Lab 2: Implementation of divide and conquer algorithms(quicksort, merge sort) using C++

Theory:

1. Divide and Conquer: Divide and Conquer is an algorithmic pattern. In algorithmic methods, the design is to tale a dispute on a huge input, break the input into minor pieces, decide the problem on each of the small pieces, and then merge the piecewise solutions into a global solution. This mechanism of solving the problem is called the Divide & Conquer Strategy. It consists of dispute using the following three steps.
    a. Divide: Break down the original problem into smaller subproblems. Each subproblem should represent a part of the overall problem. The goal is to divide the problem until no further division is possible.
    b. Conquer: Solve each of the smaller subproblems individually. If a subproblem is small enough (base case) we solve it directly without recursion. The goal is to find solutions for these subproblems independently.
    c. Merge: Combine the subproblems to get the final solution of the whole problem. Once the smaller subproblems are solved, we recursively combine their solutions to get the solution of larger problem.
2. Applications of Divide and Conquer Algorithm:
    a. Quicksort: It is a very efficient sorting algorithm, which is also known as a partition-exchange sort. It starts by selecting a pivot value from an array followed by dividing the rest of the array elements into two sub-arrays. The partition is made by comparing each of the elements with the pivot value. It compares whether the element holds a greater value or lesser value than the pivot and then sort the arrays recursively.
    Algorithm *:*
        i. Consider the first element of the array as pivot.
        ii. Define two variables left and right. Set left and right to first and last elements of the array respectively.
        iii. Increment left until array[left] > pivot then stop.
        iv. Decrement right until array[right] < pivot then stop.
        v. If left < right then exchange array[left] and array[right].
        vi. Repeat steps 3,4 & 5 until left > right.
        vii. Exchange the pivot element with array[right] element.
    b. Merge Sort: It is a sorting algorithm that follows the divide and conquer approach. It works by recursively dividing the input array into smaller subarrays and sorting those subarrays then merging them back together to obtain the sorted array. In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half and merge the sorted halves back together. The process is repeated until the array is sorted.
    Algorithm :
        i. Create two pointers, one for each sorted half.

ii. Initialize an empty temporary array to hold the merged result.
iii. Compare the elements at the pointers of the two halves:
iv. Copy the smaller element into the temporary array.
v. Move the pointer of the sub-list with the smaller element forward.
vi. Repeat step 3 until one of the sub-list is empty.
vii. Copy the remaining elements from the non-empty sub-list to the temporary array.
viii. Copy the elements back from the temporary array to the original list.

Observation:

1. Quicksort:

```
#include <iostream>
#include <vector>
#include <gettime.h>
using namespace std;

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

int partition(vector<int> &arr, int start, int end)
{
    int pivot = arr[end];
    int i = start - 1;

    for (int j = start; j <= end - 1; j++)
    {
        if (arr[j] <= pivot)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[end]);
    return (i + 1);
}

void quicksort(vector<int> &arr, int start, int end)
{
    if (start < end)
    {
        int pi = partition(arr, start, end);
        quicksort(arr, start, pi - 1);
        quicksort(arr, pi + 1, end);
    }
}

int main()
{
    vector<int> sizes;
```

```
    int start_size = 1000000;
    int increment = 500000;
    sizes.push_back(start_size);
    for (size_t i = 0; i < 4; i++)
    {
        sizes.push_back(sizes.back() + increment);
    }

    for (int size : sizes)
    {
        vector<int> arr;

        for (int i = 0; i < size; i++)
        {
            arr.push_back(rand());
        }

        long long time = getTime([&]()
                                 {
                                     quicksort(arr, 0, arr.size()); });
        cout << "Time taken to quicksort " << size << " items: " << time << "
nanoseconds\n";
    }
}
```

Output:

```
Time taken to quicksort 1000000 items: 359874600 nanoseconds
Time taken to quicksort 1500000 items: 611563600 nanoseconds
Time taken to quicksort 2000000 items: 1330628700 nanoseconds
Time taken to quicksort 2500000 items: 1269703000 nanoseconds
Time taken to quicksort 3000000 items: 1695711300 nanoseconds
```

2. Merge Sort:

```
#include <iostream>
#include <vector>
#include <gettime.h>
using namespace std;

vector<int> merge(vector<int> left, int pivot, vector<int> right)
{
    vector<int> arr;
    for (size_t i = 0; i < left.size(); i++)
    {
```

```
        arr.push_back(left[i]);
    }
    arr.push_back(pivot);
    for (size_t i = 0; i < right.size(); i++)
    {
        arr.push_back(right[i]);
    }
    return arr;
}

vector<int> mergesort(vector<int> arr)
{
    size_t size = arr.size();
    if (size == 0)
    {
        return {};
    }
    if (size == 1)
    {
        return arr;
    }
    if (size == 2)
    {
        if (arr[1] < arr[0])
        {
            int t = arr[0];
            arr[0] = arr[1];
            arr[1] = t;
        }
        return arr;
    }
    vector<int> left, right;
    size_t pivotIdx = size / 2;
    for (size_t i = 0; i < size; i++)
    {
        if (i != pivotIdx)
        {
            if (arr[i] <= arr[pivotIdx])
            {
                left.push_back(arr[i]);
            }
            else
            {
                right.push_back(arr[i]);
            }
```

```
        }
    }
    vector<int> leftSort = mergesort(left);
    int pivot = arr[pivotIdx];
    vector<int> rightSort = mergesort(right);
    return merge(leftSort, pivot, rightSort);
}

int main()
{
    vector<int> sizes;
    int start_size = 1000000;
    int increment = 500000;
    sizes.push_back(start_size);
    for (size_t i = 0; i < 4; i++)
    {
        sizes.push_back(sizes.back() + increment);
    }

    for (int size : sizes)
    {
        vector<int> arr;

        for (int i = 0; i < size; i++)
        {
            arr.push_back(rand());
        }

        long long time = getTime([&]()
                                 {
                                     mergesort(arr); });
        cout << "Time taken to mergesort " << size << " items: " << time << "
nanoseconds\n";
    }
}
```

Output:

```
Time taken to mergesort 10000 items: 67864200 nanoseconds
Time taken to mergesort 15000 items: 136451600 nanoseconds
Time taken to mergesort 20000 items: 213894800 nanoseconds
Time taken to mergesort 25000 items: 270726700 nanoseconds
Time taken to mergesort 30000 items: 390510200 nanoseconds
```

Conclusion:

In this way, we implemented divide and conquer algorithms(quicksort, merge sort) using C++.