1. **Merge Sort**

```
#include <iostream>
#include <chrono>
using namespace std::chrono;
using namespace std;

void merge(int* a, int l, int m, int r){
    int n = r - l + 1;
    int* result = new int[n];

    int i, j, k;
    i = l, j = m + 1, k = 0;

    while(i <= m && j <= r){
        if(a[i] < a[j]){
            result[k++] = a[i++];
        }
        else{
            result[k++] = a[j++];
        }
    }

    // If a couple of elements were left out in the left or right array
    // due to i > m or j > r, fill up the rest of the results array
    // with the remaining elements
    while(i <= m){
        result[k++] = a[i++];
    }
    while(j <= r){
        result[k++] = a[j++];
    }

    for(k = 0; k < n; k++){
        a[k + l] = result[k];
    }

    delete[] result;
}

void mergeSort(int* a, int l, int r){
    if(l >= r){
        return;
    }

    int mid = (l + r) / 2;
    mergeSort(a, l, mid);
    mergeSort(a, mid + 1, r);
```

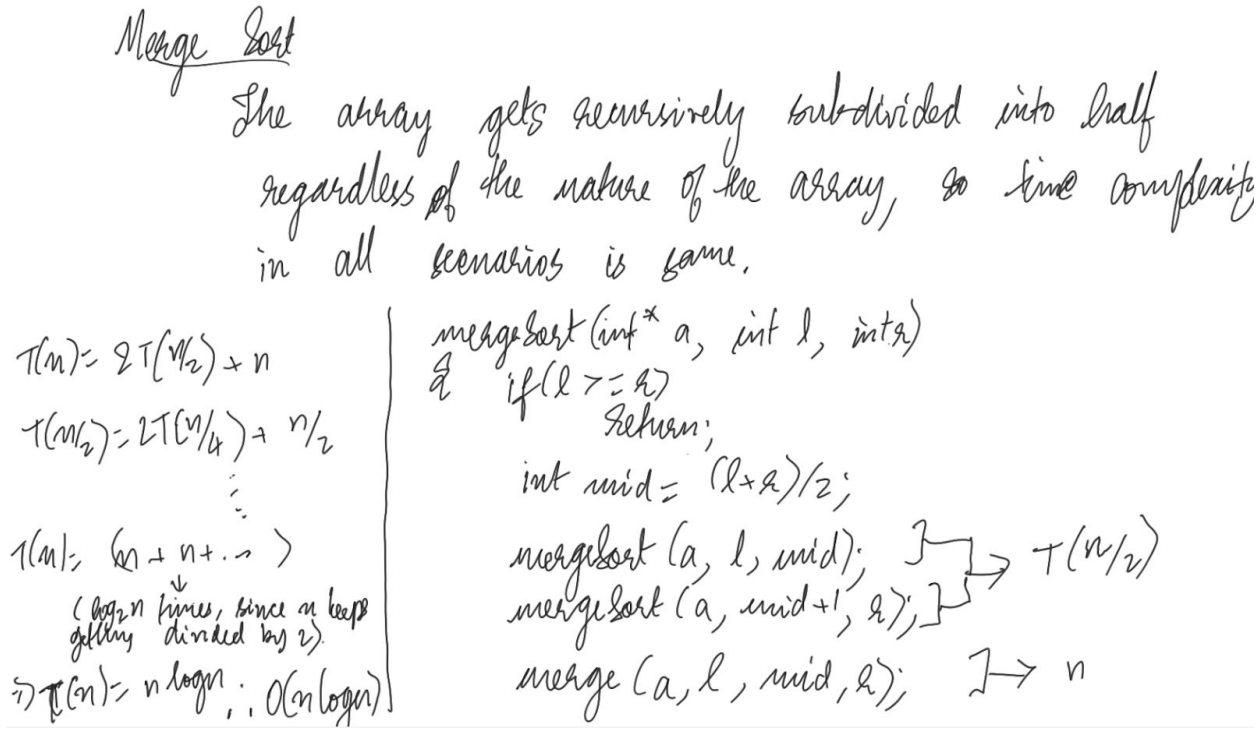
```
    merge(a, l, mid, r);
}

int main() {
        int n;
        cout << "Enter array length:\n";
        cin >> n;

        int* a = new int[n];
        cout << "Enter elements:\n";

        for(int i = 0; i < n; i++){
            cin >> a[i];
        }

        auto start = high_resolution_clock::now();
        mergeSort(a, 0, n - 1);
        auto stop = high_resolution_clock::now();

        cout << "Sorted array:\n";
        for(int i = 0; i < n; i++){
            cout << a[i] << " ";
        }

        auto duration = duration_cast<microseconds>(stop - start);
        cout << "\nTime taken to sort: " << duration.count()  << " microseconds\n";

        delete[] a;

        return 0;
}
```

