CSE4001-PARALLEL AND DISTRIBUTED COMPUTING
LAB-4

Name: O G Ragavi
Reg No: 20BCE1988
Lab: 4

—----------

1.Aim : Sorting an array of 1000 numbers

Algorithm:

- Include the preprocessor directives
- Use the clock function of time.h library to start the timer before entering into the parallel section
- Use a loop that runs from 1 to size of the array which is 1000
- Use the phase #pragma omp parallel for, so that the iterations of the inner loop gets distributed among the given number of threads to reach maximum efficiency .

Code:

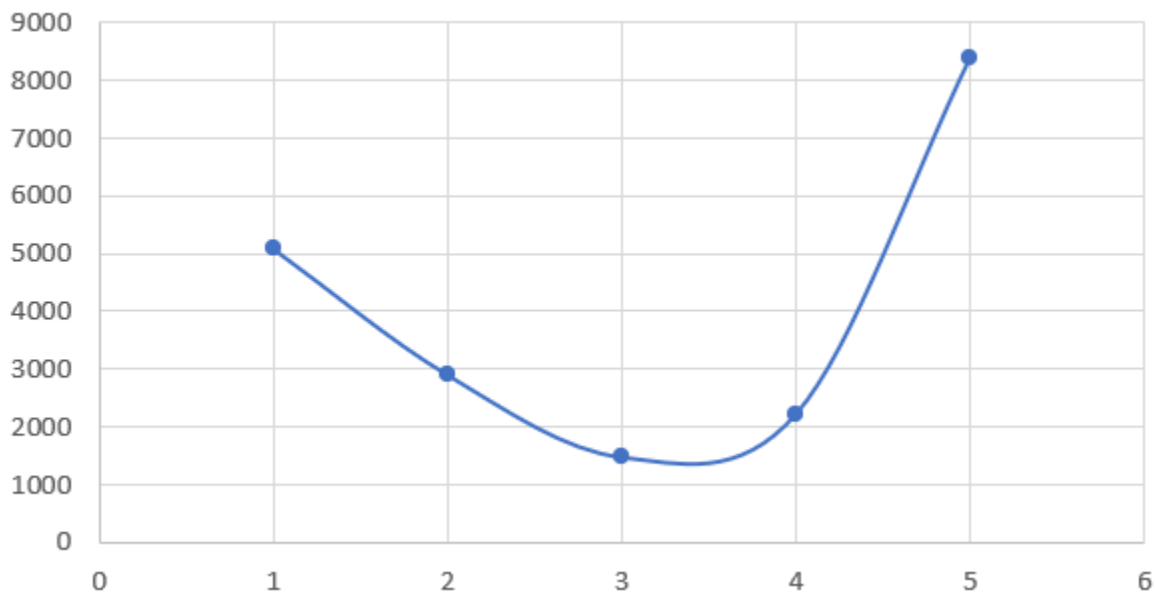
```
#include<stdio.h>
#include<omp.h>
#include<time.h>
#include<stdlib.h>

int main () {

int A[1000];
int N = 1000;
int i=0;
for(int i=0;i<N;i++)
{
 A[i]=rand()%50;
}
clock_t s,e;

for(int k=0;k<5;k++)
{
  s=clock();
  omp_set_num_threads(k+1);
  #pragma omp parallel for default(none), shared(A, N)
```

```
        for(int i=0;i<N;i++)
        {



          for(int j = 0; j < N-1; j++)
          {
if(A[j] > A[j + 1])
{
   int temp = A[j];
   A[j] = A[j + 1];
   A[j + 1] = temp;
}
    }
         }


/* for(i=0;i<N;i++)
 {
  //printf("%d ",A[i]);
 }*/
        e = clock();
        printf("\nTime taken for %d threads is: %ld\n",k+1,(e-s));
        }

}
```

Output:

```
Time taken for 1 threads is: 5075

Time taken for 2 threads is: 2895

Time taken for 3 threads is: 1475

Time taken for 4 threads is: 2223

Time taken for 5 threads is: 8391
```

Graph:



2.Aim : Searching an element in an array of elements:

Algorithm:

- Include the required pre-processor directives
- Declare an array of the required size and initialize it with random numbers
- Use the time function to start the clock before the parallel section
- User #pragma omp parallel for, so that the iterations gets distributed among the given number of threads
- Inside the loop use a binary search to search the key from the corresponding array, print the location of the element found.

