Reg No. 20BCE1685
Name: Priyanshu Vyas
Course code: CSE4001
Course: Parallel and distributing computing
Title: Week 4 Lab Assignment

Q1: Write an open mp program to sort 1000 numbers.

Algorithm:

(I) store 1000 random elements in an array.
(II) use 2 nested loops to sort the elements.
(III) use #pragma omp parallel for collapse(2).
(IV) Compute and print execution time.
(V) print the sorted array.

```
#include<omp.h>

int main(int argc,char *argv[])
{
  clock_t t;
  int list[1000];
  for(int i=0;i<1000;i++)
  {
    list[i]=rand()%10000;
  }
  t=clock();
  #pragma omp parallel for collapse(2)
  for(int i=0;i<1000;i++)
  {
    for(int j=i+1;j<1000;j++)
    {
      if(list[i]>list[j])
      {
        int temp=list[i];
        list[i]=list[j];
        list[j]=temp;
      }
    }
  }
  t=clock()-t;
  for(int i=0;i<1000;i++)
  {
    printf("%d ",list[i]);
  }
  printf("\n\n Execution Time: %lfms \n\n",(double)t/CLOCKS_PER_SEC);
}
```

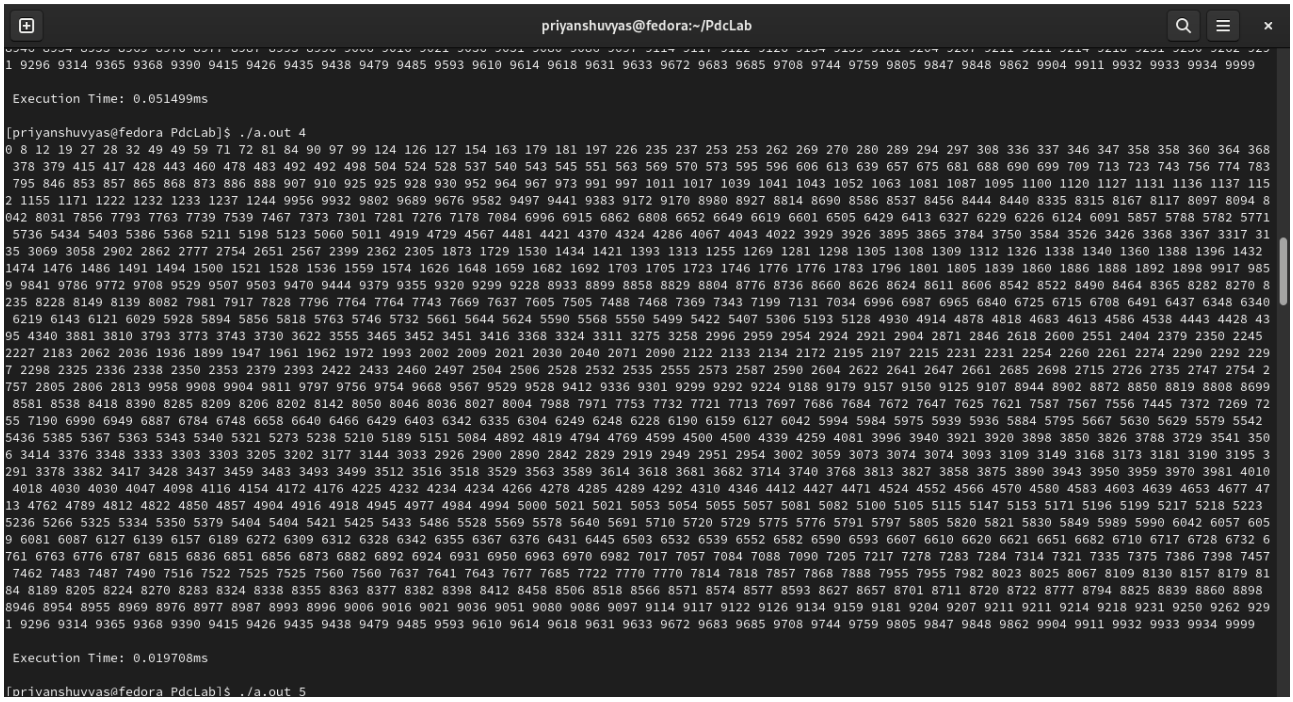
Output:

For Thread 1:

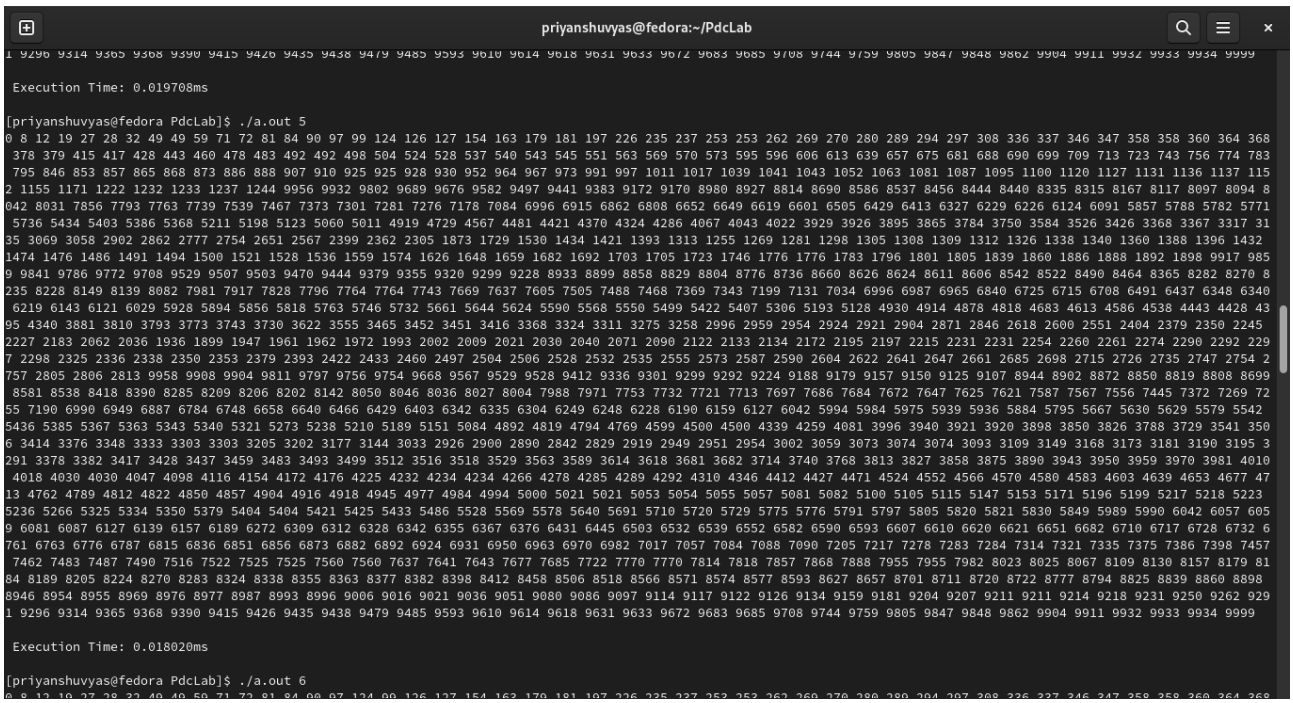
For Thread 2:

