**NATIONAL INSTITUTE OF TECHNOLOGY**

**UTTARAKHAND**

****

**PROJECT ON**

**DOUBLE-SIDED INSERTION SORT**

**Submitted to: - Submitted by: -**

Dr. Maroti DeshmukhAjay Malik-BT16CSE059

Vivek Raj-BT16CSE039

**Acknowledgement**

First of all we would like to thank our parents for their financial support to complete this project. Secondly we would like to thank Dr. Maroti Deshmukh for his expert advice and guidance throughout this difficult but interesting project. We would also like to thanks our friends and colleagues for their wonderful collaborations and always willing to help us. At last but not the least we would like to thank the internet and google for getting us online support from different research papers and different sources.

**Name and Aim of our Project:**

The project name is ‘Double-Sided Insertion Sort’ and its aim is to have the deep analysis of the algorithms and program design. By the end of this project we will be able to understand how to design an algorithm and modification of algorithms to improve its efficiency .This project is based on insertion sort which will be further modified to double sided insertion sort which will certainly improve its space and time complexity for good.

Introduction to the Topic

To insert a single item in the earlier insertion sort, many write operations were necessary to perform. A linear array has two sides and both of them can be considered to insert an element in it. It is natural that there will be unequal number of required shifts to insert an item along the left or the right side. Therefore, it would be cost effective in terms of memory writes to insert an item along the side which demands less number of shifts. As data can be shifted from left or right, there must be enough space in the left to hold the moved data. This needs equal number of cells of the original array in both the left and the right sides. A momentary array with a double length of the original array can come to aid in this situation. The initial item of the unique array should be copied in the middle of the temporary array. Succeeding items would be added either to the left or to the right or to be inserted along the left side or the right side of the temporary array according to the need.

Algorithm of the Double-Sided Insertion Sort

a = given list of elements

b = temporary list with double the size of a

len = length of given list

1. Set left\_position = Len

2. Set right\_position = Len

3. Setb [left\_position] = a[0]

enhancements In

Sorting algorithms:

4. Repeat steps 5 to 9 for i = 1 to len-1

5. If (a[i]>=b[right\_position])

Set right\_position = right\_position+1

Set b[right\_position]=a[i]

Go to step 4

6. 1f(a[i]<=b[left\_position])

Set left\_position = left\_position-1

Set b[left\_position] = a[i];

Go to step 4

7. Set location = right\_position

8. Repeat while (a[i] < b[ location ])

Set location = location -1

9. If(right\_position - location < location - left\_position)

Set j=right\_position+1

while (j-l> location +l)

Set b[j]=b[j-l]

Set j=j-1

Set right\_position=right\_position + l

Set b[ location +l] = a[i]

else

Set j=left\_position-1

while (j< location )

b[j]=b[j+l]

Set j=j+1

Set left\_position = left\_position-1

Set b[ location ] = a[i]

10. for i = 0 to n-l

Set a[i]=b[left\_position]

Set left\_position = left\_position+1

Code of the Algorithm

def modify\_insertionsort(a , n , t):

b = [None] \* 2\*n

left\_pointer = n

right\_pointer = n

b[n] = a[0]

for i in range(1,n):

if( a[i] >= b[right\_pointer] ):

right\_pointer += 1

b[right\_pointer] = a[i]

continue

if( a[i] <= b[left\_pointer] ):

left\_pointer -= 1

b[left\_pointer] = a[i]

continue

location = right\_pointer

while( a[i] < b[location] ):

location -= 1

if( right\_pointer - location < location - left\_pointer ):

j = right\_pointer +1

while(j > location + 1 ):

b[j] = b[j-1]

j -= 1

right\_pointer += 1

b[location + 1 ] = a[i]

else:

j = left\_pointer - 1

while( j < location ):

b[j] = b[j+1]

j += 1

left\_pointer -= 1

b[location] = a[i]

if(t == 1):

for x in range(n):

a[x] = b[left\_pointer]

left\_pointer += 1

else:

for x in range(n):

a[x] = b[right\_pointer]

right\_pointer -= 1

return(a)

print("enter array :-")

arr = [int(y) for y in input().split()]

print('enter 1 for increasing order or 0 for decreasing order')

t = int(input())

print ("Sorted array is:")

arr = modify\_insertionsort(arr,len(arr) , t)

for i in arr:

print (i)

Example of the Double sided insertion sort:

A = [89, -9, 12, 10, 99, 59, -15, 100] - - given array

N = size of array A = 8

B = [ \_ , \_, \_ , \_ , \_ , \_ , \_ , \_ , \_ , \_ , \_ , \_, \_ , \_ , \_, \_ ] array of size 16

Left\_pointer = 8 Right \_pointer = 8

B[n] = A[0]

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Pass 1: \_ \_ \_ \_ \_ \_ \_ -9 89 \_ \_ \_ \_ \_ \_ \_ A[i] = -9

* -9<89 left pointer = left\_pointer -1 B[left\_pointer] = A[1] ,B[6] = -9
* Left\_pointer -= 1

Pass 2: \_ \_ \_ \_ \_ \_ -9 12 89 \_ \_ \_ \_ \_ \_ \_ A[i] = 12

* 12 < B[ right\_pointer ] ,location = 8 ,right position of 12 = 7, so 12 is replaced with -9
* left\_pointer -= 1

