Shubham Sanjay Jain

CWID: 104-56-815

Stevens Institute of Technology

Master of Science in Computer Science

Submitted To: Prof. Iraklis Tsekourakis

CS 590 Algorithms

Report: Assignment I

Question 1: To develop an insertion sort for integer vector that precomputes the length of each vector before the sorting. You can test the correctness of your sorting algorithm using the provided check_sorted.

Solution:

Developed a function called insertion_sort_im which would precompute length of integer vectors using for loop. Whereas in previously given naïve algorithm length of vector array was computed every time it entered loop before swapping with previous integer arrays which ultimately led to increase in complexity to $O(n^3)$. Therefore, developing the efficient led to increase in space of m but which can be compensated because of huge decrease in time complexity by 1 degree to $O(n^2)$. The algorithm works the same as naïve but lengthsample variable is array of m which is used to store the values of length of each array in vector.

Question 2: Implement a merge sort for an array of integer vectors. As for the improved insertion sort algorithm, you should precompute the length of the vectors before the sorting and the sorting is done according to the vector length. Test the correctness of your merge sort implementation using the provided check_sorted function.

Solution:

Merge_sort function is called from main to implement the merge sort algorithm. In merge_sort, the function precomputes the vector length in the same previous variable lengthsample. The lengthsample being array is passed to function called merge_sort1 where the recursive implementation of merge sort takes place using divide and conquer and combine way. The vector is divided in this function by computing middle value of the array. The middle value is half or first index and last index. And the merge function is used to merge the divided array by adding values in the ascending order. The time complexity for this algorithm to sort the array of integer vector is $O(n\log_2 n)$.

Question 3: Measure the runtime performance of insertion sort and merge sort for random, sorted, and inverse sorted inputs of size m = 10000, 25000, 50000,100000,250000,500000,1000000, 2500000 and vector dimension n = 10, 25,50. You can use the provided function create_random_ivector, create_sorted _ivector, create reverse_sorted_ivector.
Repeat each test a number of times and compute the average running time for each combination of algorithm, input, size m, and vector dimension n. Report and comment on your results.

Solution: I had run results for random inputs with input size from 10,000 to 100000. As for 100000 or more input size the time required for naïve insertion was more than 6 minutes. The results of each test cases is saved in google drive for reference in below given link https://drive.google.com/open?id=1gaOqoSm8r2dZzs2kohF0XTb3KwovHohQ

OutCome and Conclusion:

Random Input values

Naïve Insertion Sort

N of Vector (No. of Array)	**M of Vector (Dimension of Array)**	**Time Required (**In MilliSec)
10000	10	1323.6
	25	3477.7
	50	7677.4
25000	10	8667.9
	25	21286.7
	50	51027.6
50000	10	35442.5
	25	95271.2
	50	209828.3
100000	10	158821.9
	25	466335

Improved Insertion Sort

N of Vector (No. of Array)	**M of Vector (Dimension of Array)**	**Time Required (**In MilliSec)
10000	10	75.5
	25	77.1
	50	77.8
25000	10	471.9
	25	456.8
	50	484.9
50000	10	1894.1
	25	1827.3
	50	1852
100000	10	7360.4
	25	7270.2

Merge Sort

N of Vector (No. of Array)	**M of Vector (Dimension of Array)**	**Time Required (**In MilliSec)
10000	10	2.6
	25	3.2
	50	4.1
25000	10	6.7
	25	7.7
	50	10.2
50000	10	13.8
	25	16.6
	50	19.6
100000	10	29.6
	25	34.1

For Random Input values, we can see that Merge Sort is fastest of all for every dimension and height input. Merge Sort is about 200 times faster than Improved Insertion Sort and Improved Insertion Sort is about 64 times faster than Naïve Insertion Sort. The time required for Random input is not highest of all types of input for Naïve and Improved.

Sorted Inputs

Naïve Insertion Sort

N of Vector (No. of Array)	**M of Vector (Dimension of Array)**	**Time Required (**In MilliSec)
10000	10	0.7
	25	1.7
	50	3.5
25000	10	1.6
	25	3.7
	50	7.8
50000	10	2.7
	25	7.4
	50	15.9
100000	10	5.5
	25	13.5
	50	30.6
250000	10	13.7
	25	35.3
	50	74.4

Improved Insertion Sort

N of Vector (No. of Array)	**M of Vector (Dimension of Array)**	**Time Required (**In MilliSec)
10000	10	0.1
	25	0.9
	50	1.3
25000	10	0.8
	25	2.3
	50	4.0
50000	10	1.7
	25	3.7
	50	8.6
100000	10	3.3
	25	7.6
	50	16.6
250000	10	8.4
	25	18.8
	50	39.9

Merge Sort

N of Vector (No. of Array)	**M of Vector (Dimension of Array)**	**Time Required (**In MilliSec)
	10	2.2
10000	25	2.6
	50	3.7
25000	10	5.7
	25	7.3
	50	9.4
50000	10	11.9
	25	14.5
	50	19.4
100000	10	24.5
	25	28.8
	50	37.0
250000	10	65.7
	25	80.9
	50	119.8

Sorted inputs take least time for Naïve Insertion sort and Improved Insertion Sort. Whereas, merge sort in sorted input take same time more or less as reverse sort as the whether the array are sorted or not, merge sort will divide an array to least dividable value i. e 1 unit. But for small values merge sort works very badly and both naïve and improved works really very well as they don't need process the swapping, just need to run the loop.

REVERSE SORTED

Naïve Insertion Sort

N of Vector (No. of Array)	**M of Vector (Dimension of Array)**	**Time Required (**In MilliSec)
10000	10	2530.2
	25	6524.7
	50	14216.8
25000	10	16032.6
	25	41271.5
	50	93999.5
50000	10	66886.2
	25	369839.7
	50	355362

Improved Insertion Sort

N of Vector (No. of Array)	**M of Vector (Dimension of Array)**	**Time Required (**In MilliSec)
10000	10	145.8
	25	146.4
	50	144
25000	10	905.1
	25	919.5
	50	954
50000	10	3787.6
	25	3647.1
	50	3566

Merge Sort

N of Vector (No. of Array)	**M of Vector (Dimension of Array)**	**Time Required (**In MilliSec)
10000	10	2
	25	2.5
	50	3.1

25000	10 25 50	5 6.2 8.5
50000	10 25 50	11.3 12.6 18

Reverse Sort takes most time for Naïve and Improved Sort because, each value has to be swapped to the last position as they are reversely sorted. But merge runs efficiently that case scenario as it doesn’t matter which way input are provided.