For Thread 3:

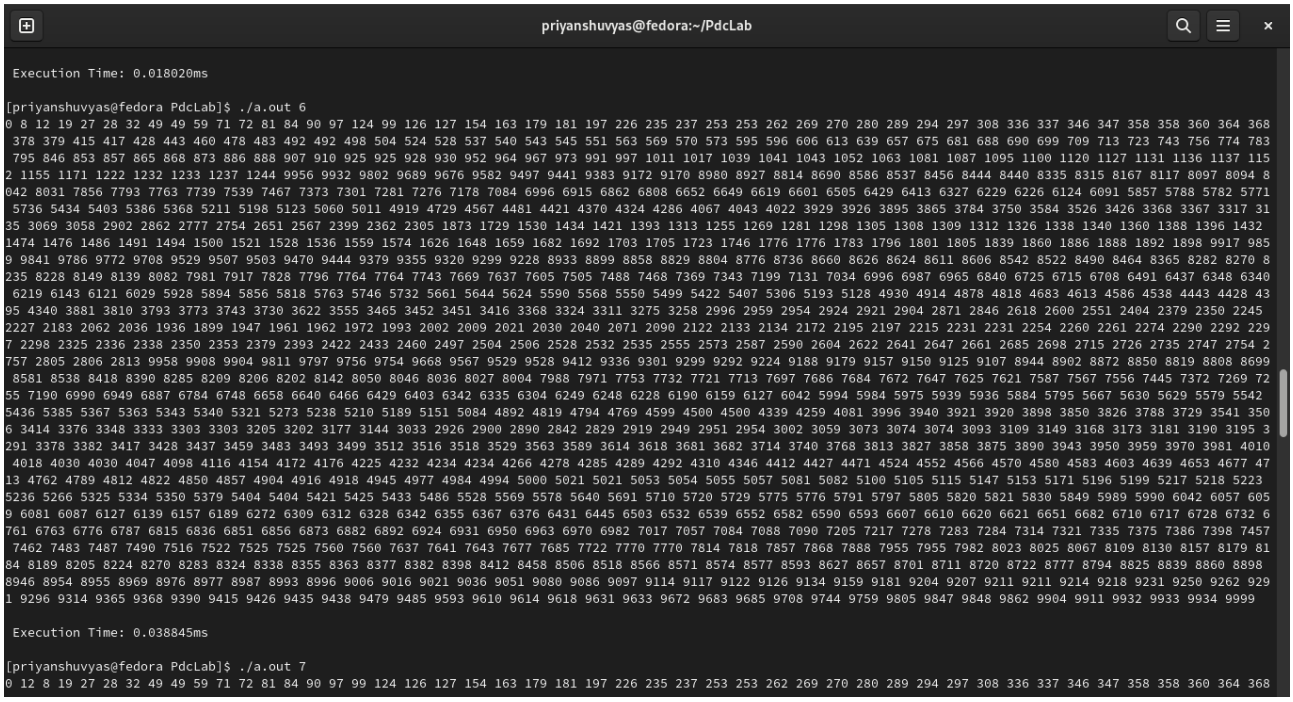
For Thread 4:



For Thread 5:



For Thread 6:



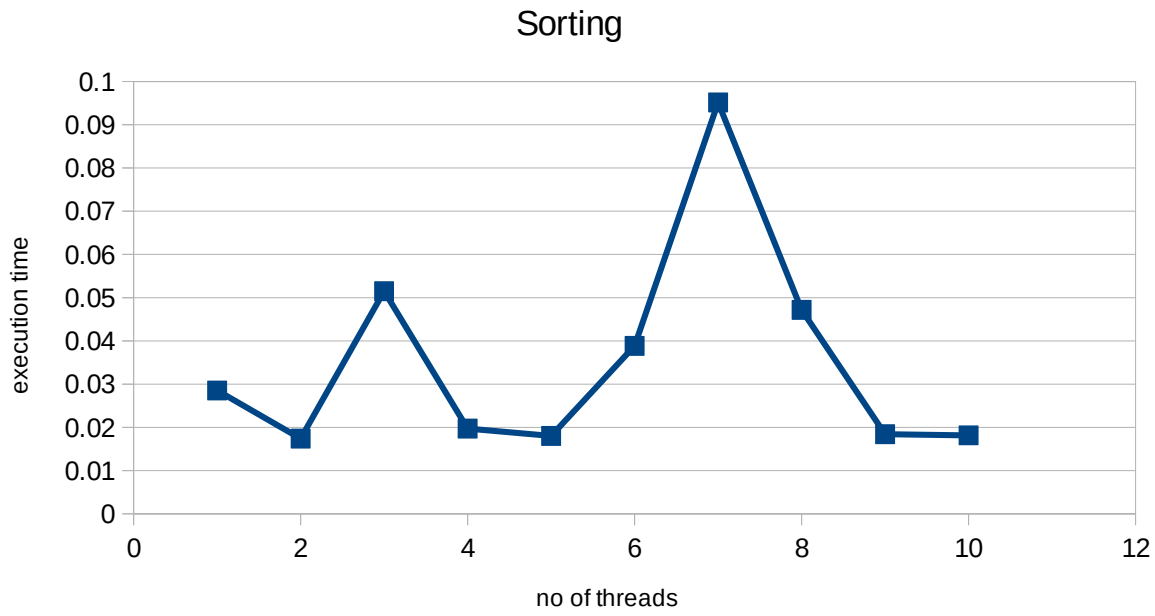
For Thread 7:

