ID-191-15-12944
Tobayet Hossain Nilay

```
Package com.company;
class BubbleSort
{
  void bubbleSort(int a[])
  { int n = a.length;
    for(int i = 0; i< n-1; i++)
      if (a[j] > a[j+1])
      { // swap temp and arr[i]
        int temp = a[j];
        a[j] = a[j+1];
        a[j+1] = temp;
      }
  }
  void PrintArray(int x[])
  {
```

```
int n = x.length;
for(int i=0; i<n; ++i)
  System.out.println();
}
Public static void main (String
args[])
{
  Bubblesort ob = new
Bubblesort();
int array[] = {12, 20, 100, 90, 500, 5, 24};
ob.bubblesort(array);
System.out.println("SORTED
ARRAYS:");
  ob.PrintArray(array);
}
}
```

Robayet Hossain Niloy
191-15-12944

Algorithm - 2020
0-14

Implementing Bubble sort Algorithm

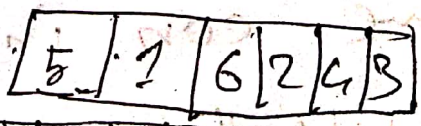
① Sorting with the first element (index=0), compare the current element with the next element of the array.

② If the current element is greater than the next element of the array swap them

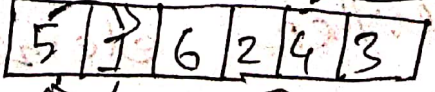
③ If the current element is less than the next element. Repeat step 1

Let's consider array with values {5, 1, 6, 2, 4, 3}

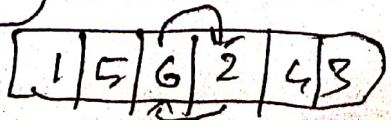
5 > 1, so interchange

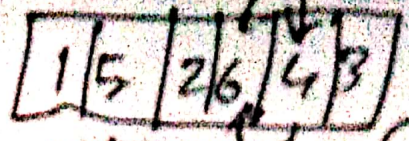


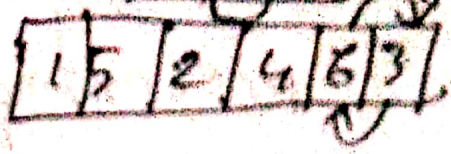
5 > 6, No swaping

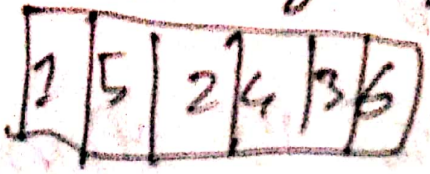


5 > 2, so, interchange



6>4. so interchange 

6>3; so interchange 



so as we can see representation above, after the first iteration, 6 is placed at the last index, which is the correct position of it similarly after the second interation, 5 will be at the second last index : and so on

Optimized the Bubble sort Algorithm

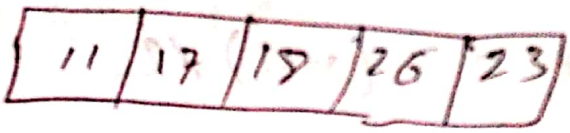
we can indicate flag & monitor whether elements are getting

swapped inside the inner for loop.

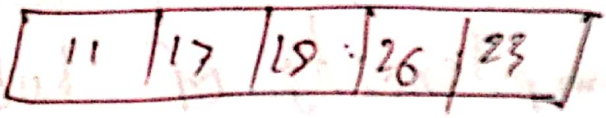
Hence, in the inner for loop, we check whether swaping of elements is taking place not every time.