Code:

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

int bin(int a[], int l, int r, int key, int n)
{
    int index = -1;
    int size = (r - l + 1) / n;
    if (size == 0 || n == 1)
    {
        #pragma omp parallel for
        for (int i = l; i <= r; i++)
        {
            if (a[i] == key)
                index = i;
        }
        return index;
    }
    int left = l;
    int right = r;
    omp_set_num_threads(n);
    omp_set_nested(1);
    #pragma omp parallel
    {
        int id = omp_get_thread_num();
        int lt = l + id * size;
        int rt = lt + size - 1;
        if (id == n - 1)
            rt = r;

        if (a[lt] <= key && a[rt] >= key)
        {
            left = lt;
            right = rt;
```

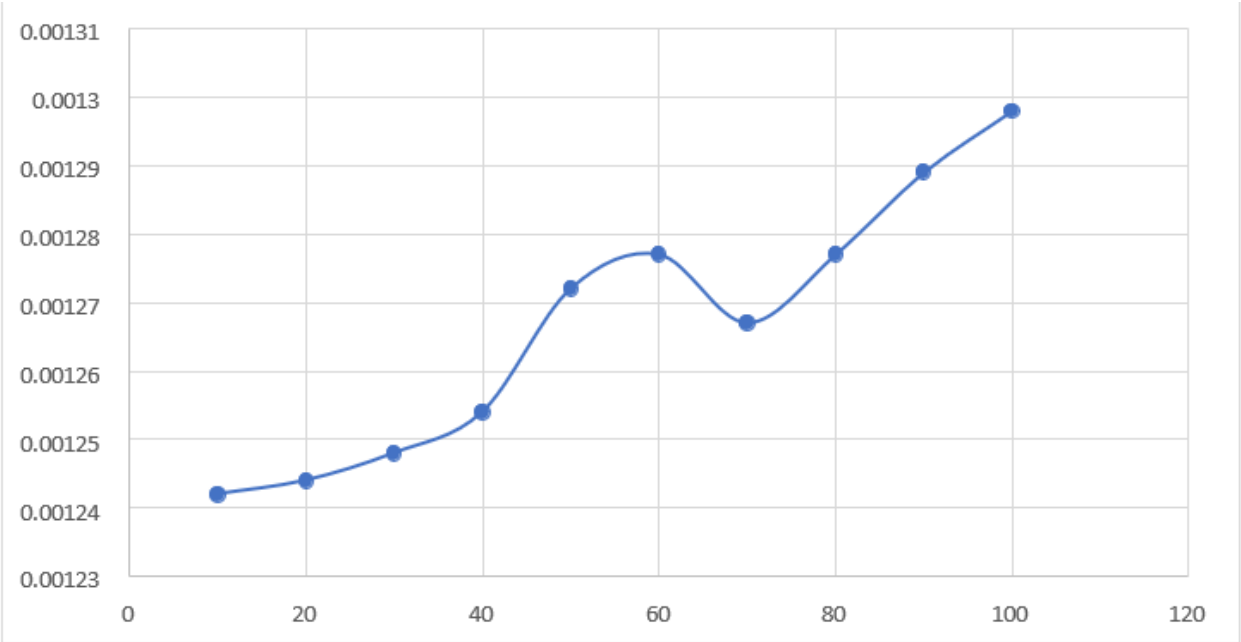
```
        }
    }
    if (left == l && right == r)
        return -1;
    return bin(a, left, right, key, n);
}

int main()
{
    int n;
    int nthread;
    clock_t s, e;
    printf("Enter the size of the array: ");
    scanf("%d", &n);
    int a[n];

    for (int i = 0; i < n; i++)
        a[i] = rand()%50;
        printf("Enter the element to be searched: ");
    int key;
    scanf("%d", &key);
    for(int i=0;i<5;i++){


    s = clock();
    printf("Position of the elemnt:%d\n", bin(a, 0, n- 1, key, i+1));
    e = clock();
    printf("Time taken  for %d threads is : %ld\n",i+1,(e-s));
    }

}
```

Output:

```
Enter the size of the array: 100
Enter the element to be searched: 10
Position of the elemnt:91
Time taken  for 1 threads is : 347
Position of the elemnt:-1
Time taken  for 2 threads is : 11
Position of the elemnt:-1
Terminal n  for 3 threads is : 535
Position of the elemnt:-1
Time taken  for 4 threads is : 1224
Position of the elemnt:-1
Time taken  for 5 threads is : 575
```