Output:

```
vaidehee@penguin:~/sem-4/daa$ ./mergeSort
Enter array length:
8
Enter elements:
14 7 3 12 9 11 6 12
Sorted array:
3 6 7 9 11 12 12 14
Time taken to sort: 21 microseconds
vaidehee@penguin:~/sem-4/daa$ ./mergeSort
Enter array length:
8
Enter elements:
3 6 7 9 11 12 12 14
Sorted array:
3 6 7 9 11 12 12 14
Time taken to sort: 21 microseconds
```

```
vaidehee@penguin:~/sem-4/daa$ ./mergeSort
Enter array length:
8
Enter elements:
14 12 12 11 9 7 6 3
Sorted array:
3 6 7 9 11 12 12 14
Time taken to sort: 21 microseconds
```

Time Complexity Analysis



2. **Quick Sort**

```
#include <iostream>
#include <chrono>
using namespace std::chrono;
using namespace std;

int partition(int* a, int l, int r){
   int p = l;

   //find pivot i.e. find the position at which the element currently at
   // a[l] needs to be in the sorted array by counting the number of elements
   // in the array that are smaller than or equal to it
   for(int i = l + 1; i <= r; i++){
      if(a[i] <= a[l]){
         p++;
      }
   }

   //move a[l] to pivot
   int t = a[l];
   a[l] = a[p];
   a[p] = t;
```

