

**Symbiosis Institute of Technology**

**Faculty of Engineering**

**CSE- Academic Year 2024-25**

**Data Structures – Lab Batch 2023-27**

|  |  |
| --- | --- |
| **Lab Assignment No:- 3** | |
|  | |
| **Name of Student** | Faheemuddin Sayyed |
| **PRN No.** | 23070122196 |
| **Batch** | 23-27 |
| **Class** | CSE C-1 |
| **Academic Year & Semester** | 24-25 Sem 3 |
| **Date of Performance** | 1/08/24 |
|  | |
| **Title of Assignment:** | Implement following sorting techniques find the time complexity: Merge |
| **Theory Questions:** | 1. Apply merge Sort on 9 input items and show the partial pass-wise sorting done. Analyze its Time Complexity (Best, Worst, and Average Case) & Space Complexity 2. Discuss time complexity of merge sort and quick sort in detail.   **Answer 2:**  Merge Sort:  • Time Complexity:  • Best Case: O(n log n) – Always divides the array into halves and requires merging.  • Worst Case: O(n log n) – Similar to the best case; the algorithm consistently requires log n levels of merging.  • Average Case: O(n log n) – Same as best and worst cases.  Quick Sort:  • Time Complexity:  • Best Case: O(n log n) – Occurs when the pivot divides the array into nearly equal parts.  • Worst Case: O(n^2) – Occurs when the pivot is the smallest or largest element, causing unbalanced splits (eg, when the array is already sorted).  • Average Case: O(n log n) – Generally occurs with random pivots.  Summary:  • Merge Sort has a stable O(n log n) time complexity for all cases but requires additional space.  • Quick Sort is faster on average but can degrade to O(n^2) in the worst case, although it is usually more space-efficient than Merge Sort.  **Answer 1:** |
| **Source Code/Algorithm/Flow Chart:** | #include <stdio.h>  #include <stdlib.h>  void Merge(int a[], int l, int mid, int h){  int i = l, j = mid + 1, k = l;  int b[100];  while(i <= mid && j <= h){  if(a[i] < a[j])  b[k++] = a[i++];  else  b[k++] = a[j++];  }  for(; i <= mid; i++)  b[k++] = a[i];  for(; j <= h; j++)  b[k++] = a[j];  for(i = l; i <= h; i++)  a[i] = b[i];  }  void MergeSort(int a[], int l, int h){  int mid;  if(l < h){  mid = (l + h) / 2;  MergeSort(a, l, mid);  MergeSort(a, mid + 1, h);  Merge(a, l, mid, h);  }  }  int main(){  int \*A;  int n;  printf("\nEnter no. of elements of array: ");  scanf("%d", &n);  A = (int \*)malloc(n \* sizeof(int));  printf("\nEnter elements of the array:\n");  for(int i = 0; i < n; i++)  scanf("%d", &A[i]);  MergeSort(A, 0, n - 1);  printf("\nSorted array:\n");  for(int i = 0; i < n; i++)  printf("%d ", A[i]);  printf("\n");  return 0;  } |
| **Output Screenshots** |  |
| **Practice questions** | 1. Implement Quick sort 2. o/p screenshot |
| **Conclusion** | Thus we have studied different sorting algorithms and their time complexities. |