**Experiment no:1 Date:3/02/19**

**AIM:**  Experiments for time complexity analysis

**Insertion Sort and Selection Sort**

**Theory:**

**Selection Sort -**

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

1) The subarray which is already sorted.  
2) Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

Alg.: SELECTION-SORT*(A)*

|  |  |
| --- | --- |
| Cost | times |
| C1 | 1 |
| C2 | n |
| C3 | n-1 |
| C4 |  |
| C5 |  |
| C6 |  |
| C7 | n-1 |

n ← length[A]

**for** j ← 1 **to** n - 1

**do** smallest ← j

**for** i ← j + 1 **to** n

**do if** A[i] < A[smallest]

**then** smallest ← i

exchange A[j] ↔ A[smallest]

Time Complexity –



**Insertion Sort –**

Insertion sort [iterates](https://en.wikipedia.org/wiki/Iteration), consuming one input element each repetition, and growing a sorted output list. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

Sorting is typically done in-place, by iterating up the array, growing the sorted list behind it. At each array-position, it checks the value there against the largest value in the sorted list (which happens to be next to it, in the previous array-position checked). If larger, it leaves the element in place and moves to the next. If smaller, it finds the correct position within the sorted list, shifts all the larger values up to make a space, and inserts into that correct position.

|  |  |
| --- | --- |
| Cost | Times |
| C1 | n |
| C2 | n-1 |
| 0 | n-1 |
| C4 | n-1 |
| C5 |  |
| C6 |  |
| C7 |  |
| C8 | n-1 |

Alg.: INSERTION-SORT*(A)*

for j ← 2 to n

do key ← A[ j ]

//Insert A[ j ] into the

// sorted sequence A[1 . . j -1]

i ← j - 1

while i > 0 and A[i] > key

do A[i + 1] ← A[i]

i ← i – 1

A[i + 1] ← key

Time Complexity –



**Program Code:**

//**Insertion Sort**

#include <stdio.h>

#include <math.h>

//function to print array.

void printArray(int arr[], int n) {

int i;

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

printf("%d ", arr[i]);

printf("\n");

}

/\* Function to sort an array using insertion sort\*/

void insertionSort(int arr[], int n) {

int i, key, j, swap = 0,comp=0;

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

comp++;

key = arr[i];

j = i - 1;

/\* Move elements of arr[0..i-1], that are greater than key, to one position ahead of their current position \*/

while (j >= 0 && arr[j] > key) {

comp++;

arr[j + 1] = arr[j];

swap++;

j = j - 1;

}

comp++;

arr[j + 1] = key;

printf("Iteration %d \n", i);

printArray(arr, n);

}

printf("\n\n%d Comparisons and %d Swap performed \n\n",comp, swap);

}

int main() {

int n = 7;

int best[7] = {15, 25, 35, 45, 55, 65, 75}; //Best Case

int worst[7] = {75, 65, 55, 45, 35, 25, 15}; //Worst case

int randomcase[7] = {25, 75, 45, 65, 55, 35, 15}; //average

printf("For Best case:Input Array \n");

printArray(best, n);

printf("Insertion sort \n");

insertionSort(best, n);

printf("Sorted array: \n");

printArray(best, n);

printf("\n\n");

printf("For Worst case:Input Array \n");

printArray(worst, n);

printf("Insertion sort \n");

insertionSort(worst, n);

printf("Sorted array: \n");

printArray(worst, n);

printf("\n\n");

printf("Average case:Input Array \n");

printArray(randomcase, n);

printf("Insertion sort \n");

insertionSort(randomcase, n);

printf("Sorted array: \n");

printArray(randomcase, n);

printf("\n\n");

return 0;

}

**Output:-**

For Best case:Input Array

15 25 35 45 55 65 75

Insertion sort

Iteration 1

15 25 35 45 55 65 75

Iteration 2

15 25 35 45 55 65 75

Iteration 3

15 25 35 45 55 65 75

Iteration 4

15 25 35 45 55 65 75

Iteration 5

15 25 35 45 55 65 75

Iteration 6

15 25 35 45 55 65 75

12 Comparisons and 0 Swap performed

Sorted array:

15 25 35 45 55 65 75

For Worst case:Input Array

75 65 55 45 35 25 15

Insertion sort

Iteration 1

65 75 55 45 35 25 15

Iteration 2

55 65 75 45 35 25 15

Iteration 3

45 55 65 75 35 25 15

Iteration 4

35 45 55 65 75 25 15

Iteration 5

25 35 45 55 65 75 15

Iteration 6

15 25 35 45 55 65 75

33 Comparisons and 21 Swap performed

Sorted array:

