Name : Nikhil Khade

Roll No. : COBA14

Lab : HPC 2 (a)

Input :

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

void bubble(int array[], int n){
   for (int i = 0; i < n - 1; i++){
      for (int j = 0; j < n - i - 1; j++){
         if (array[j] > array[j + 1]) swap(array[j], array[j + 1]);
      }
   }
}
void pBubble(int array[], int n){
   //Sort odd indexed numbers
   for(int i = 0; i < n; ++i){
      #pragma omp for
      for (int j = 1; j < n; j += 2){
      if (array[j] < array[j-1])
      {
        swap(array[j], array[j - 1]);
      }
   }
   // Synchronize
   #pragma omp barrier
   //Sort even indexed numbers
   #pragma omp for
   for (int j = 2; j < n; j += 2){
```

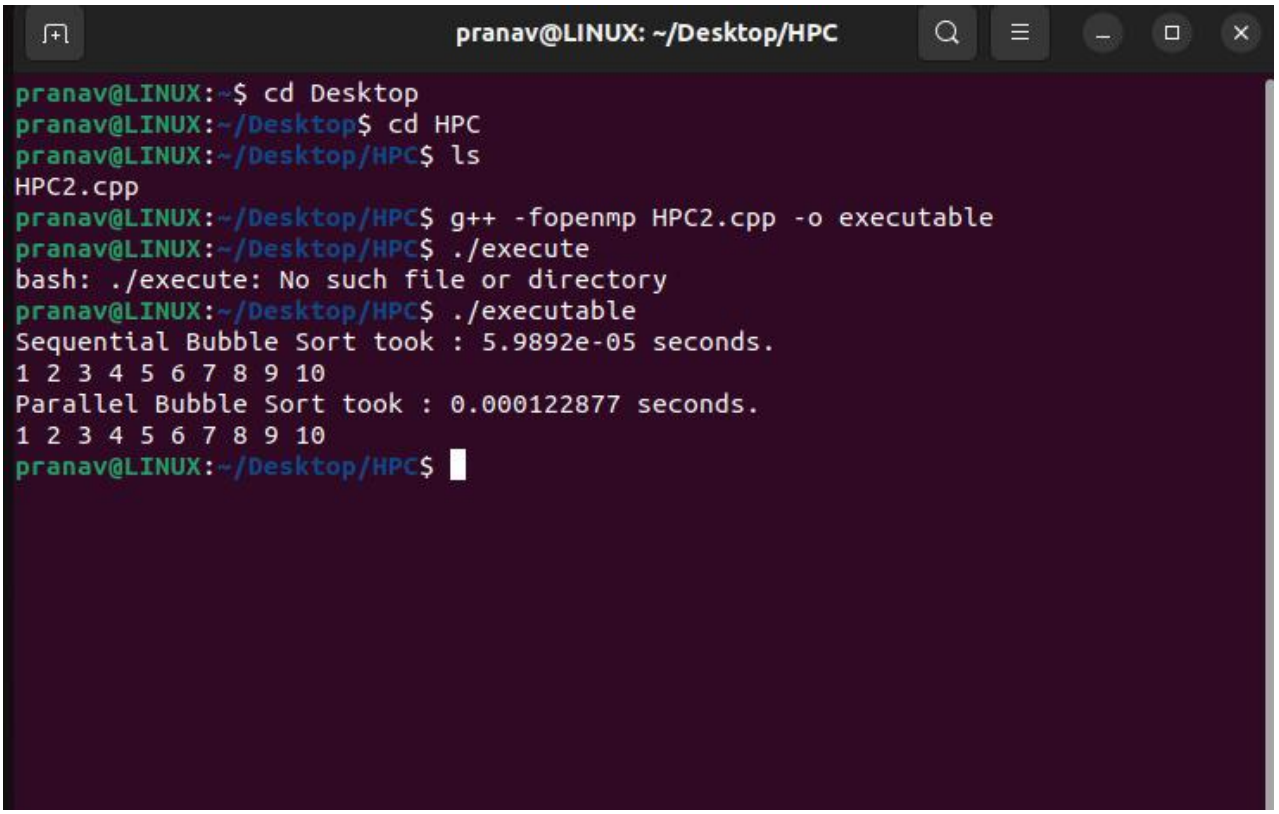
```
      if (array[j] < array[j-1])
      {
        swap(array[j], array[j - 1]);
      }
    }
  }
}
void printArray(int arr[], int n){
    for(int i = 0; i < n; i++) cout << arr[i] << " ";
    cout << "\n";
}
int main(){
    // Set up variables
    int n = 10;
    int arr[n];
    int brr[n];
    double start_time, end_time;

    // Create an array with numbers starting from n to 1
    for(int i = 0, j = n; i < n; i++, j--) arr[i] = j;
    // Sequential time
    start_time = omp_get_wtime();
    bubble(arr, n);
    end_time = omp_get_wtime();
    cout << "Sequential Bubble Sort took : " << end_time - start_time << " seconds.\n";
    printArray(arr, n);

    // Reset the array
    for(int i = 0, j = n; i < n; i++, j--) arr[i] = j;
    // Parallel time
    start_time = omp_get_wtime();
    pBubble(arr, n);
```

```
    end_time = omp_get_wtime();
    cout << "Parallel Bubble Sort took : " << end_time - start_time << " seconds.\n";
    printArray(arr, n);
}
```

Output:



HPC 2 (b)

Input :

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

void merge(int arr[], int low, int mid, int high) {
   // Create arrays of left and right partititons
   int n1 = mid - low + 1;
   int n2 = high - mid;
   int left[n1];
   int right[n2];
   // Copy all left elements
   for (int i = 0; i < n1; i++) left[i] = arr[low + i];
   // Copy all right elements
   for (int j = 0; j < n2; j++) right[j] = arr[mid + 1 + j];
   // Compare and place elements
   int i = 0, j = 0, k = low;
   while (i < n1 && j < n2) {
      if (left[i] <= right[j]){
         arr[k] = left[i];
         i++;
      }
      else{
         arr[k] = right[j];
         j++;
      }
      k++;
   }
```

```
    // If any elements are left out
    while (i < n1) {
        arr[k] = left[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = right[j];
        j++;
        k++;
    }
}

void parallelMergeSort(int arr[], int low, int high) {
    if (low < high) {
        int mid = (low + high) / 2;
        #pragma omp parallel sections
        {
            #pragma omp section
            {
                parallelMergeSort(arr, low, mid);
            }

            #pragma omp section
            {
                parallelMergeSort(arr, mid + 1, high);
            }
        }
        merge(arr, low, mid, high);
    }
```

```
void mergeSort(int arr[], int low, int high) {
    if (low < high) {
        int mid = (low + high) / 2;
        mergeSort(arr, low, mid);
        mergeSort(arr, mid + 1, high);
        merge(arr, low, mid, high);
    }
}
void printArray(int arr[], int n){
    for(int i = 0; i < n; i++) cout << arr[i] << " ";
    cout << "\n";
}

int main() {
    int n = 10;
    int arr[n];
    double start_time, end_time;
    // Create an array with numbers starting from n to 1.
    for(int i = 0, j = n; i < n; i++, j--) arr[i] = j;

    // Measure Sequential Time
    start_time = omp_get_wtime();
    mergeSort(arr, 0, n - 1);
    end_time = omp_get_wtime();
    cout << "Time taken by sequential algorithm: " << end_time - start_time << " seconds\n";

    // Reset the array
    for(int i = 0, j = n; i < n; i++, j--) arr[i] = j;

    //Measure Parallel time
    start_time = omp_get_wtime();
```

```
    parallelMergeSort(arr, 0, n - 1);
    end_time = omp_get_wtime();
    cout << "Time taken by parallel algorithm: " << end_time - start_time << " seconds";

    return 0;
}
```

Output :