Let's consider any array with values

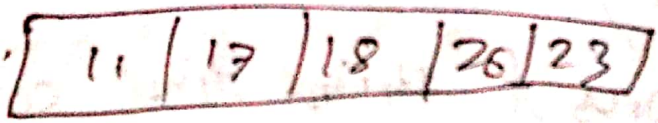
{11, 17, 18, 16, 23}

11 > 17 : 

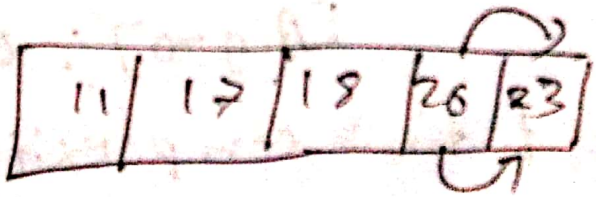
flag = 0
(No swaping)

17 > 18 

flag = 0
(No swaping)

18 > 26 : 

flag = 0
(No)

26 < 23
(swaptten) 

flag = 1

complexity:

In bubble sort, $n-1$ comparisons will be done in the 1st pass, $n-2$ in 2nd pass, $n-3$ in 3rd pass and so on. So the total number

output: $(n-1) + (n-2) + (n-3) + \ldots$ first

$$\text{sum} = n(n-1)/2$$

i.e $O(n^2)$

Time complexity of bubble sort is $O(n^2)$

- worst time complexity [Big-O]: $O(n^2)$
- Best time complexity [Big-omega] $\Omega(n)$
- Space complexity: $O(1)$

```
class GFG
{
    public static int search(int arr[]
    {
        int n = arr.length;
        for(int i = 0; i < n; i++)
        {
            if(arr[i] == x)
                return i;
        }
        return -1;
    }
    public static void main(String
    {
        int arr[] = {2, 3, 4, 10, 40};
        int x = 10;
        int result = search(arr, x);
        if(result == -1)
        System.out.Print(" Element
```

```
else
    System.out.println("Element
}
}
```

191-15-12944

Linear Search

Worst case:

In the linear search, the worst case happens when the element to be searched (x in the above code) is not present in the array. When x is not present the search() function compares it with all the elements of a[] one by one.

So the worst case time complexity of linear search would be O(n)

Average case:

For the linear search problem, that

the all cases are unfortunatly disturbed. so we sum all the cases and ~~de-disturbed~~ devided the sum by $(n+1)$

Avarage case time $= \frac{\sum_{i=1}^{n+1} O(i)}{(n+1)}$

$= \frac{O((n+1)*(n+2)/2)}{(n+1)}$

$= O(n)$

Best case: Merge sort does $O(n \log n)$ operations in all cases.

191-15-12944

```
Pobic class selectionSoint Example {
  Poblic static void
selection soint (int [] onrr) {
    for (int i=0 ; i<onrr.length -1; i++)
  {
    int index = i;
  for (int j = i+1 ; j <onrr.length ; j++){
  if (onrr[j] <onrr[index]) {
    index =j; // searching for lowest index
    }
}
in smallen Number = onrr[index];
    onrr [index ] = onn [i];
    onrr [i] = smallen Number;
  }
}
  Public static void main (string a[]){
```

```
int [] arr] = {9, 14, 3, 2, 43, 11, 58, 22};
system.out.Println ("Before selection
                        sort");
for (int i : arr]){
    system.out.println();
selection sort (arr] ); // sorting array
using selection sort
system.out.Println ("After selection
                        sort");
    for (inA i: arr]){
    system.out.Print (i + " ");
    }
}
```

```
* include <stdio.h>
// Function to Print an array
void PrintArray(int array[], int size){
  for (int i = 0; i < size; i++){
  Printf("%d", array[i]);
}
  Printf("\n");
}
void insertion sort (int array [], int size)
{
  for (int step = 1; step < size; step++)
{
int key = array[step];
int j = step - 1;
```

// Compare key with each element on the left of it until an element smaller than it is found.

// For descending order, change key < array[j] to key > array[j].

```
while (key < array[j] && j >= 0) {
  arr[j+1] = array[j]; --j;
}
array[j+1] = key;
}
}
// Driver code
int main () {
int data = {9, 5, 1, 4, 3};
int size = sizeof (data) /
```

```
sizeof(data[0]);
insertionSort(data, size);
printf("sorted array in ascending
        order:\n");
PrintArray(data, size);
}
```