For Thread 8:

For Thread 9:

For Thread 10:

Graph:



Result:

Hence all programs are implemented and executed successfully.

Q2: Write an open mp program to find an element in 1000 elements.

Algorithm:

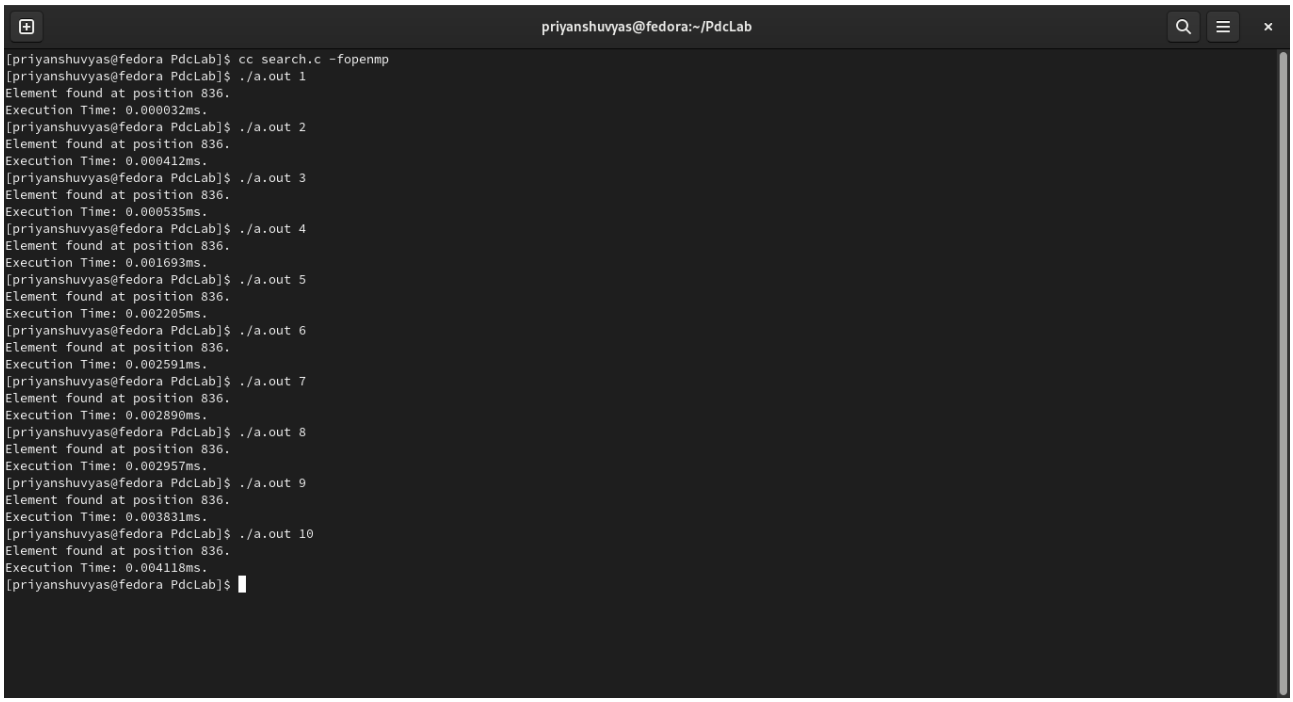
(I) store 1000 random elements in an array.
(II) use a loop to search the element in an array.
(III) use #pragma omp parallel for.
(IV) Compute and print execution time.
(V) print the element index or not found.

Code:

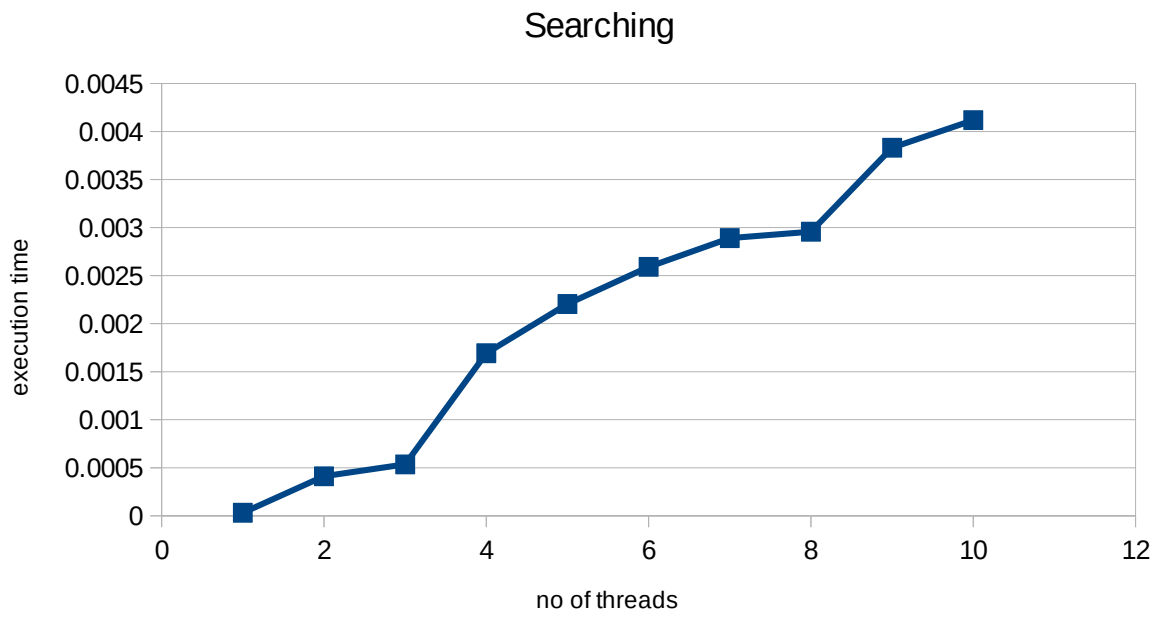
```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<omp.h>

int main(int argc,char *argv[])
{
  omp_set_num_threads(atoi(argv[1]));
  clock_t t;
  int list[1000];
  for(int i=0;i<1000;i++)
  {
    list[i]=rand()%2000;
  }
  t=clock();
  int flag=0;
```

```
 #pragma omp parallel for
 for (int i=0;i<1000;i++)
 {
  if(list[i]==10)
  {
    printf("Element found at position %d. \n",i+1);
    flag=1;
  }
 }
 if(flag==0)
 {
  printf("Element not found in the list. \n");
 }
 t=clock()-t;
 printf("Execution Time: %lfms. \n",(double)t/CLOCKS_PER_SEC);
}
```

Output:



Graph:



Result:

Hence all programs are implemented and executed successfully.

Q3: Write an open mp program to implement:

(i) Master:

Algorithm:

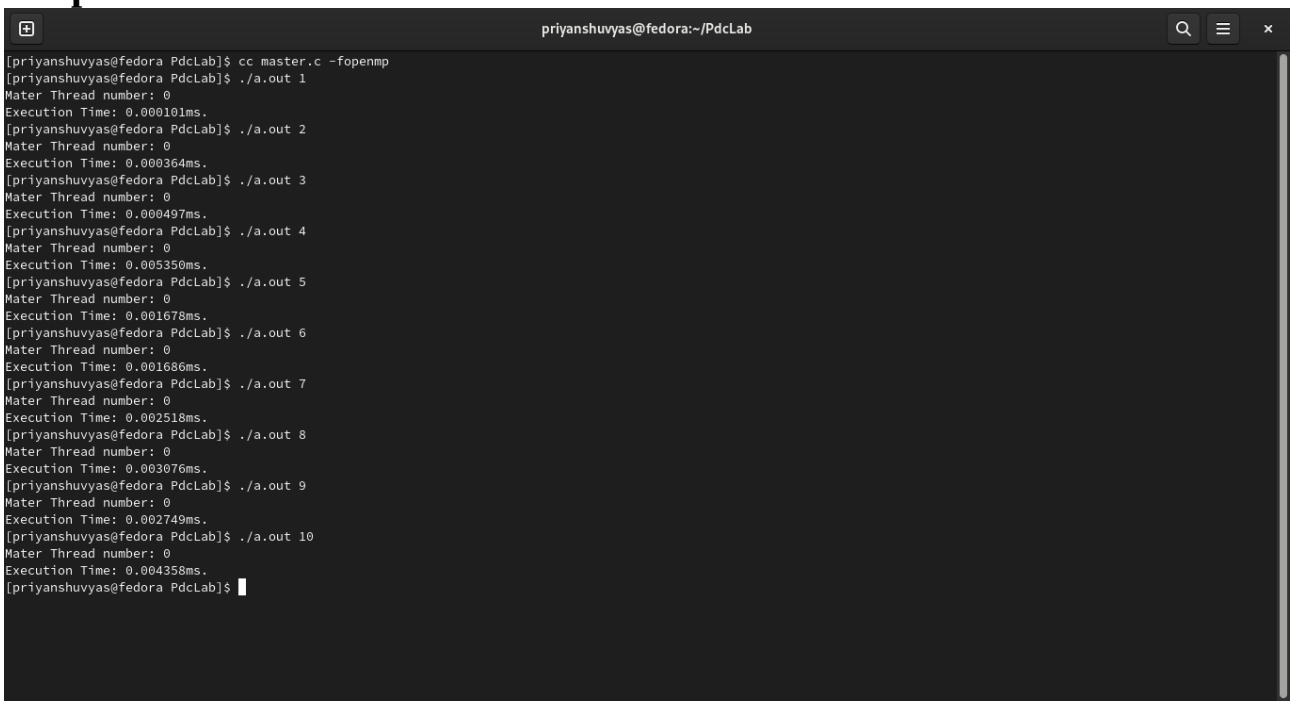
(I) use #pragma omp parallel
(II) use #pragma omp master to print master thread.
(III) Compute and print execution time.

