpful. However, if you still face any difficulty then you are advised to watch our complete video lecture on Selection Sort