```cpp
// Number Generator
#include <bits/stdc++.h>
#include <iostream>

using namespace std;

int main()
{
	srand(static_cast<unsigned int>(time(0)));
	const int numCount = 10000;

	ofstream outFile("number.txt");
	if (outFile.is_open())
	{
	for (int i = 0; i < numCount; i++)
	{
		int randomNum = rand() % 1000;
		outFile << randomNum <<endl;
	}
	 outFile.close();
	cout << "Random numbers written to number.txt." << endl;
	}
	else
	{
	cerr << "Unable to open file for writing." << endl;
	return 1;
	}

	cout<<numCount<<endl;

	return 0;
}
```

```cpp
// Quick Sort
 #include <iostream>
#include <vector>
#include <ctime>
#include <fstream>
#include <chrono>

using namespace std;

vector<int> a ;
```

```
void Interchange(vector<int>& a, int i, int j)
{
    int p = a[i];
    a[i] = a[j];
    a[j] = p;
}

int partition(vector<int>& a, int m, int p)
{
    int v = a[m]; // Pivot element
    int i = m ; // Start from the next element
    int j = p;

    while (i<=j)
    {
        while (a[i] <= v)
        { i++; }

        while (a[j] > v)
        {  j--; }

        if (i < j)
        { Interchange(a, i, j); }
    }
    a[m] = a[j];
    a[j] = v;

    return j;
}

void QuickSort(int p, int q)
{
    if (p < q)
    {
        int j = partition(a, p, q);
        QuickSort(p, j - 1);
        QuickSort(j + 1, q);
    }
}

int main()
{
    ifstream inFile("number.txt");
```

```
    if (inFile.is_open())
    {
        int num;
        while (inFile >> num)
        {
            a.push_back(num);
        }
        inFile.close();
    }
    else
    {
        cerr << "Unable to open file for reading." << endl;
        return 1;
    }

    int n = a.size();
    cout << "Vector Size: " << n << endl;

    // Start timing
    auto start = chrono::high_resolution_clock::now();

    // Execute the function to measure
    QuickSort(0, n - 1);

    // End timing
    auto End = chrono::high_resolution_clock::now();

    // Calculate the duration
    chrono::duration<double> duration = End - start;

    // Output the time taken in seconds
    cout << "Time taken: " << duration.count() << " seconds" << endl;

    return 0;
}
```

// Merge Sort

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>
```

```
#include <chrono>

using namespace std;

vector<int> a;
vector<int> b;

void Merge(int low, int mid, int high)
{
      int h = low;
      int i = low;
      int j = mid+1;

      while((h<=mid) && (j<=high))
    {
      if(a[h]<=a[j])
      {
            b[i] = a[h];
            h = h+1;
      }
      else
      {
            b[i] = a[j];
            j = j+1;
      }
      i = i+1;
      }

      if(h>mid)
      {
      for(int k=j; k<=high; k++)
      {
            b[i]= a[k];
            i = i+1;
      }
      }
      else
      {
      for(int k=h; k<=mid; k++)
      {
            b[i] = a[k];
            i = i+1;
      }
      }
```

```
      for(int k=low; k<= high; k++)
      {
      a[k] = b[k];
    }
}
void MergeSort(int low , int high)
{
      if(low<high)
      {
      int mid = floor( (low+high)/2 );

      MergeSort(low,mid);
      MergeSort(mid+1, high);

      Merge(low, mid, high);
      }
}


int main()
{
      ifstream inFile("number.txt");
      if (inFile.is_open())
      {
      int num;
      while (inFile >> num)
      {
            a.push_back(num);
      }
      inFile.close();
      }
      else
      {
      cerr << "Unable to open file for reading." << endl;
      return 1;
      }

      int n = a.size();
      b.resize(n);
      cout << "Vector Size: " << n << endl;

      // Start timing
      auto start = chrono::high_resolution_clock::now();
```

```
        // Execute the function to measure
        MergeSort(0, n - 1);

        // End timing
        auto End = chrono::high_resolution_clock::now();

        // Calculate the duration in nanoseconds
        chrono::duration<double> duration = End - start;

        // Output the time taken in seconds
        cout << "Time taken: " << duration.count() << " seconds" << endl;

        return 0;
}
```

QuickSort VS MergeSort Time Complexity:

Data Size	QuickSort	MergeSort
0	0	0
1500	0.0011964	0.0009998
2000	0.0009959	0.0009971
10000	0.0019927	0.0021806
25000	0.0157453	0.0169219
33333	0.0157652	0.0157218
50000	0.0156221	0.0227456
75000	0.0613803	0.038278
100000	0.0323791	0.0476355
125000	0.0495895	0.0633047
142857	0.0666323	0.0835348
142857	0.0904196	0.0468647
166666	0.0809924	0.0696443
200000	0.125742	0.0939801
200000	0.0989485	0.0830926
250000	0.172974	0.101723
333333	0.266912	0.160627
500000	0.575388	0.220155
1000000	2.20308	0.575299
2000000	7.93392	0.985382
3000000	17.5057	1.44732
4000000	30.1109	2.07954
5000000	46.5913	2.4839
6000000	66.8696	3.03795
7000000	95.5196	3.65044
10000000	186.685	5.45769