Code:

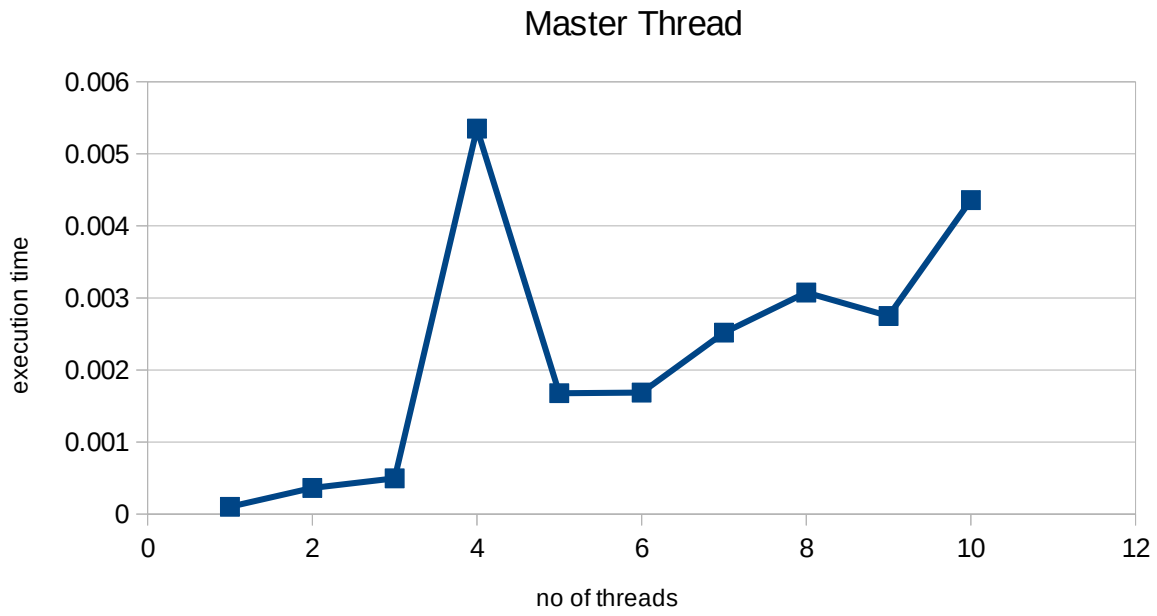
```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<omp.h>

int main(int argc,char *argv[])
{
  clock_t t;
  t=clock();
  omp_set_num_threads(atoi(argv[1]));
  #pragma omp parallel
  {
    #pragma omp master
    {
      printf("Mater Thread number: %d \n",omp_get_thread_num());
    }
  }
  t=clock()-t;
```

```
  printf("Execution Time: %lfms. \n",(double)t/CLOCKS_PER_SEC);
}
```

Output:



Graph:



Result:

Hence all programs are implemented and executed successfully.

(ii) Single:

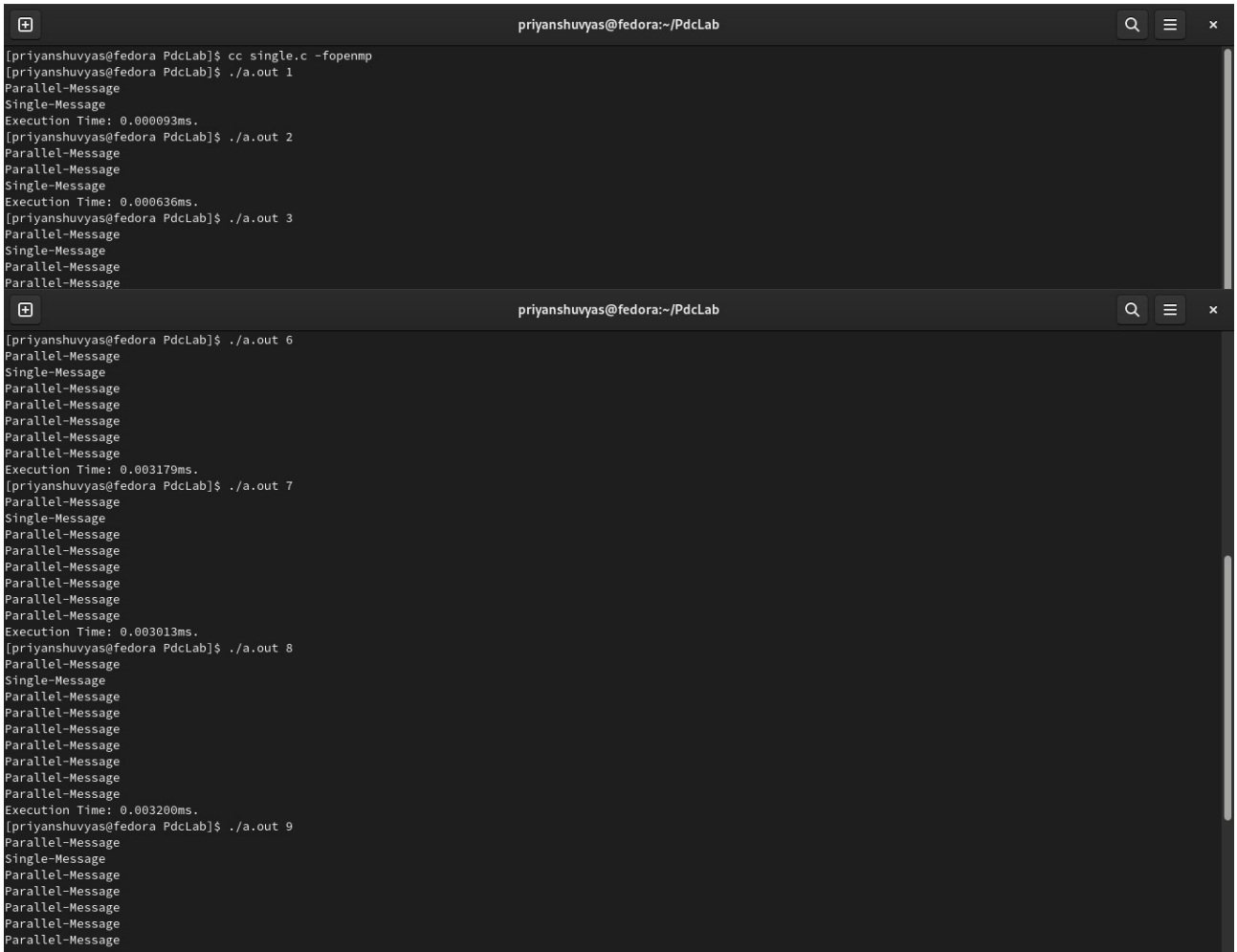
Algorithm:

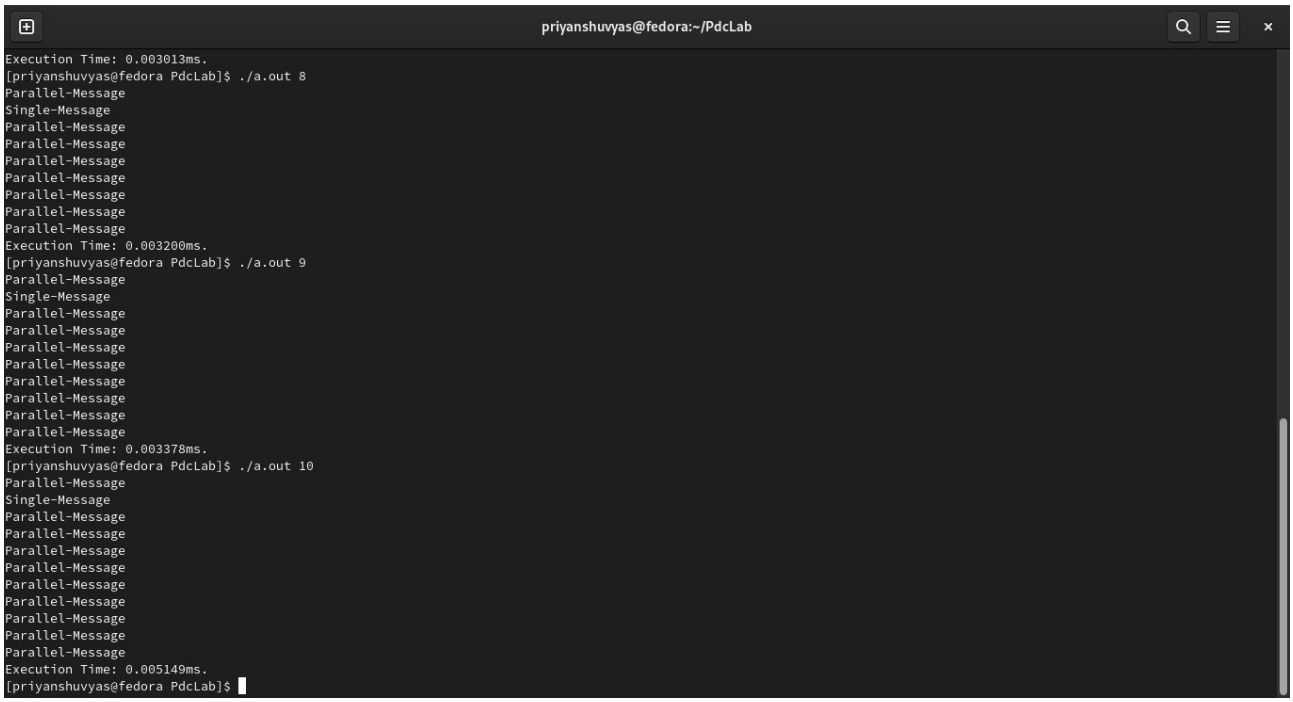
(I) set no of threads.
(II) use #pragma omp parallel.
(III) use #pragma omp single to print a statement single time.
(IV) Compute and print execution time.

Code:

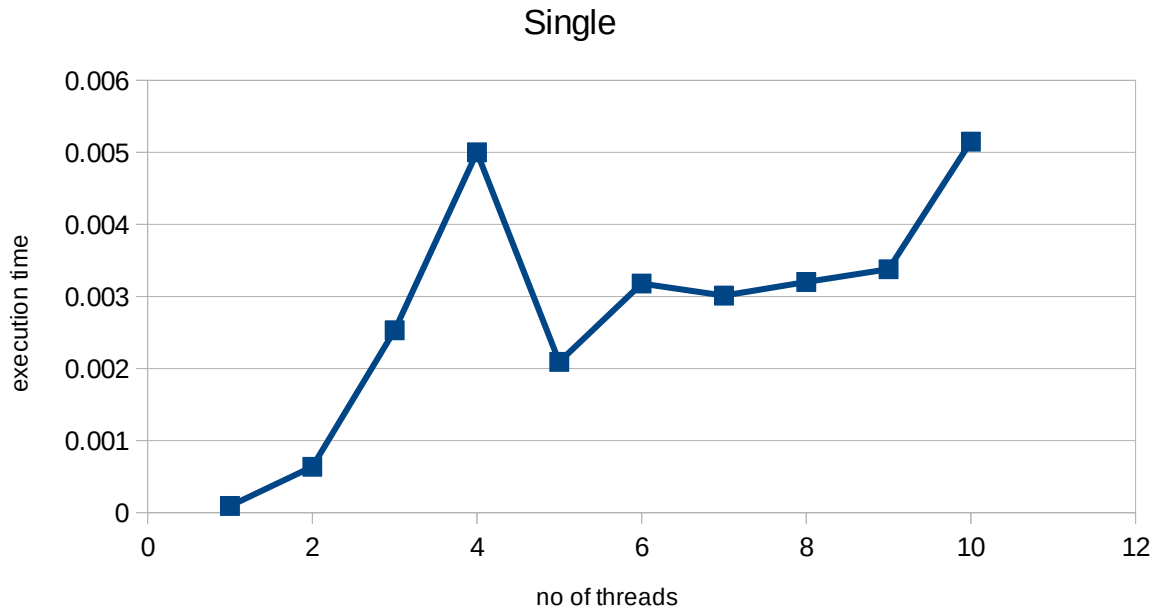
```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<omp.h>

int main(int argc,char* argv[])
{
  omp_set_num_threads(atoi(argv[1]));
  clock_t t;
  t=clock();
  #pragma omp parallel
  {
    printf("Parallel-Message\n");
    #pragma omp single
    {
      printf("Single-Message\n");
    }
  }
  t=clock()-t;
  printf("Execution Time: %lfms. \n",(double)t/CLOCKS_PER_SEC);
}
```

Output:





Graph:



Result:

Hence all programs are implemented and executed successfully.