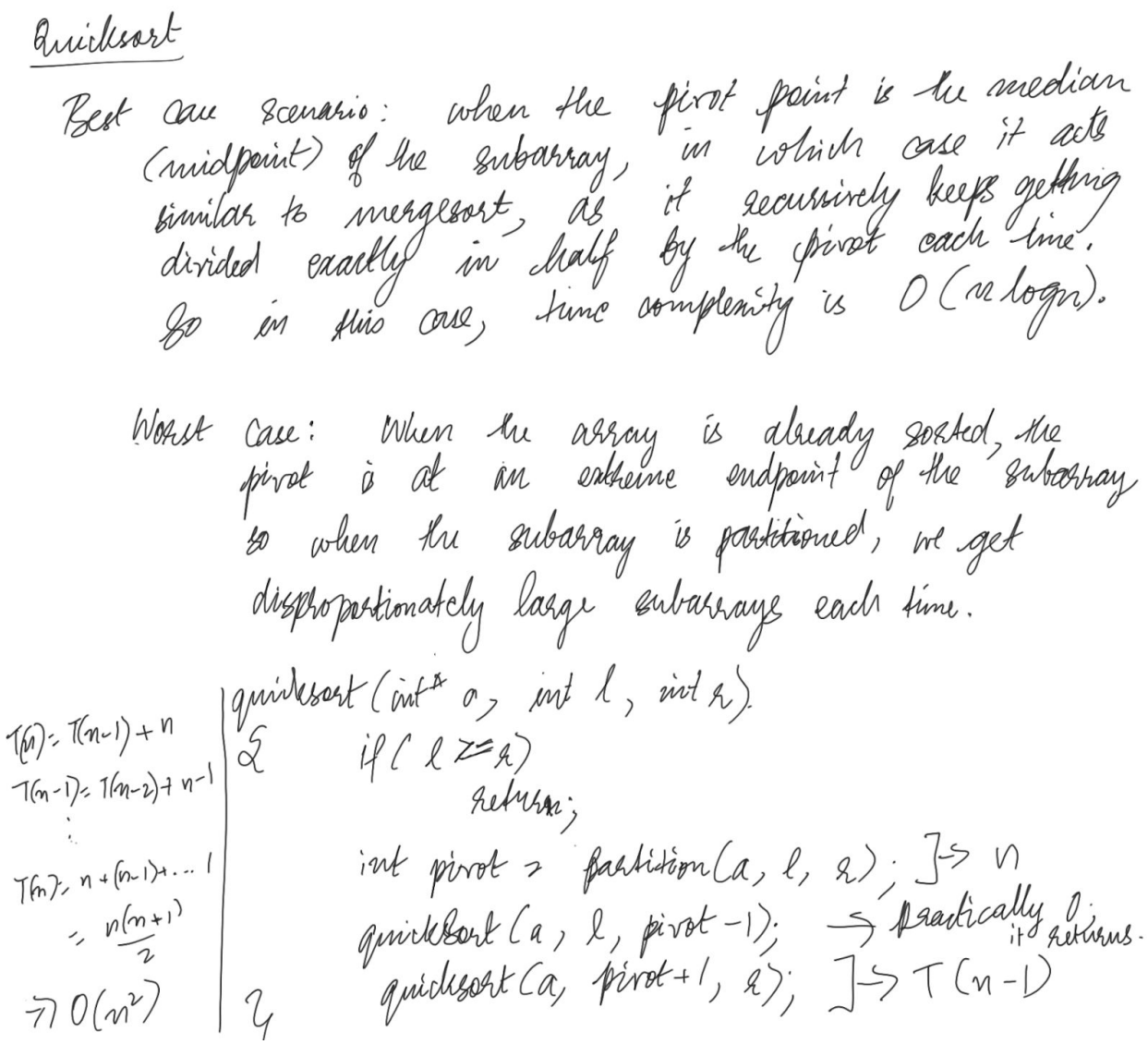
```
    //partition
    for(int i = l, j = r; i < p && j > p; i++, j--){
        while(a[i] <= a[p] && i < p){
            i++;
        }
        while(a[j] > a[p] && j > p){
            j--;
        }
        if(i < p && j > p){
            int t = a[i];
            a[i] = a[j];
            a[j] = t;
        }
    }

    return p;
}

void quickSort(int* a, int l, int r) {
    if(l >= r){
        return;
    }

    int pivot = partition(a, l, r);

    quickSort(a, l, pivot - 1);
    quickSort(a, pivot + 1, r);
}

int main() {
        int n;
        cout << "Enter array length:\n";
        cin >> n;

        int* a = new int[n];
        cout << "Enter elements:\n";

        for(int i = 0; i < n; i++){
            cin >> a[i];
        }

        auto start = high_resolution_clock::now();
        quickSort(a, 0, n - 1);
        auto stop = high_resolution_clock::now();

        cout << "Sorted array:\n";
        for(int i = 0; i < n; i++){
            cout << a[i] << " ";
```

```
        }

        auto duration = duration_cast<microseconds>(stop - start);
        cout << "\nTime taken to sort: " << duration.count()  << " microseconds\n";

        delete[] a;

        return 0;
}
```

Output

```
vaidehee@penguin:~/sem-4/daa$ ./quickSort
Enter array length:
8
Enter elements:
14 7 3 12 9 11 6 12
Sorted array:
3 6 7 9 11 12 12 14
Time taken to sort: 2 microseconds
vaidehee@penguin:~/sem-4/daa$ ./quickSort
Enter array length:
8
Enter elements:
14 12 12 11 9 7 6 3
Sorted array:
3 6 7 9 11 12 12 14
Time taken to sort: 2 microseconds
vaidehee@penguin:~/sem-4/daa$ ./quickSort
Enter array length:
8
Enter elements:
3 6 7 9 11 12 12 14
Sorted array:
3 6 7 9 11 12 12 14
Time taken to sort: 16 microseconds
```

