Design Analysis of Algorithm

Assignment 3:

Name: Dilawer Khalid **Sap:49085**

Bubble sort Algorithm:

```
#include <iostream>
#include <ctime>
#include <algorithm>
using namespace std;

void bubbleSort(int arr[], int n) {
  for (int i = 0; i < n - 1; ++i) {
    for (int j = 0; j < n - i - 1; ++j) {
      if (arr[j] > arr[j + 1]) {
        swap(arr[j], arr[j + 1]);
      }
    }
  }
}

int main() {
  const int n = 1000;
  int best_case[n], average_case[n], worst_case[n];

  for (int i = 0; i < n; ++i) best_case[i] = i;          // Best case: sorted array
  for (int i = 0; i < n; ++i) worst_case[i] = n - i;       // Worst case: reverse sorted array
  for (int i = 0; i < n; ++i) average_case[i] = rand() % n;  // Average case: random array

  clock_t start = clock();
  bubbleSort(best_case, n);
```

```
    cout << "Bubble Sort Best Case: " << double(clock() - start) / CLOCKS_PER_SEC << " seconds\n";


    start = clock();
    bubbleSort(average_case, n);
    cout << "Bubble Sort Average Case: " << double(clock() - start) / CLOCKS_PER_SEC << "
seconds\n";


    start = clock();
    bubbleSort(worst_case, n);
    cout << "Bubble Sort Worst Case: " << double(clock() - start) / CLOCKS_PER_SEC << "
seconds\n";


    return 0;
}
```

Selection Sort:

```
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; ++i) {
        int min_idx = i;
        for (int j = i + 1; j < n; ++j) {
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }
        swap(arr[i], arr[min_idx]);
    }
}


int main() {
    const int n = 1000;
    int best_case[n], average_case[n], worst_case[n];
    for (int i = 0; i < n; ++i) best_case[i] = i;
    for (int i = 0; i < n; ++i) worst_case[i] = n - i;
    for (int i = 0; i < n; ++i) average_case[i] = rand() % n;

    clock_t start = clock();
    selectionSort(best_case, n);
    cout << "Selection Sort Best Case: " << double(clock() - start) / CLOCKS_PER_SEC << "
seconds\n";

    start = clock();
    selectionSort(average_case, n);
    cout << "Selection Sort Average Case: " << double(clock() - start) / CLOCKS_PER_SEC << "
seconds\n";

    start = clock();
    selectionSort(worst_case, n);
    cout << "Selection Sort Worst Case: " << double(clock() - start) / CLOCKS_PER_SEC << "
seconds\n";

    return 0;
}
```

Merge Sort:

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

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

int main() {
    const int n = 1000;
    int best_case[n], average_case[n], worst_case[n];
    for (int i = 0; i < n; ++i) best_case[i] = i;
    for (int i = 0; i < n; ++i) worst_case[i] = n - i;
    for (int i = 0; i < n; ++i) average_case[i] = rand() % n;

    clock_t start = clock();
    mergeSort(best_case, 0, n - 1);
    cout << "Merge Sort Best Case: " << double(clock() - start) / CLOCKS_PER_SEC << " seconds\n";
    start = clock();

    mergeSort(average_case, 0, n - 1);

    cout << "Merge Sort Average Case: " << double(clock() - start) / CLOCKS_PER_SEC << "
seconds\n";


    start = clock();

    mergeSort(worst_case, 0, n - 1);

    cout << "Merge Sort Worst Case: " << double(clock() - start) / CLOCKS_PER_SEC << " seconds\n";


    return 0;

}
```

Quick Sort:

```
int partition(int arr[], int low, int high) {

    int pivot = arr[high];

    int i = low - 1;

    for (int j = low; j < high; ++j) {

        if (arr[j] < pivot) {

            ++i;

            swap(arr[i], arr[j]);

        }

    }

    swap(arr[i + 1], arr[high]);

    return i + 1;

}


void quickSort(int arr[], int low, int high) {

    if (low < high) {

        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int main() {
    const int n = 1000;
    int best_case[n], average_case[n], worst_case[n];
    for (int i = 0; i < n; ++i) best_case[i] = i;
    for (int i = 0; i < n; ++i) worst_case[i] = n - i;
    for (int i = 0; i < n; ++i) average_case[i] = rand() % n;

    clock_t start = clock();
    quickSort(best_case, 0, n - 1);
    cout << "Quick Sort Best Case: " << double(clock() - start) / CLOCKS_PER_SEC << " seconds\n";

    start = clock();
    quickSort(average_case, 0, n - 1);
    cout << "Quick Sort Average Case: " << double(clock() - start) / CLOCKS_PER_SEC << "
seconds\n";

    start = clock();
    quickSort(worst_case, 0, n - 1);
    cout << "Quick Sort Worst Case: " << double(clock() - start) / CLOCKS_PER_SEC << " seconds\n";

    return 0;
}
```