15 25 35 45 55 65 75

Average case:Input Array

25 75 45 65 55 35 15

Insertion sort

Iteration 1

25 75 45 65 55 35 15

Iteration 2

25 45 75 65 55 35 15

Iteration 3

25 45 65 75 55 35 15

Iteration 4

25 45 55 65 75 35 15

Iteration 5

25 35 45 55 65 75 15

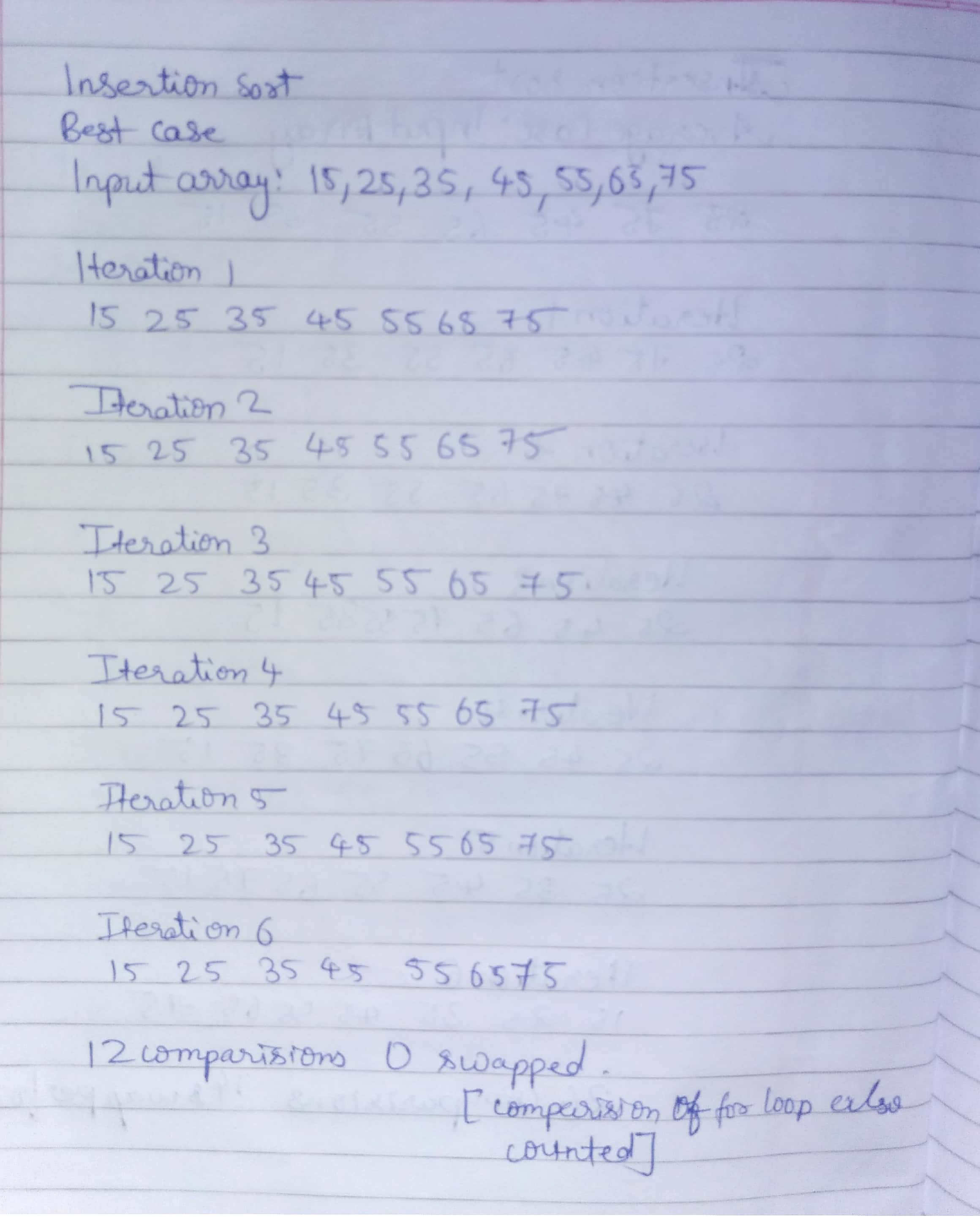
Iteration 6

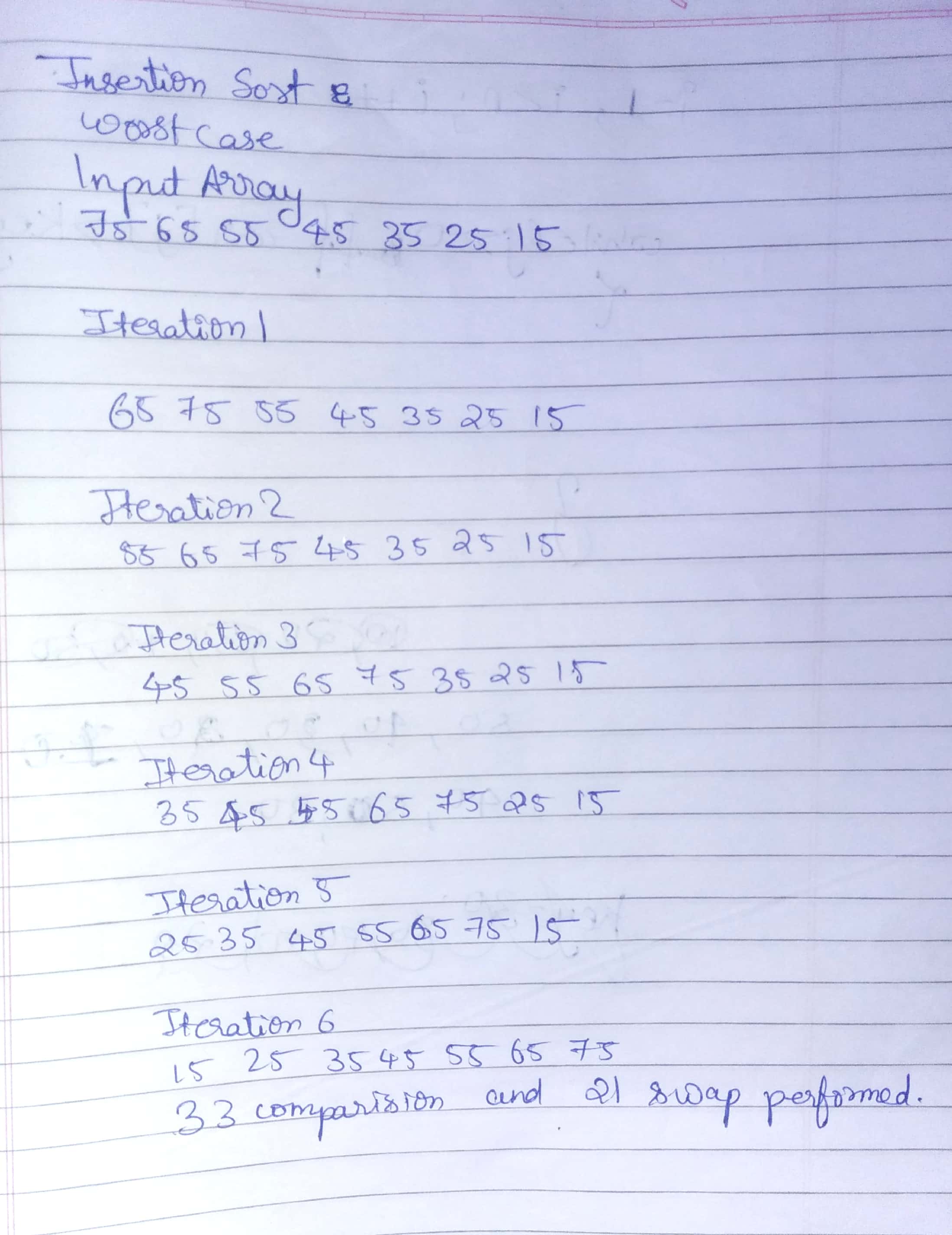
15 25 35 45 55 65 75

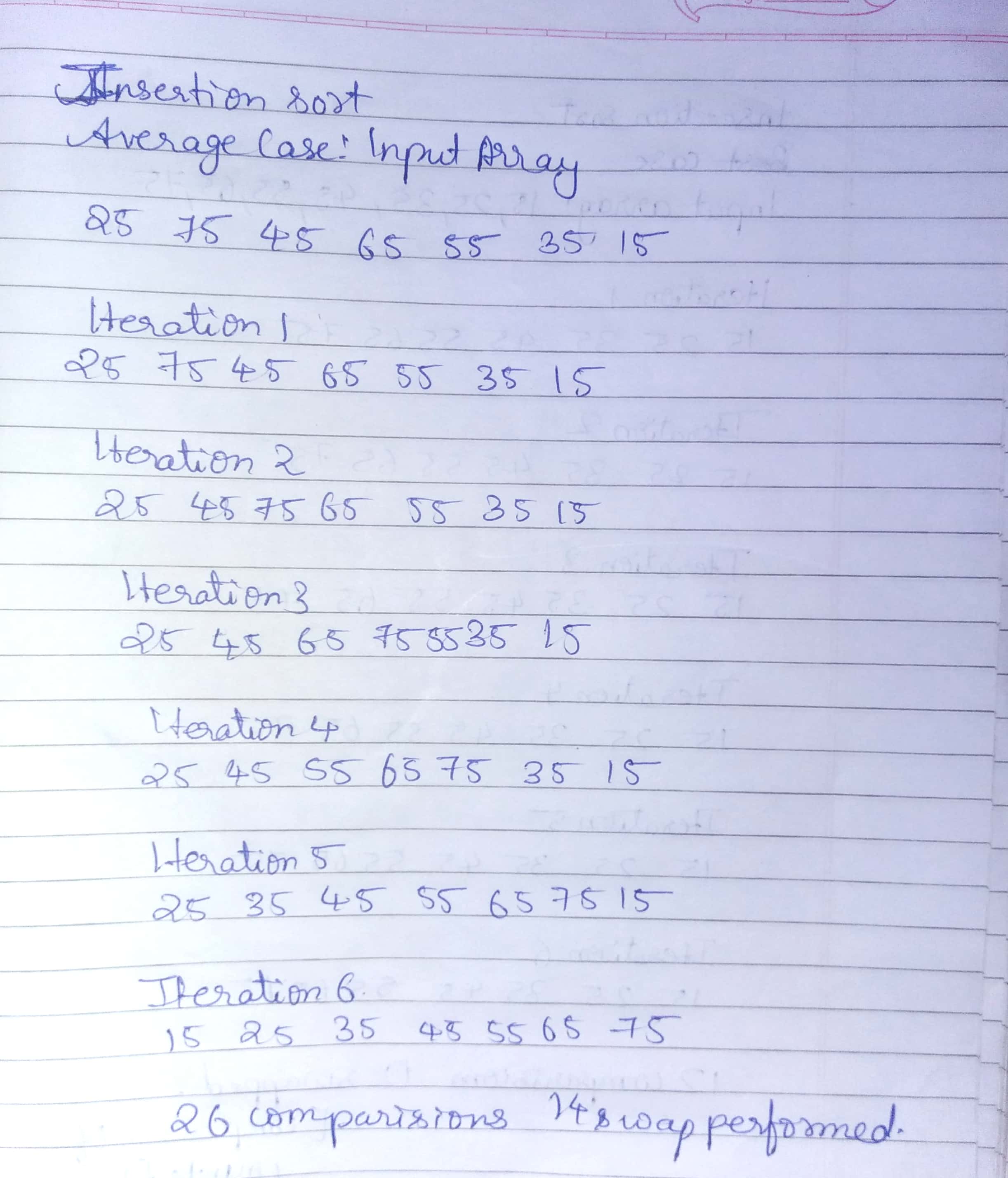
26 Comparisons and 14 Swap performed

Sorted array:

15 25 35 45 55 65 75







**//Selection Sort:-**

#include <stdio.h>

void printArray(int arr[], int size)

{

int i;

for (i=0; i < size; i++)

printf("%d ", arr[i]);

printf("\n");

}

void selectionSort(int arr[], int n)

{

int comp=0,swap=0;

int i, j, min\_idx,temp;

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

{

// Find the minimum element in unsorted array

min\_idx = i;

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

{

if (arr[j] < arr[min\_idx])

min\_idx = j;

comp+=2;

}

// Swap the found minimum element with the first element

temp=arr[i];

arr[i]=arr[min\_idx];

arr[min\_idx]=temp;

swap++;

}

printf("\n%d comparisons and %d swap performed",comp,swap);

}

int main()

{

int n=7;

int arr[7]={25,75,45,65,55,35,15};

printf("Input array \n");

printArray(arr, n);

selectionSort(arr, n);

printf("Sorted array: \n");

printArray(arr, n);

int arr2[7]={15,25,35,45,55,65,75};

printf("Input array \n");

printArray(arr2, n);

selectionSort(arr2, n);

printf("Sorted array: \n");

printArray(arr2, n);

return 0;

