Experiment 2 [B]

Problem Statement: Given a sequence of n numbers <a1, a2, ...,an>, find a permutation (reordering) <a1', a2', ...,an'> of the input sequence such that a1'<= a2'<=, ...,<=an', using merge sort algorithm.

Theory

- A merge sort is a sorting algorithm that starts by splitting an unordered list of items into two halves called sublists.
- Then the algorithm repeatedly splits the sublists into smaller sublists until it reaches sublists of single elements.
- These single element sublists are then merged in pairs to form sublists of two items. In the process of merging the items are ordered.
- This process is repeated until, in the final step, the two ordered halves are merged together to form a single ordered list.
- As a result, the whole list is sorted.
- The merge sort algorithm is an example of a divide and conquer approach to problem solving. It operates as follows.
- **Divide:** Divide the n-elements sequence to be sorted into two sub-sequences of n/2 elements each recursively.
- **Conquer:** Sort the two sub-sequences recursively using merge sort.
- **Combine:** Merge the two sorted sub-sequences to produce the sorted answer.
- In the case of Merge sort, the recursion “bottoms out” when the sequence to be sorted has length 1, in which case there is no work to be done since every sequence of length 1 is already in sorted order.

Dheeraj Lalwani
1902085

Algorithm

- ➔ Accept n, the number of elements
- ➔ Accept the data (numbers) into array
- ➔ function merge (array, left, mid, right)
  - ◆ p1 **<<<** 0
  - ◆ p2 **<<<** 0
  - ◆ n1 **<<<** mid - left + 1
  - ◆ n2 **<<<** right - mid
  - ◆ array1[n1] **<<<** array[left ... mid]
  - ◆ array2[n2] **<<<** array[mid+1 ... right]
  - ◆ while (p1 < n1 and p2 < n2)
    - if (array1[p1] > array2[p2])
      - array[k] **<<<** array2[p2]
      - increment p2 & k
    - else
      - array[k] **<<<** array1[p1]
      - increment p1 & k
  - ◆ while (p1 < n1)
    - array[k] **<<<** array1[p1]
    - increment p1 & k
  - ◆ while (p2 < n2)
    - array[k] **<<<** array2[p2]
    - increment p2 & k
- ➔ function mergeSort(array, left, right)
  - ◆ If (left < right)
    - mid **<<<** (left + right) / 2
    - mergeSort(array, left, mid)
    - mergeSort(array, mid + 1, right)
    - merge(array, left, mid, right)

Example

Consider an array of size 5

| 5 | 4 | 3 | 2 | 1 |

Pass 1: | 4 | 5 | 3 | 2 | 1 |
Pass 2: | 3 | 4 | 5 | 2 | 1 |
Pass 3: | 3 | 4 | 5 | 1 | 2 |
Pass 4: | 1 | 2 | 3 | 4 | 5 |

Sorted array: | 1 | 2 | 3 | 4 | 5 |

Complexity Analysis

Time Complexity

In general, the time complexity depends on the size and arrangement of values within an array. Therefore, the standard terms like “best-case” and “worst-case” complexities are a measure of time-complexities for algorithms.

Worst Case
If the 1st pass results only in one partition i.e. say only left or only right (this will happen when the pivot is either the largest or smallest in the list), then we have n comparisons in the 1st pass & then n – 1 in the 2nd pass & n – 3 in the 3rd pass & so on…. Thus we have n – 1 passes with number of comparisons in each pass being n, n -1, n – 2, n – 3, ……..
Thus the sum of comparisons is an arithmetic progression and hence is order of n2 i.e. O(n2).

Best Case
The time complexity of Merge Sort is: O(n log n)
And that is regardless of whether the input elements are presorted or not. Merge Sort is therefore no faster for sorted input elements than for randomly arranged ones.

Space Complexity

In simpler terms, space complexity refers to the amount of excess space or memory used during the running of the algorithm. The space complexity of Merge sort is O(n). This means that this algorithm takes a lot of space and may slower down operations for the last data sets

Code

```
#include <stdio.h>
#define MAX 100
int data[MAX], n, i, pass = 0;

void printArray(int array[])
{
    for (i = 0; i < n; i++)
    {
        printf("| %d |", array[i]);
    }
    printf("\n");
}

void merge(int array[], int l, int mid, int r)
{
    int n1 = mid - l + 1;
    int n2 = r - mid;
    int array1[n1];
    int array2[n2];
    int p1 = 0;
    int p2 = 0;
    int j = 0;
    int k = l;

    for (j = 0; j < n1; j++)
    {
        array1[j] = array[l + j];
    }

    for (j = 0; j < n2; j++)
    {
        array2[j] = array[mid + 1 + j];
    }

    while (p1 < n1 && p2 < n2)
```

```
    {
        if (array1[p1] > array2[p2])
        {
            array[k] = array2[p2];
            p2++;
            k++;
        }
        else
        {
            array[k] = array1[p1];
            p1++;
            k++;
        }
    }

    while (p1 < n1)
    {
        array[k] = array1[p1];
        p1++;
        k++;
    }
    while (p2 < n2)
    {
        array[k] = array2[p2];
        p2++;
        k++;
    }
}

void mergeSort(int array[], int l, int r)
{
    int mid;
    if (l < r)
    {
        mid = (l + r) / 2;
        mergeSort(array, l, mid);
        mergeSort(array, mid + 1, r);
        merge(array, l, mid, r);
        printf("Pass No. %d: ", pass);
        printArray(array);
        pass++;
    }
}

int main()
{
    printf("\n");
```

```
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Start entering %d integers: \n", n);
    for (i = 0; i < n; i++)
    {
        printf("data[%d] = ", i);
        scanf("%d", &data[i]);
    }

    printf("\n------- Array before sorting -------\n");
    printArray(data);
    printf("------- x ------- x ------- x -------\n\n");

    mergeSort(data, 0, n - 1);

    printf("\n------- Array after sorting -------\n");
    printArray(data);
    printf("------- x ------- x ------- x -------\n\n");

    return 0;
}
```

Output

```
Enter number of elements: 7
Start entering 7 integers:
data[0] = 7
data[1] = 3
data[2] = 2
data[3] = 5
data[4] = 4
data[5] = 1
data[6] = 6
```

```
------- Array before sorting -------
| 7 || 3 || 2 || 5 || 4 || 1 || 6 |
------- x ------- x ------- x -------

Pass No. 0: | 3 || 7 || 2 || 5 || 4 || 1 || 6 |

Pass No. 1: | 3 || 7 || 2 || 5 || 4 || 1 || 6 |

Pass No. 2: | 2 || 3 || 5 || 7 || 4 || 1 || 6 |

Pass No. 3: | 2 || 3 || 5 || 7 || 1 || 4 || 6 |

Pass No. 4: | 2 || 3 || 5 || 7 || 1 || 4 || 6 |

Pass No. 5: | 1 || 2 || 3 || 4 || 5 || 6 || 7 |

------- Array after sorting -------
| 1 || 2 || 3 || 4 || 5 || 6 || 7 |
------- x ------- x ------- x -------
```