Prg1

```
#include <stdio.h>
#include <stdlib.h>
#include<omp.h>

void merge(int arr[], int l,int m, int r)
{
        int i, j, k;
        int n1 = m - l + 1;
        int n2 = r - m;

        int L[n1], R[n2];

        for (i = 0; i < n1; i++)
                L[i] = arr[l + i];
        for (j = 0; j < n2; j++)
                R[j] = arr[m + 1 + j];

        i = 0;
        j = 0;

        k = l;
        while (i < n1 && j < n2)
```

```
{
        if (L[i] <= R[j])
        {
                arr[k] = L[i];
                i++;
        }
        else
        {
                arr[k] = R[j];
                j++;
        }
        k++;
}

while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
}

while (j < n2)
{
        arr[k] = R[j];
        j++;
```

```
            k++;
        }
}

void mergesortParallel(int a[],int l,int h){
   if(l<h){
      int mid = l+(h-l)/2;
      #pragma omp parallel sections
      {
         #pragma omp section
         mergesortParallel(a,l,mid);

         #pragma omp section
         mergesortParallel(a,mid+1,h);
      }
      merge(a,l,mid,h);
   }
}

void mergesortSerial(int a[],int l,int h){
   if(l<h){
      int mid = l+(h-l)/2;

      #pragma omp parallel sections
```

```
        {
            #pragma omp section
            mergesortSerial(a,l,mid);

            #pragma omp section
            mergesortSerial(a,mid+1,h);
        }
        merge(a,l,mid,h);
    }
}

int main(int argc,char *argv[]){
  int *a,num,i;
          num=20;
          a = (int *)malloc(sizeof(int)*num);
          printf("array before sorting\n");
          for(i=0;i<num;i++){
                    a[i]= rand()%100;
          printf("%d ",a[i]);
          }
          double start = omp_get_wtime();
          mergesortSerial(a,0,num-1);
          double end = omp_get_wtime();
  printf("\narray after sorting\n");
```

```
        for(i =0;i<num;i++) printf("%d ",a[i]);
        double val = end - start;
        printf("\nTime for serial is:%f\n",val);
  start = omp_get_wtime();
  mergesortParallel(a,0,num-1);
  end = omp_get_wtime();
  val = end-start;
  printf("Time for parallel execution is %f\n",val);
}
```

Prg 2

```
#include <math.h>
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

#define num_steps 1000000 // infinity assumption

int main(int argc, char *argv[]) {
  double pi = 0;
  // serial
  double start = omp_get_wtime();
  for (int k = 0; k < num_steps; k++) {
    pi += pow(-1, k) / (2 * k + 1);
  }
  pi = 4 * pi;
  double end = omp_get_wtime();
  double time = end - start;
  printf("value of pi in serial : %lf with time : %lf\n", pi, time);

  // parallel
  pi = 0;
  start = omp_get_wtime();
```

```
  int size=omp_get_num_threads();
  double thread[size];
#pragma omp parallel for
  for (int k = 0; k < num_steps; k++) {
    int t = omp_get_thread_num();
    thread[t] += pow(-1, k) / (2 * k + 1);
  }
  for (int i = 0; i < size; i++) {
    pi += thread[i];
  }
  pi = 4 * pi;
  end = omp_get_wtime();
  time = end - start;
  printf("value of pi in parallel : %lf with time : %lf\n", pi, time);
  return 0;
}
```

Prg 3

```
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>

int main(int argc,char *argv[]){
   int iterns,i,itern=1;
   iterns=8;
   #pragma omp parallel for schedule(static,2)
   for(i=1; i<=iterns; i++){
      int t = omp_get_thread_num();
      itern+=1;
      itern%=2;
      printf("thread %d itern %d value: %d\n",t,itern+1,i);
   }
}
```

Prg 4

```
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>
int fib(int n){
   int a=0,b=1,t;
   #pragma omp parallel for  schedule(static,2)
   for(int i=0;i<n;i++){
      #pragma omp critical
      {
         t = a+b;
         a = b;
         b = t;
      }
   }
   return a;
}
int main(int argc,char *argv[]){
   int n = 20;
   double start = omp_get_wtime();
   #pragma omp parallel for
   for(int i=0;i<n;i++){
      int t = omp_get_thread_num();
      printf("thread: %d fib(%d) = %d\n",t,i,fib(i));
```

```
}
double end = omp_get_wtime();
printf("using schedule time is : %f\n",end-start);
```

Prg 5

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int isPrime(int x) {
  for (int i = 2; i <= x / 2; i++) {
    if (x % i == 0) {
      return 0;
    }
  }
  return 1;
}

void parallelprime(int n) {
  int x = 2;
#pragma omp parallel
  while (x <= n) {
    if (isPrime(x)) {
      printf("%d ", x);
    }
#pragma omp atomic
    x++;
  }
```

```
}

void serialprime(int n) {
  int x = 2;
  while (x <= n) {
    if (isPrime(x)) {
      printf("%d ", x);
    }
    x++;
  }
}
int main(int argc, char *argv[]) {
  double start = omp_get_wtime();
  parallelprime(100);
  double end = omp_get_wtime();
  printf("Time for parallel execution is %f\n", end - start);
  start = omp_get_wtime();
  serialprime(100);
  end = omp_get_wtime();
  printf("Time for serial execution is %f\n", end - start);
}
```

Prg6

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

double *vecAdd(double *c,double *a,double *b,int n){
  #pragma omp parallel for
  for(int i=0;i<n;i++)
    c[i]=a[i]+b[i];
}
int main(int argc,char *argv[]) {
  double a[]={1,2,3,4,5};
  double b[]={6,7,8,9,10};
  double c[5];
  vecAdd(c,a,b,5);
  for(int i=0;i<5;i++)
    printf("%lf\n",c[i]);
  return 0;
}
```

Prg 7

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define max 100
int main(int argc, char *argv[]) {
  int i, sum = 0;
#pragma omp parallel for
  for (i = 1; i <= max; i++)
#pragma omp critical
    sum += i;
  printf("Sum : %d\n", sum);
  return 0;
}
```