

PROGRAM 1

Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

ALGORITHM:

```
void merge (int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int L[n1], R[n2];
    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j])
            arr[k] = L[i];
            i++;
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
}
```

```
while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;
}
while (j < n2) {
    arr[k] = R[j];
    j++;
    k++;
}
}

void mergeSort (int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort (arr, left, mid);
        mergeSort (arr, mid + 1, right);
        merge (arr, left, mid, right);
    }
}
```

CODE:

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <time.h>
```

```
// Function to merge two halves
```

```
void merge(int arr[], int left, int mid, int right) {
```

```
    int i, j, k;
```

```
    int n1 = mid - left + 1;
```

```
    int n2 = right - mid;
```

```
    // Temporary arrays
```

```
    int L[n1], R[n2];
```

```
    // Copy data
```

```
    for (i = 0; i < n1; i++)
```

```
        L[i] = arr[left + i];
```

```
    for (j = 0; j < n2; j++)
```

```
        R[j] = arr[mid + 1 + j];
```

```
    // Merge the temp arrays
```

```
    i = 0;
```

```
    j = 0;
```

```
    k = left;
```

```
    while (i < n1 && j < n2) {
```

```
        if (L[i] <= R[j]) {
```

```
            arr[k++] = L[i++];
```

```
        } else {
```

```
            arr[k++] = R[j++];
```

```
        }
```

```
    }
```

```

// Copy remaining elements
while (i < n1) {
    arr[k++] = L[i++];
}
while (j < n2) {
    arr[k++] = R[j++];
}
}

// Merge Sort function
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;

        mergeSort(arr, left, mid);    // Left half
        mergeSort(arr, mid + 1, right); // Right half

        merge(arr, left, mid, right); // Merge halves
    }
}

int main() {
    int N, i;
    printf("Enter the number of elements (N): ");
    scanf("%d", &N);

    int arr[N];
    printf("Enter %d integer elements:\n", N);
    for (i = 0; i < N; i++) {

```

```

        scanf("%d", &arr[i]);
    }

    clock_t start, end;
    double time_taken;

    start = clock();
    mergeSort(arr, 0, N - 1);
    end = clock();

    time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;

    printf("Sorted array:\n");
    for (i = 0; i < N; i++) {
        printf("%d ", arr[i]);
    }

    printf("\nTime taken to sort %d elements: %f seconds\n", N, time_taken);

    return 0;
}

```

OUTPUT:

```

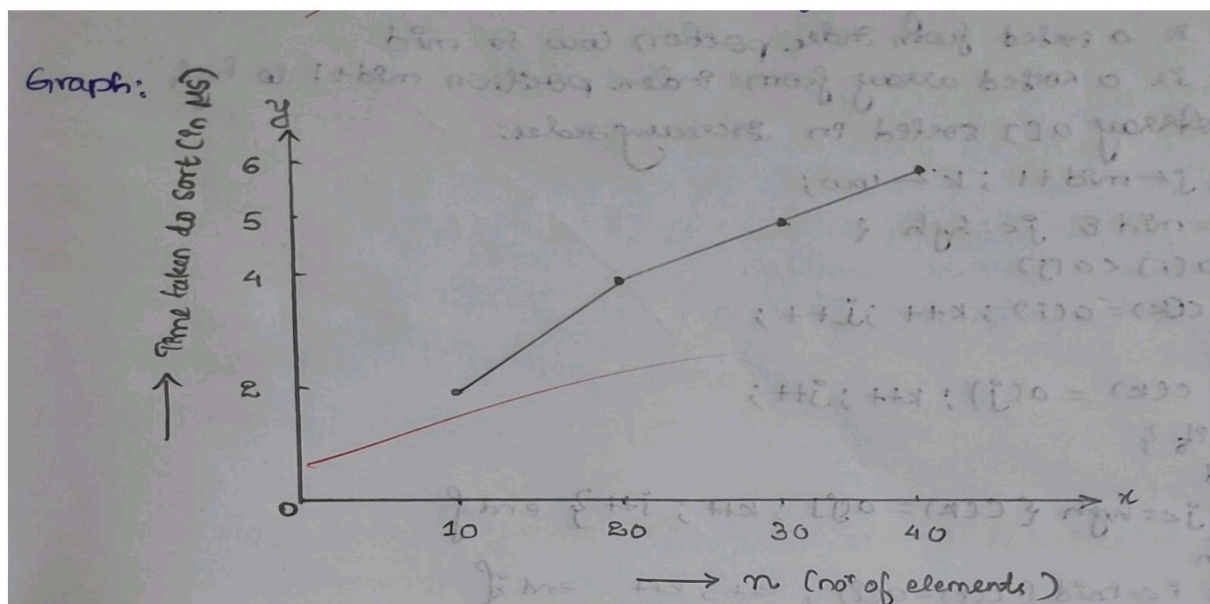
Enter the number of elements (N): 5
Enter 5 integer elements:
1 2 3 4 5
Sorted array:
1 2 3 4 5
Time taken to sort 5 elements: 0.000002 seconds

```

TRACING:

Tracing:
Input: [5, 2, 8]
call: mergeSort(arr, 0, 2)
Left half: mergeSort(arr, 0, 1)
Right half: mergeSort(arr, 2, 2)
mergeSort(arr, 0, 1)
→ mergeSort(arr, 0, 0) // 5
→ mergeSort(arr, 1, 1) // 2
→ merge(arr, 0, 0, 1) // merge 5 & 2
Result: [2, 5]
merge(arr, 0, 1, 2)
compare:
2 < 8 → arr[0] = 2
5 < 8 → arr[1] = 5
arr[2] = 8
Final result: [2, 5, 8]

GRAPH:



LEETCODE 1

COUNT RANGE SUM

ALGORITHM

Input:
* num[]: array of integers
* lower, upper: the inclusive range of valid subarray sums
Q: # of subarrays num[l..r] such that sum lies in [lower, upper]