Time Complexity Analysis



3. Bubble Sort

```
#include <iostream>
#include <chrono>
using namespace std::chrono;
using namespace std;

void bubbleSort(int* a, int n){
  if(n <= 1){
    return;
  }

  bool isSorted;
```

```
  for(int i = 0; i < n; i++){
    isSorted = true;
    for(int j = 0; j < n - i - 1; j++){
      if(a[j] > a[j + 1]){
        isSorted = false;

        int t = a[j];
        a[j] = a[j + 1];
        a[j + 1] = t;
      }
    }
    if(isSorted){
      break;
    }
  }
}

int main() {
        int n;
        cout << "Enter array length:\n";
        cin >> n;

        int* a = new int[n];
        cout << "Enter elements:\n";

        for(int i = 0; i < n; i++){
            cin >> a[i];
        }

        auto start = high_resolution_clock::now();
        bubbleSort(a, n);
        auto stop = high_resolution_clock::now();

        cout << "Sorted array:\n";
        for(int i = 0; i < n; i++){
            cout << a[i] << " ";
        }

        auto duration = duration_cast<microseconds>(stop - start);
        cout << "\nTime taken to sort: " << duration.count()  << " microseconds\n";

        delete[] a;

        return 0;
}
```

Output

```
vaidehee@penguin:~/sem-4/daa$ g++ -o bubbleSort bubbleSort.cpp
vaidehee@penguin:~/sem-4/daa$ ./bubbleSort
Enter array length:
8
Enter elements:
14 7 3 12 9 11 6 12
Sorted array:
3 6 7 9 11 12 12 14
Time taken to sort: 1 microseconds
vaidehee@penguin:~/sem-4/daa$ ./bubbleSort
Enter array length:
8
Enter elements:
3 6 7 9 11 12 12 14
Sorted array:
3 6 7 9 11 12 12 14
Time taken to sort: 0 microseconds
vaidehee@penguin:~/sem-4/daa$ ./bubbleSort
Enter array length:
8
Enter elements:
14 12 12 11 9 7 6 3
Sorted array:
3 6 7 9 11 12 12 14
Time taken to sort: 1 microseconds
```

Time Complexity Analysis

BubbleSort

It has 2 nested for loops, so in the worst case scenario it has to completely iterate over both loops, in which case time complexity is $O(n^2)$.

In best case, when the array is already sorted, it just iterates once, in which case isSorted turns out to be true in the 1st iteration itself. In this case, time complexity is $O(n)$.

4. **Selection Sort**

```
#include <iostream>
#include <chrono>
using namespace std::chrono;
using namespace std;

void selectionSort(int* a, int n){
  if(n <= 1){
    return;
  }

  int minIndex;

  for(int i = 0; i < n; i++){
    minIndex = i;
    for(int j = i; j < n; j++){
      if(a[j] < a[minIndex]){
        minIndex = j;
      }
    }
    int t = a[i];
    a[i] = a[minIndex];
    a[minIndex] = t;
  }
}

int main() {
        int n;
        cout << "Enter array length:\n";
        cin >> n;

        int* a = new int[n];
        cout << "Enter elements:\n";

        for(int i = 0; i < n; i++){
            cin >> a[i];
        }

        auto start = high_resolution_clock::now();
        selectionSort(a, n);
        auto stop = high_resolution_clock::now();

        cout << "Sorted array:\n";
        for(int i = 0; i < n; i++){
            cout << a[i] << " ";
        }
```

```
    auto duration = duration_cast<microseconds>(stop - start);
    cout << "\nTime taken to sort: " << duration.count()  << " microseconds\n";

    delete[] a;

    return 0;
}
```

Output

```
vaidehee@penguin:~/sem-4/daa$ ./selectionSort
Enter array length:
8
Enter elements:
14 7 3 12 9 11 6 12
Sorted array:
3 6 7 9 11 12 12 14
Time taken to sort: 1 microseconds
vaidehee@penguin:~/sem-4/daa$ ./selectionSort
Enter array length:
8
Enter elements:
3 6 7 9 11 12 12 14
Sorted array:
3 6 7 9 11 12 12 14
Time taken to sort: 1 microseconds
vaidehee@penguin:~/sem-4/daa$ ./selectionSort
Enter array length:
8
Enter elements:
14 12 12 11 9 7 6 3
Sorted array:
3 6 7 9 11 12 12 14
Time taken to sort: 2 microseconds
```

Time Complexity Analysis

Selection Sort

The algorithm always goes till the end to check if the 1st element of the right subarray is the minimum or not. So time complexity is $O(n^2)$ (since 2 nested for loops).