}

**Output:-**

Input array

25 75 45 65 55 35 15

42 comparisons and 6 swap performedSorted array:

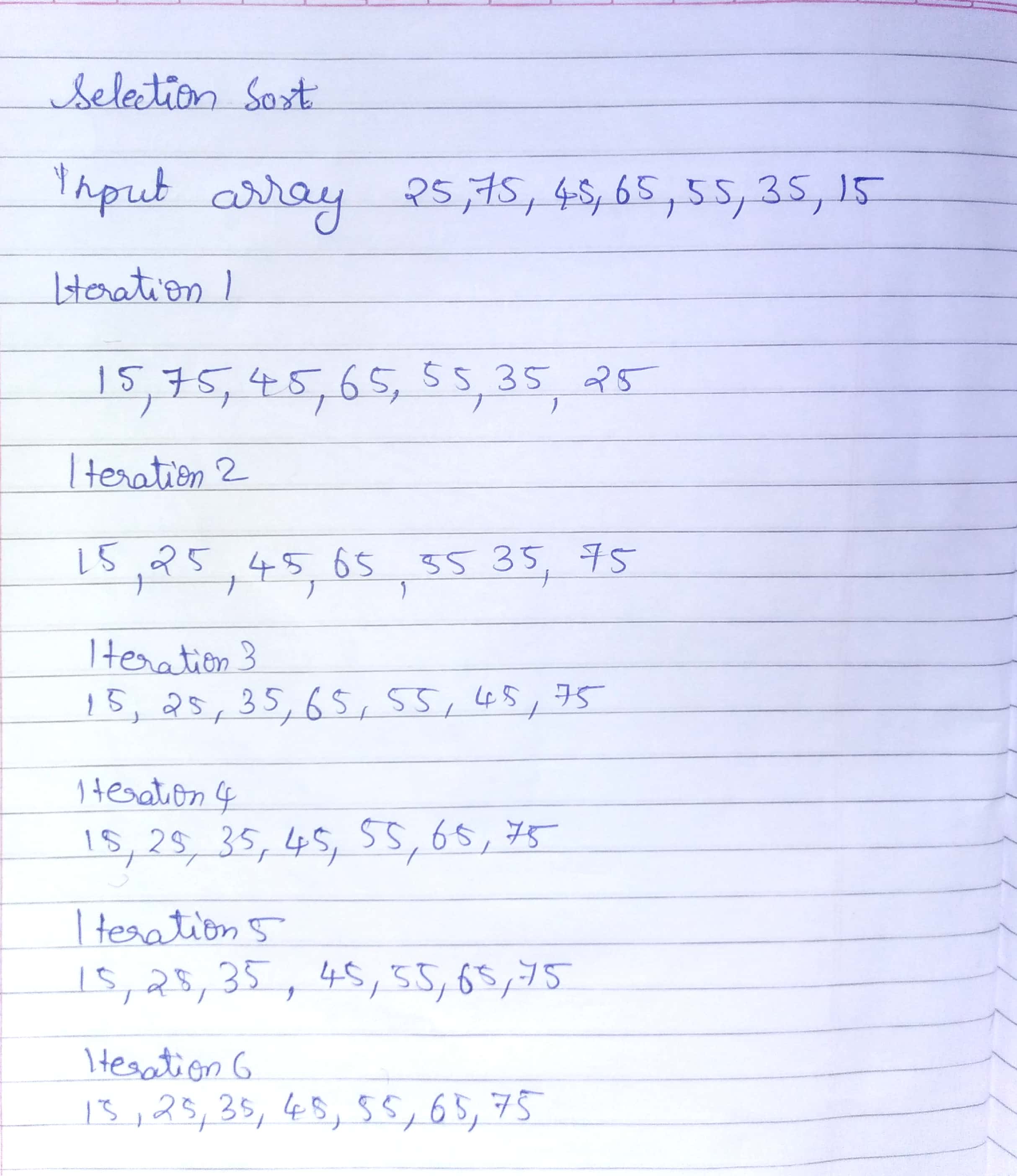
15 25 35 45 55 65 75

Input array

15 25 35 45 55 65 75

42 comparisons and 6 swap performedSorted array:

15 25 35 45 55 65 75



**Conclusion: -**

There is no best case or worst case in Selection sort. Time complexity remains same in each case.

In Insertion sort there is best and worst case which determines time taken for computing sorting the input array.