```
countRangeSum (nums, l, u)
    n = length(nums)
    prefix = array of size n + 1
    prefix[0] = 0
    for i from 0 to n-1:
        prefix[i+1] = prefix[i] + nums[i]
    return mergeSortAndCount (prefix, 0, n+1, l, u)

mergeSortAndCount (prefix, left, right, l, u):
    if right - left <= 1:
        return 0
    mid = (left + right) / 2
    count = 0
    count += mergeSortAndCount (prefix, left, mid, lower, upper)
    count += mergeSortAndCount (prefix, mid, right, lower, upper)
```

```
j = k = t = mid
temp = empty array
s = 0
for i from left to mid-1:
    while k < right & prefix[k] - prefix[i] <= upper:
        k += 1
    while j < right & prefix[j] - prefix[i] <= lower:
        j += 1
    count += (j - k)
    while t < right & prefix[t] < prefix[i]:
        append prefix[t] to temp
        t += 1
    append prefix[i] to temp
# merge remaining elements
copy temp back to prefix [left to left + size of temp]
return count
```

CODE:

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
int countWhileMergeSort(long* sum, int left, int right, int lower, int upper) {
```

```
    if (right - left <= 1) return 0;
```

```
    int mid = (left + right) / 2;
```

```
    int count = countWhileMergeSort(sum, left, mid, lower, upper) +  
                countWhileMergeSort(sum, mid, right, lower, upper);
```

```
    int j = mid, k = mid, t = mid, r = 0;
```

```
    long* cache = (long*)malloc((right - left) * sizeof(long));
```

```
    for (int i = left; i < mid; ++i) {
```

```
        while (k < right && sum[k] - sum[i] < lower) k++;
```

```
        while (j < right && sum[j] - sum[i] <= upper) j++;
```

```
        count += j - k;
```

```
        while (t < right && sum[t] < sum[i])
```

```
            cache[r++] = sum[t++];
```

```
        cache[r++] = sum[i];
```

```
    }
```

```
    for (int i = 0; i < t - left; ++i)
```

```
        sum[left + i] = cache[i];
```

```
    free(cache);
```

```
    return count;
```

```
}
```

```

int countRangeSum(int* nums, int numsSize, int lower, int upper) {
    long* prefix = (long*)malloc((numsSize + 1) * sizeof(long));
    prefix[0] = 0;
    for (int i = 0; i < numsSize; i++) {
        prefix[i + 1] = prefix[i] + nums[i];
    }

    int result = countWhileMergeSort(prefix, 0, numsSize + 1, lower, upper);
    free(prefix);
    return result;
}

int main() {
    int nums[] = {-2, 5, -1};
    int size = sizeof(nums) / sizeof(nums[0]);
    int lower = -2, upper = 2;

    int result = countRangeSum(nums, size, lower, upper);
    printf("Count of Range Sum in [%d, %d] is: %d\n", lower, upper, result);

    return 0;
}

```


OUTPUT:

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

nums =
[0]

lower =
0

upper =
0

Output

1

Expected

1

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

nums =
[-2, 5, -1]

lower =
-2

upper =
2

Output

3

Expected

3

TRACING:

