

Bubble sort

```
#include<iostream>
#include<stdlib.h>
#include<omp.h>
using namespace std;
```

```
void bubble(int *, int);
void swap(int &, int &);
```

```
void bubble(int *a, int n)
{
    for( int i = 0; i < n; i++ )
    {
        int first = i % 2;

        #pragma omp parallel for shared(a,first)
        for( int j = first; j < n-1; j += 2 )
        {
            if( a[j] > a[j+1] )
            {
                swap( a[j], a[j+1] );
            }
        }
    }
}
```

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

```
int main(){
    int *a,n;
    cout<<"\nEnter size of Array : ";
    cin>>n;
    a=new int[n];
    cout<<"\nEnter elements : \n";
    for(int i=0;i<n;i++)
    {
        cin>>a[i];
    }

    bubble(a,n);

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

```

}
return 0;
}

```

The screenshot shows a C++ IDE with two files: `bubble.cpp` and `merge.cpp`. The `bubble.cpp` file contains the following code:

```

17 //parallel omp parallel for shared(a,i,j)
18 for( int j = first; j < n-1; j += 2 )
19 {
20     if( a[ j ] > a[ j+1 ] )
21     {
22         swap( a[ j ], a[ j+1 ] );
23     }
24 }
25 }
26
27 void swap(int &a, int &b)
28 {
29     int temp;
30     temp=a;
31     a=b;
32     b=temp;
33 }
34
35 int main(){
36     int *a,n;
37     cout<<"\nEnter size of Array : ";
38     cin>>n;
39     a=new int[n];
40     cout<<"\nEnter elements : \n";
41     for(int i=0;i<n;i++)
42     {
43         cin>>a[i];
44     }
45
46     bubble(a,n);
47
48     cout<<"\nSorted array is : \n";
49     for(int i=0;i<n;i++)
50     {
51         cout<<a[i]<<endl;
52     }
53     return 0;
54 }
55
56
57

```

The terminal output shows the execution of the program:

```

mmcoe@mmcoe-System-Product-Name:~$ g++ -fopenmp bubble.cpp
mmcoe@mmcoe-System-Product-Name:~$ ./a.out
Enter size of Array : 5
Enter elements :
5
3
7
1
Sorted array is :
1
3
5
7
mmcoe@mmcoe-System-Product-Name:~$

```

Merge Sort

```

#include<iostream>
#include<stdlib.h>
#include<omp.h>
using namespace std;

```

```

void mergesort(int a[], int i, int j);
void merge(int a[], int i1, int j1, int i2, int j2);

```

```

void mergesort(int a[], int i, int j)
{

```

```

    int mid;
    if (i < j)
    {
        mid = (i + j) / 2;

```

```

#pragma omp parallel sections
{
    #pragma omp section
    {
        mergesort(a, i, mid);
    }

```

```

#pragma omp section

```

```

        {
            mergesort(a, mid + 1, j);
        }
    }
    merge(a, i, mid, mid + 1, j);
}
}

```

```

void merge(int a[], int i1, int j1, int i2, int j2)

```

```

{
    int temp[1000];
    int i, j, k;
    i = i1;
    j = i2;
    k = 0;

    cout << "\nMerging: ";
    for (int x = i1; x <= j1; x++)
    {
        cout << a[x] << " ";
    }
    cout << "and ";
    for (int x = i2; x <= j2; x++)
    {
        cout << a[x] << " ";
    }
    cout << endl;

    while (i <= j1 && j <= j2)
    {
        if (a[i] < a[j])
        {
            temp[k++] = a[i++];
        }
        else
        {
            temp[k++] = a[j++];
        }
    }
    while (i <= j1)
    {
        temp[k++] = a[i++];
    }
    while (j <= j2)
    {
        temp[k++] = a[j++];
    }
    for (i = i1, j = 0; i <= j2; i++, j++)
    {
        a[i] = temp[j];
    }
}

```

```

cout << "Result after merging: ";

```

```

    for (int x = i1; x <= j2; x++)
    {
        cout << a[x] << " ";
    }
    cout << endl;
}

int main()
{
    int *a, n, i;
    cout << "\nEnter size of Array : ";
    cin >> n;
    a = new int[n];
    cout << "\nEnter elements : \n";
    for (i = 0; i < n; i++)
    {
        cin >> a[i];
    }
    mergesort(a, 0, n - 1);
    cout << "\nSorted array is : ";
    for (i = 0; i < n; i++)
    {
        cout << a[i] << " ";
    }

    return 0;
}

```

The screenshot shows a C++ IDE with a file named `merge.cpp` open. The code implements a merge sort algorithm using OpenMP for parallelization. The terminal window shows the execution of the program, which prompts the user to enter the size of the array (5) and the elements (5, 4, 3, 7, 1). The output shows the merging process and the final sorted array: 1 3 4 5 7.

```

home > mmcoe > merge.cpp
1  #include<iostream>
2  #include<stdlib.h>
3  #include<omp.h>
4  using namespace std;
5
6  void mergesort(int a[], int i, int j);
7  void merge(int a[], int i1, int j1, int
8
9  void mergesort(int a[], int i, int j)
10 {
11     int mid;
12     if (i < j)
13     {
14         mid = (i + j) / 2;
15
16         #pragma omp parallel sections
17         {
18             #pragma omp section
19             {
20                 mergesort(a, i, mid);
21             }
22             #pragma omp section
23             {
24                 mergesort(a, mid + 1, j);
25             }
26         }
27         merge(a, i, mid, mid + 1, j);
28     }
29 }
30
31
32 void merge(int a[], int i1, int j1, int i2, int j2)
33 {
34     int temp[1000];
35     int i, j, k;
36     i = i1;
37     j = i2;
38     k = 0;
39
40     cout << "\nMerging: ";
41     for (int x = i1; x <= j1; x++)
42     {

```

```

mmcoe@mmcoe-System-Product-Name: ~
mmcoe@mmcoe-System-Product-Name:~$ g++ -fopenmp merge.cpp
mmcoe@mmcoe-System-Product-Name:~$ ./a.out
Enter size of Array : 5
Enter elements :
5
4
3
7
1
Merging:
Merging: 5 and 4
Result after merging: 4 5
Merging: 4 5 and 3
Result after merging: 3 4 5
7 and 1
Result after merging: 1 7
Merging: 3 4 5 and 1 7
Result after merging: 1 3 4 5 7
Sorted array is : 1 3 4 5 7 mmcoe@mmcoe-System-Product-Name:~$

```