(iii) firstprivate and lastprivate

Algorithm:

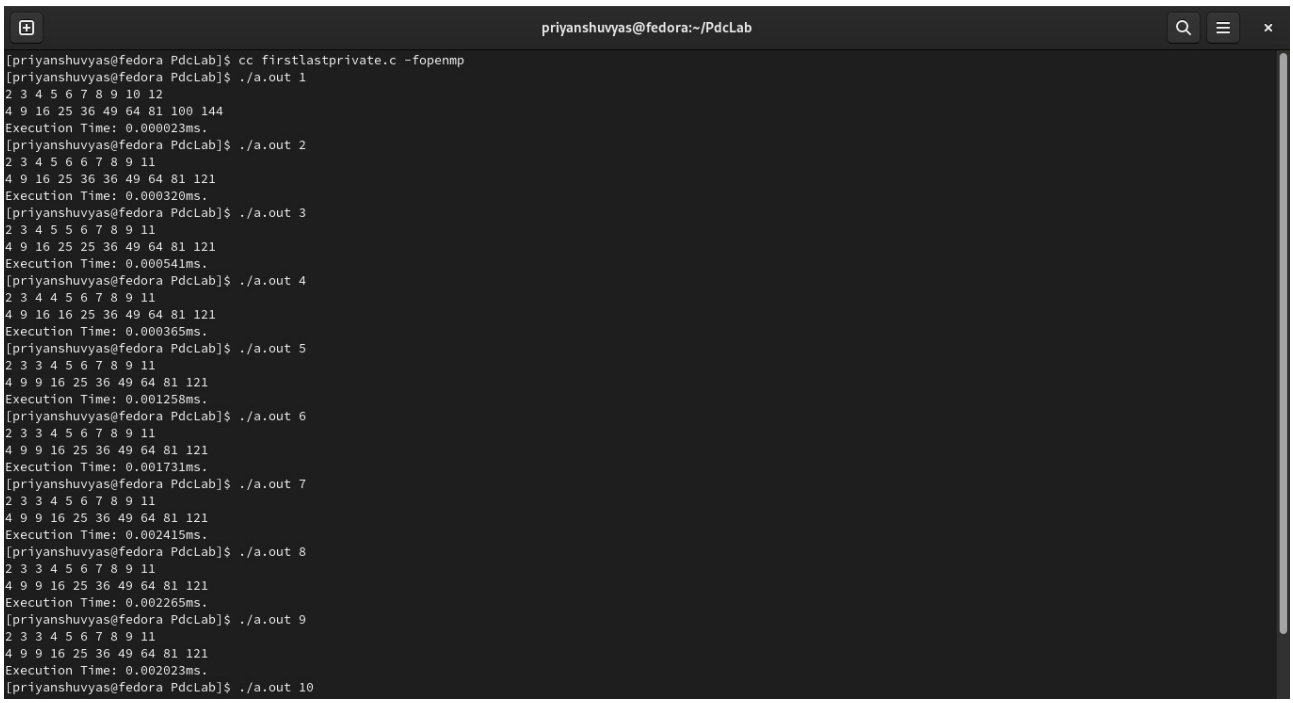
(I) set no of threads.

(II) use #pragma omp parallel for firstprivate() lastprivate().
(III) Compute and print execution time.

Code:

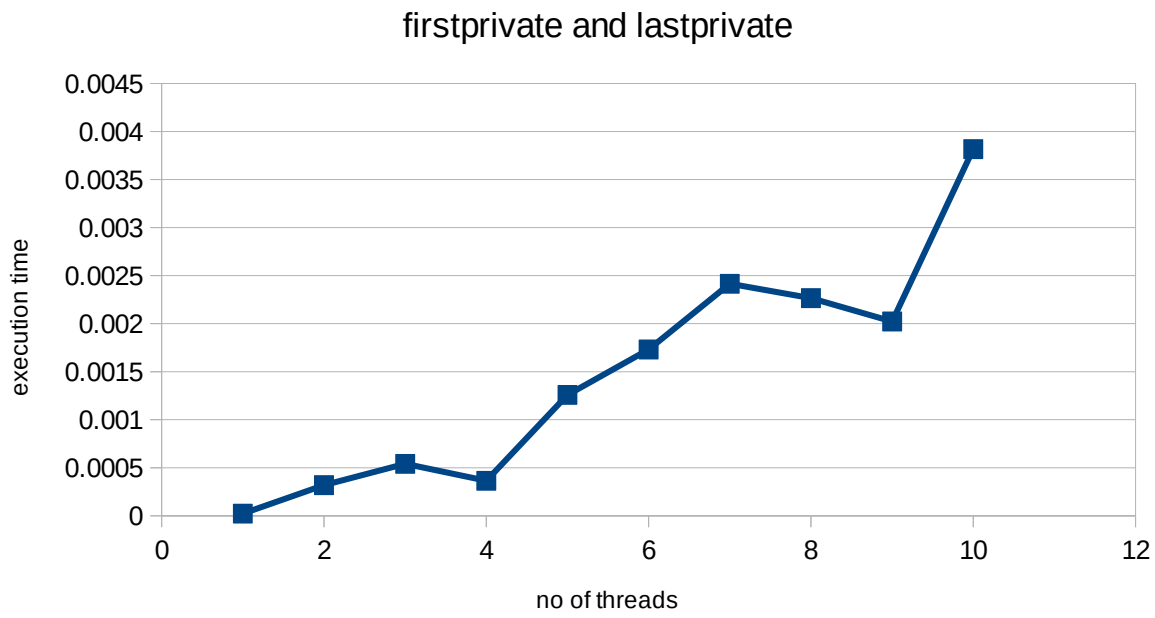
```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<omp.h>

int main(int argc,char* argv[])
{
  omp_set_num_threads(atoi(argv[1]));
  clock_t t;
  t=clock();

  int j=0,x;
  int a[11]={-1,1,2,3,4,5,6,7,8,9,10};
  int b[11];
  #pragma omp parallel for firstprivate(j) lastprivate(x)
  for(int i=1;i<=10;i++)
  {
     if(i==1||i==10)
     {
       j=j+1;
     }
     a[i]=a[i]+j;
     x=a[i];
     b[i]=(x*x);
  }
  t=clock()-t;
  for(int i=1;i<=10;i++)
  {
    printf("%d ",a[i]);
  }
  printf("\n");
  for(int i=1;i<=10;i++)
  {
    printf("%d ",b[i]);
  }
  printf("\nExecution Time: %lfms. \n",(double)t/CLOCKS_PER_SEC);
}
```

