

Name	Ayush Kedare
UID	2021300058
SUBJECT	DAA
EXPERIMENT NO.	2
AIM	Merge sort and Quick sort
ALGORITHM	<p>Merge sort:</p> <ol style="list-style-type: none"> 1. MERGE_SORT(arr, beg, end) 2. if beg < end 3. set mid = (beg + end)/2 4. MERGE_SORT(arr, beg, mid) 5. MERGE_SORT(arr, mid + 1, end) 6. MERGE (arr, beg, mid, end) 7. end of if 8. END MERGE_SORT <p>Quick sort:</p> <ol style="list-style-type: none"> 1. QUICKSORT (array A, start, end) 2. if (start < end) 3. p = partition(A, start, end) 4. QUICKSORT (A, start, p - 1) 5. QUICKSORT (A, p + 1, end)
PROGRAM	<pre>#include <iostream> #include <fstream> #include <cstdlib> #include <ctime> #include <bits/stdc++.h> using namespace std; int listt[100000]; void read() { ifstream fin("values.txt", ios::binary); for (long i = 0; i < 100000; i++) { fin.read((char *)&listt[i], sizeof(int)); } fin.close(); } void merge(int arr[], int p, int q, int r) {</pre>

```

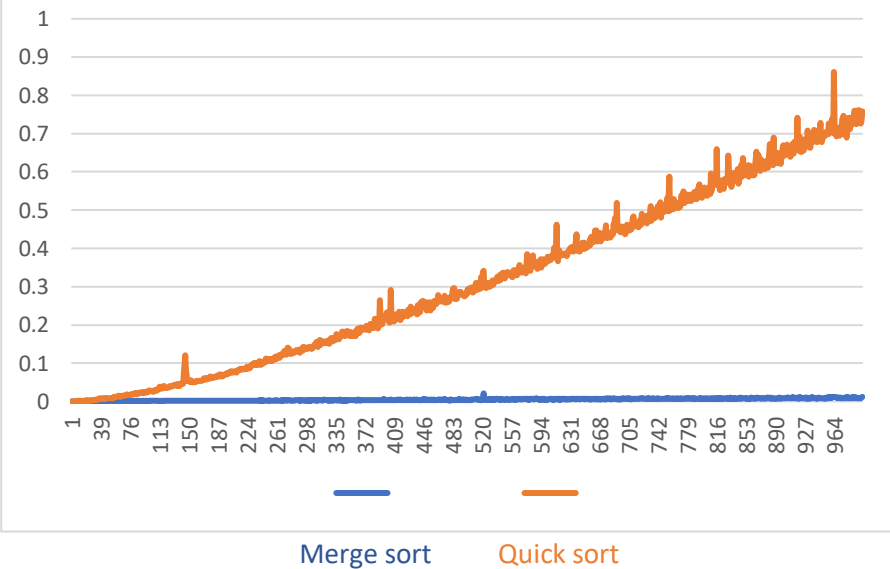
int n1 = q - p + 1;
int n2 = r - q;
int L[n1], M[n2];
for (int i = 0; i < n1; i++)
    L[i] = arr[p + i];
for (int j = 0; j < n2; j++)
    M[j] = arr[q + 1 + j];
int i, j, k;
i = 0;
j = 0;
k = p;
while (i < n1 && j < n2)
{
    if (L[i] <= M[j])
    {
        arr[k] = L[i];
        i++;
    }
    else
    {
        arr[k] = M[j];
        j++;
    }
    k++;
}
while (i < n1)
{
    arr[k] = L[i];
    i++;
    k++;
}
while (j < n2)
{
    arr[k] = M[j];
    j++;
    k++;
}
void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}
long partition(long left, long right)
{
    int pivot_element = listt[left];

```

```

int lb = left, ub = right;
int temp;
while (left < right)
{
    while (listt[left] <= pivot_element)
        left++;
    while (listt[right] > pivot_element)
        right--;
    if (left < right)
    {
        temp = listt[left];
        listt[left] = listt[right];
        listt[right] = temp;
    }
}
listt[lb] = listt[right];
listt[right] = pivot_element;
return right;
}
void quickSort(long left, long right)
{
    if (left < right)
    {
        long pivot = partition(left, right);
        quickSort(left, pivot - 1);
        quickSort(pivot + 1, right);
    }
}
int main()
{
    clock_t t1, t2, t3, t4;
    read();
    int num = 100;
    ofstream output("./output2.csv");
    output << "block_size,insertion,selection\n";
    for (int i = 0; i < 1000; i++)
    {
        t1 = clock();
        mergeSort(listt, 0, num - 1);
        t2 = clock();
        t3 = clock();
        quickSort(0, num - 1);
        t4 = clock();
        double mergetime = double(t2 - t1) /
            double(CLOCKS_PER_SEC);
        double quicktime = double(t4 - t3) /
            double(CLOCKS_PER_SEC);
        cout << endl;
        output << i+1 << ", " << mergetime <<
            setprecision(5) << '\t';
        output << fixed << ", " << quicktime <<

```

	<pre>setprecision(5) <<"\n"; cout << i + 1 << " " << fixed << mergetime << "\t"; cout << fixed << quicktime; num += 100; } return 0; }</pre> <p>Values.cpp :</p> <pre>#include <iostream> #include <cstdlib> using namespace std; int main() { for(int i = 0; i < 100000; i++) { cout<<"\n " <<rand(); } }</pre>
OBSERVATION (SNAPSHOT)	 <p>The graph shows two data series: Merge sort (blue line) and Quick sort (orange line). The x-axis represents the size of the data, ranging from 1 to 964. The y-axis represents the running time, ranging from 0 to 1. The Merge sort line is nearly flat, indicating a very slow increase in running time. The Quick sort line is steep and noisy, indicating a much faster increase in running time as the data size increases.</p> <p>Foe quick sort as the size increases the running time also increases and the slope of the graph becomes steeper. For merge sort as well, as the size increases the running time increases but this is at a very slower rate which is evident from the lesser steep slope.</p>
Conclusion	By performing the above experiment I understood how the merge sort and quick sort work and experimented them and found that it depends on the size of the data.