# Assignment No-2

**Bubble sort –**

## Program-

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

void bubble(vector <int>& arr); void swap(int \*a, int \*b)

{

int test; test=\*a;

\*a=\*b;

\*b=test;

}

void bubble(vector <int>& arr)

{

bool isSorted=false; while(!isSorted)

{

isSorted=true;

#pragma omp parallel for

for( int i = 0; i<arr.size()-1; i+=2 )

{

if( arr[i] > arr[i+1] )

{

swap(&arr[i],&arr[i+1]); isSorted=false;

}

}

#pragma omp parallel for

for( int i = 1; i<arr.size()-1; i+=2 )

{

if( arr[i] > arr[i+1] )

{

swap(&arr[i],&arr[i+1]); isSorted=false;

}

}

}

}

int main()

{

double start, end;

vector<int> arr={4,5,11,9,6,1,2,0};

//measure performance start=omp\_get\_wtime(); bubble(arr); end=omp\_get\_wtime();

cout<<"\n Start time="<<start; cout<<"\n End time="<<end; cout<<"\n sorted array is="; for(int i=0;i<arr.size();i++)

{

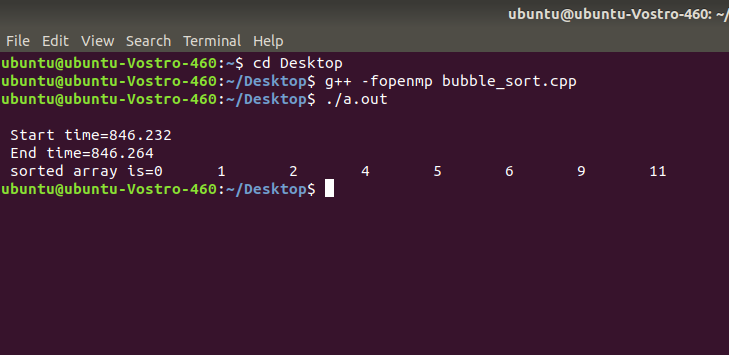
cout<<arr[i]<<"\t";

}

cout<<"\n"; return 0;

}

## OUTPUT-



**Merge sort-**

Program-

#include<iostream> #include <string.h> #include <stdlib.h> #include "omp.h"

#define MAX\_SIZE 1000

void generate\_list(int \* x, int n) { int i,j,t;

std::cout<<"\n Enter the element:"; for (i = 0; i < n; i++)

std::cin>>x[i];

}

void print\_list(int \* x, int n)

{

int i;

for (i = 0; i < n; i++) { std::cout<<x[i]<<" ";

}

}

void merge(int \* X, int n, int \* tmp) { int i = 0;

int j = n/2; int ti = 0;

while (i<n/2 && j<n) { if (X[i] < X[j]) {

tmp[ti] = X[i]; ti++; i++;

} else {

tmp[ti] = X[j]; ti++; j++;

}

}

while (i<n/2) { /\* finish up lower half \*/ tmp[ti] = X[i];

ti++; i++;

}

while (j<n) { /\* finish up upper half \*/ tmp[ti] = X[j];

ti++; j++;

}

memcpy(X, tmp, n\*sizeof(int));

} // end of merge()

void mergesort(int \* X, int n, int \* tmp)

{

if (n < 2) return;

#pragma omp task firstprivate (X, n, tmp) mergesort(X, n/2, tmp);

#pragma omp task firstprivate (X, n, tmp) mergesort(X+(n/2), n-(n/2), tmp);

#pragma omp taskwait

/\* merge sorted halves into sorted list \*/ merge(X, n, tmp);

}

int main()

{

int n = 5;

double start, stop;

int data[MAX\_SIZE], tmp[MAX\_SIZE];

generate\_list(data, n);

std::cout<<"\nList Before Sorting...\n"; print\_list(data, n);

start = omp\_get\_wtime(); #pragma omp parallel

{

#pragma omp single mergesort(data, n, tmp);

}

stop = omp\_get\_wtime(); std::cout<<"\nList After Sorting...\n"; print\_list(data, n); std::cout<<"\nTime: "<<stop-start;

}

## OUTPUT-