Pass 3: \_ \_ \_ \_ \_ -9 10 12 89 \_ \_ \_ \_ \_ \_ \_ A[i] = 10

* 10 > B[6] , location =6, so 10 is at 6th and -9 is shifted to 5th position
* left\_pointer -= 1

Pass 4: \_ \_ \_ \_ \_ -9 10 12 89 99 \_ \_ \_ \_ \_ \_ A[i] = 99

* 99 > rightmost element
* right\_pointer += 1, b[right\_pointer] = A[i]

Pass 5: \_ \_ \_ \_ \_ -9 10 12 59 89 99 \_ \_ \_ \_ \_ A[i] = 59

* Correct location of A[i] is 8 so element of B will shift from right side rather than the left side
* Element 89 , 99 are shifted to right ,B[location] = A[i] , right\_pointer += 1

Pass 6: \_ \_ \_ \_ -12 -9 10 12 59 89 99 \_ \_ \_ \_ \_ A[i] = -12

* A[i] < B[left\_pointer ] , so it will directly add at( left\_pointer -1)
* Left\_pointer -= 1 , B[left\_pointer ] = A[i]

Pass 7: \_ \_ \_ \_ -12 -9 10 12 59 89 99 100 \_ \_ \_ \_ A[i] = 100

* A[i] > B[ right\_pointer ] , so it will directly add at (right\_pointer + 1)
* Right\_pointer += 1 , B[right\_pointer] = A[i]

Sorted list is [-12 , -9 , 10 , 12 , 59 , 89 , 99 , 100]

Example: (comparison between insertion sort and double-sided insertion sort)

Average case (Insertion Sort)

n-1 passes are required to sort the list

0 1 2 3 4 5 6 7 - indices of list

PASS-1 3 7 4 9 5 2 6 1 write operation = 0

3\* 7 4 9 5 2 6 1

PASS-2 3 7\* 4 9 5 2 6 1

3 4\* 7 9 5 2 6 1 write operation = 1

PASS-3 3 4 7 9\* 5 2 6 1 write operation = 0

PASS-4 3 4 7 5\* 9 2 6 1

3 4 5\* 7 9 2 6 1 write operation = 2

PASS-5 3 4 5 7 2\* 9 6 1

3 4 5 2\* 7 9 6 1

3 4 2\* 5 7 9 6 1

3 2\* 4 5 7 9 6 1

2\* 3 4 5 7 9 6 1 write operation = 4

PASS-6 2 3 4 5 7 9 6\* 1

2 3 4 5 7 6\* 9 1

2 3 4 5 6\* 7 9 1 write operation = 2

PASS-7 2 3 4 5 6 7 1\* 9

2 3 4 5 6 1\* 7 9

2 3 4 5 1\* 6 7 9

2 3 4 1\* 5 6 7 9

2 3 1\* 4 5 6 7 9

2 1\* 3 4 5 6 7 9

1\* 2 3 4 5 6 7 9 write operation = 7

**total write operation = 16**

Average Case (Double Sided Insertion Sort)

n-1 passes are required to sort the list

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 - indices of list (2 \* size of original list)

\_ \_ \_ \_ \_ \_ \_ \_ \*3\* \_ \_ \_ \_ \_ \_ \_ write operation = 0

PASS-1 \_ \_ \_ \_ \_ \_ \_ \_ \*3 7\* \_ \_ \_ \_ \_ \_

PASS-2 \_ \_ \_ \_ \_ \_ \_ \_ \*3 7\* \_ \_ \_ \_ \_ \_

\_ \_ \_ \_ \_ \_ \_ \_ \*3 4 7\* \_ \_ \_ \_ \_ write operation = 1

PASS-3 \_ \_ \_ \_ \_ \_ \_ \_ \*3 4 7 9\* \_ \_ \_ \_ write operation = 1

PASS-4 \_ \_ \_ \_ \_ \_ \_ \_ \*3 4 7 5\* 9 \_ \_ \_

\_ \_ \_ \_ \_ \_ \_ \_ \*3 4 5\* 7 9 \_ \_ \_ write operation = 2

PASS-5 \_ \_ \_ \_ \_ \_ \_ \*2 3 4 5\* 7 9 \_ \_ \_ write operation = 0

PASS-6 \_ \_ \_ \_ \_ \_ \_ \*2 3 4 5 7 6\* 9 \_ \_

\_ \_ \_ \_ \_ \_ \_ \*2 3 4 5 6\* 7 9 \_ \_ write operation =2

PASS-7 \_ \_ \_ \_ \_ \_ \*1 2 3 4 5 6 7 9\* \_ \_ write operation = 0

====================================

**total write operation = 6**

Analysis

The number of write operation in insertion sort is more than twice of the double-sided insertion sort which is 16 in insertion sort and 6 in double-sided insertion sort which is 10 operations less than that.

**Conclusion**

In original insertion sort time complexity are O(n^2), O(n), O(n^2) in average ,best ,worst case respectively , With the help of double-sided insertion sort we can reduce the time complexity of worst case to O(n) . This algorithm will give time complexity O(n) if the elements are arranged such that next element either larger or smaller than the previous all , because in that case element will add at before left most element (in case of smaller) or after right most element (in case of larger) no further requirement of finding the correct location for that element .