Output:


priyanshuvyas@fedora:~/PdcLab
[priyanshuvyas@fedora PdcLab]$ cc firstlastprivate.c -fopenmp
[priyanshuvyas@fedora PdcLab]$ ./a.out 1
2 3 4 5 6 7 8 9 10 12
4 9 16 25 36 49 64 81 100 144
Execution Time: 0.000023ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 2
2 3 4 5 6 6 7 8 9 11
4 9 16 25 36 36 49 64 81 121
Execution Time: 0.000320ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 3
2 3 4 5 5 6 7 8 9 11
4 9 16 25 25 36 49 64 81 121
Execution Time: 0.000541ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 4
2 3 4 4 5 6 7 8 9 11
4 9 16 16 25 36 49 64 81 121
Execution Time: 0.000365ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 5
2 3 3 4 5 6 7 8 9 11
4 9 9 16 25 36 49 64 81 121
Execution Time: 0.001258ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 6
2 3 3 4 5 6 7 8 9 11
4 9 9 16 25 36 49 64 81 121
Execution Time: 0.001731ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 7
2 3 3 4 5 6 7 8 9 11
4 9 9 16 25 36 49 64 81 121
Execution Time: 0.002415ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 8
2 3 3 4 5 6 7 8 9 11
4 9 9 16 25 36 49 64 81 121
Execution Time: 0.002265ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 9
2 3 3 4 5 6 7 8 9 11
4 9 9 16 25 36 49 64 81 121
Execution Time: 0.002023ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 10


priyanshuvyas@fedora:~/PdcLab
Execution Time: 0.000023ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 2
2 3 4 5 6 6 7 8 9 11
4 9 16 25 36 36 49 64 81 121
Execution Time: 0.000320ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 3
2 3 4 5 5 6 7 8 9 11
4 9 16 25 25 36 49 64 81 121
Execution Time: 0.000541ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 4
2 3 4 4 5 6 7 8 9 11
4 9 16 16 25 36 49 64 81 121
Execution Time: 0.000365ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 5
2 3 3 4 5 6 7 8 9 11
4 9 9 16 25 36 49 64 81 121
Execution Time: 0.001258ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 6
2 3 3 4 5 6 7 8 9 11
4 9 9 16 25 36 49 64 81 121
Execution Time: 0.001731ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 7
2 3 3 4 5 6 7 8 9 11
4 9 9 16 25 36 49 64 81 121
Execution Time: 0.002415ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 8
2 3 3 4 5 6 7 8 9 11
4 9 9 16 25 36 49 64 81 121
Execution Time: 0.002265ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 9
2 3 3 4 5 6 7 8 9 11
4 9 9 16 25 36 49 64 81 121
Execution Time: 0.002023ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 10
2 2 3 4 5 6 7 8 9 11
4 4 9 16 25 36 49 64 81 121
Execution Time: 0.003816ms.
[priyanshuvyas@fedora PdcLab]$

Graph:



Result:

Hence all programs are implemented and executed successfully.

(iv) Ordered:

Algorithm:

(I) set no of threads.
(II) use #pragma omp parallel for ordered.
(III) use #pragma omp ordered to print a statement in order.
(IV) Compute and print execution time.

Code:

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

int main(int argc,char* argv[])
{
  omp_set_num_threads(atoi(argv[1]));
  clock_t t;
  t=clock();
  #pragma omp parallel for ordered
  for (int i=1;i<=10;i++)
  {
    #pragma omp ordered
    {
      printf("%d ",i);
    }
  }
```

```
  t=clock()-t;
  printf("\nExecution Time: %lfms. \n",(double)t/CLOCKS_PER_SEC);
}
```

Output:

```
priyanshuvyas@fedora:~/PdcLab
[priyanshuvyas@fedora PdcLab]$ cc ordered.c -fopenmp
[priyanshuvyas@fedora PdcLab]$ ./a.out 1
1 2 3 4 5 6 7 8 9 10
Execution Time: 0.000108ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 2
1 2 3 4 5 6 7 8 9 10
Execution Time: 0.000384ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 3
1 2 3 4 5 6 7 8 9 10
Execution Time: 0.000472ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 4
1 2 3 4 5 6 7 8 9 10
Execution Time: 0.000618ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 5
1 2 3 4 5 6 7 8 9 10
Execution Time: 0.002175ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 6
1 2 3 4 5 6 7 8 9 10
Execution Time: 0.001894ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 7
1 2 3 4 5 6 7 8 9 10
Execution Time: 0.003139ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 8
1 2 3 4 5 6 7 8 9 10
Execution Time: 0.003027ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 9
1 2 3 4 5 6 7 8 9 10
Execution Time: 0.003858ms.
[priyanshuvyas@fedora PdcLab]$ ./a.out 10
1 2 3 4 5 6 7 8 9 10
Execution Time: 0.002155ms.
[priyanshuvyas@fedora PdcLab]$
[priyanshuvyas@fedora PdcLab]$
[priyanshuvyas@fedora PdcLab]$
[priyanshuvyas@fedora PdcLab]$
[priyanshuvyas@fedora PdcLab]$
[priyanshuvyas@fedora PdcLab]$
[priyanshuvyas@fedora PdcLab]$
```

Graph:

Result:

Hence all programs are implemented and executed successfully.