```
Enter the size of the array: 100
Enter the element to be searched: 10
Position of the elemnt:91
Time taken  for 1 threads is : 347
Position of the elemnt:-1
Time taken  for 2 threads is : 11
Position of the elemnt:-1
Time taken  for 3 threads is : 535
Position of the elemnt:-1
Time taken  for 4 threads is : 1224
Position of the elemnt:-1
Time taken  for 5 threads is : 575
```


Graph:



3.Aim: To write an OpenMP program to illustrate master, single, firstprivate, lastprivate and ordered .

a. Master

Code:

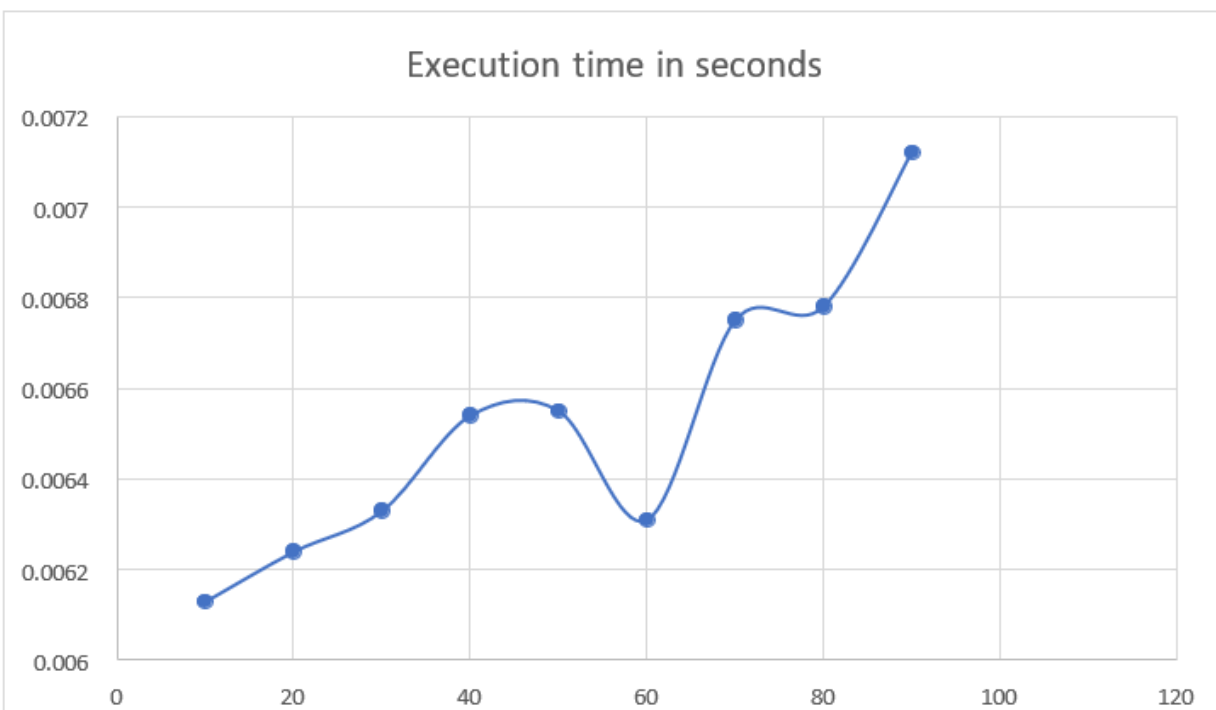
```
#include<stdio.h>
#include<omp.h>
#include<stdlib.h>
 #include<time.h>
void main()
{
   time_t start,end;
   start = clock();
   #pragma omp parallel
   {
      printf("The block of code is executed by thread
%d\n",omp_get_thread_num());
       #pragma omp master
      {
          printf("This is the master block and exeucted by thread
%d\n",omp_get_thread_num());
      }
   }
   end = clock();
```

```
  double diff = (end-start)*1.0/CLOCKS_PER_SEC;
  printf("The execution time is %lf seconds\n",diff);
}
```

Output:

```
The block of code is executed by thread 0
This is the master block and exeucted by thread 0
The block of code is executed by thread 3
The block of code is executed by thread 4
The block of code is executed by thread 5
The block of code is executed by thread 6
The block of code is executed by thread 7
The block of code is executed by thread 8
The block of code is executed by thread 9
The block of code is executed by thread 2
The block of code is executed by thread 1
The execution time is 0.000613 seconds
```

Graph:



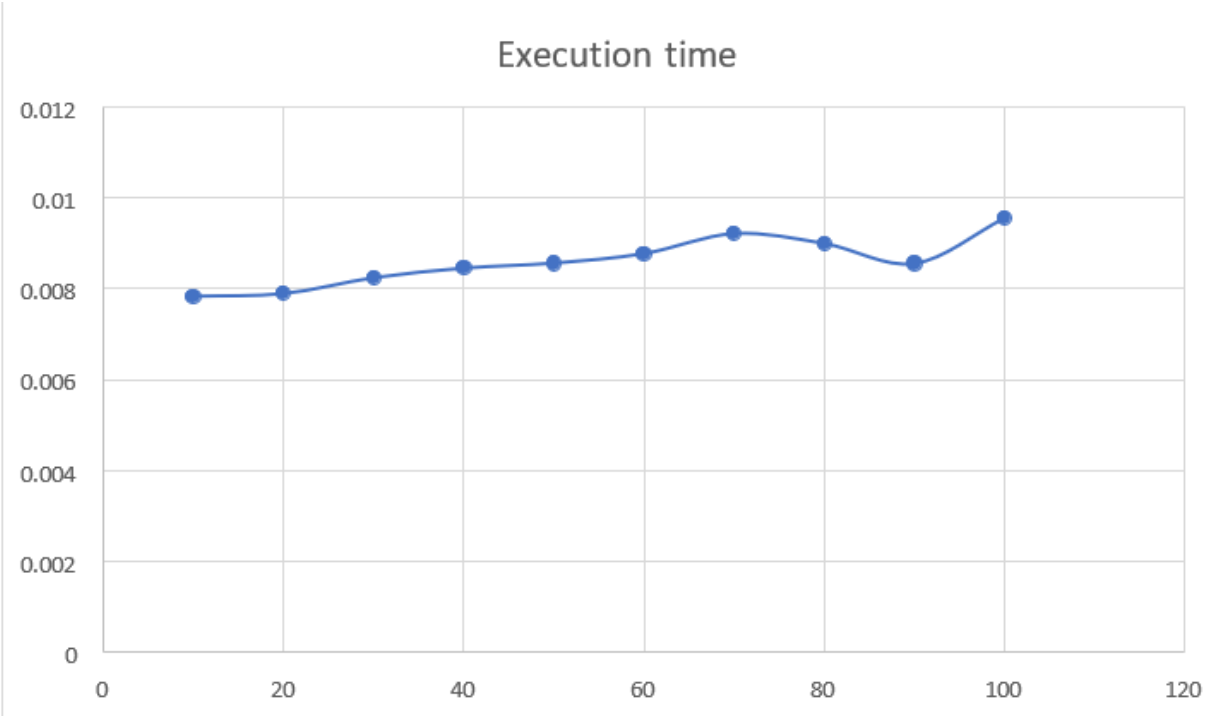
b.Single:

Code:

```
#include<stdio.h>
#include<omp.h>
#include<stdlib.h>
#include<time.h>
 void main()
{     time_t start,end;
    start = clock();
    #pragma omp parallel
    {
       printf("The block of code is executed by thread
%d\n",omp_get_thread_num());        #pragma omp single
       {
           printf("This is the single block and exeucted by thread
%d\n",omp_get_thread_num());
       }
    }
    end = clock();
    double diff = (end-start)*1.0/CLOCKS_PER_SEC;     printf("The execution time
is %lf seconds\n",diff);
}
```

Output:

```
The block of code is executed by thread 9
This is the single block and exeucted by thread 9
The block of code is executed by thread 1
The block of code is executed by thread 2
The block of code is executed by thread 3
The block of code is executed by thread 4
The block of code is executed by thread 5
The block of code is executed by thread 6
The block of code is executed by thread 7
The block of code is executed by thread 8
The block of code is executed by thread 0
The execution time is 0.000783 seconds
```



c.Firstprivate:

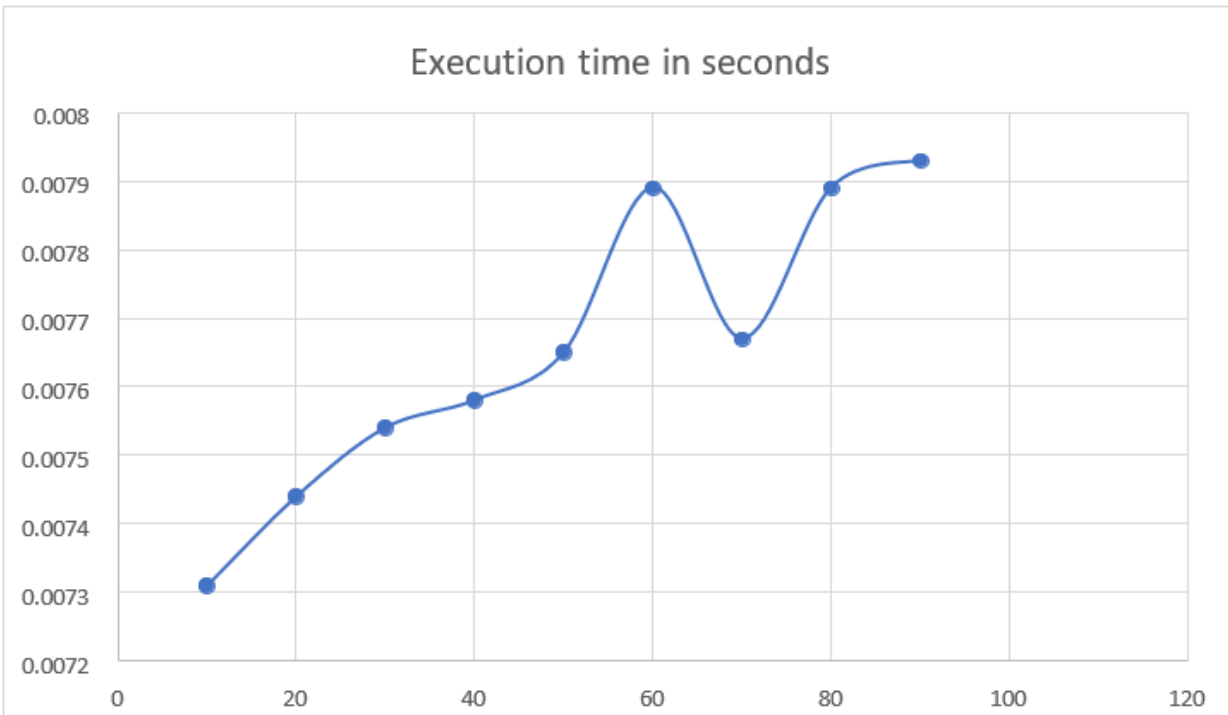
Code:

```
#include<stdio.h>
#include<omp.h>
#include<stdlib.h>
 #include<time.h>
 void main()
{
    time_t start,end;
    int i = 127;
    start = clock();
    #pragma omp parallel firstprivate(i)
    {
       printf("The value of i in the thread %d is %d\n",omp_get_thread_num(),i);
    }
    end = clock();
    double diff = (end-start)*1.0/CLOCKS_PER_SEC;
 printf("The execution time is %lf seconds\n",diff);
 }
```

Output:

```
The value of i in the thread 9 is 127
The value of i in the thread 0 is 127
The value of i in the thread 1 is 127
The value of i in the thread 2 is 127
The value of i in the thread 3 is 127
The value of i in the thread 4 is 127
The value of i in the thread 5 is 127
The value of i in the thread 6 is 127
The value of i in the thread 7 is 127
The value of i in the thread 8 is 127
The execution time is 0.000731 seconds
```

Graph:



d.Lastprivate:

Code:

```
#include<stdio.h>
#include<omp.h>
#include<stdlib.h>
#include<time.h>
 void main()
{
     time_t start,end;
```

```
    start = clock();
    int j = 0;
   #pragma omp parallel for lastprivate(j)
      for(int i=0;i<10;i++)
      {
      printf("The value of j in the thread %d is %d\n",omp_get_thread_num(),j);
       j++;
      }
   printf("The value of j outside the parallel region is %d\n",j);
  end = clock();
   double diff = (end-start)*1.0/CLOCKS_PER_SEC;
 printf("The execution time is %lf seconds\n",diff);
}
```

Output:

```
The value of j in the thread 3 is 0
The value of j in the thread 3 is 1
The value of j in the thread 1 is 0
The value of j in the thread 1 is 1
The value of j in the thread 1 is 2
The value of j in the thread 2 is 0
The value of j in the thread 2 is 1
The value of j in the thread 0 is 0
The value of j in the thread 0 is 1
The value of j in the thread 0 is 2
The value of j outside the parallel region is 2
The execution time is 0.000419 seconds
```

Graph:

D.ordered:

Code:

```
#include<stdio.h>
#include<omp.h>
#include<stdlib.h>
 #include<time.h>
 void main()
{
    time_t start,end;
    start = clock();
    int j = 0;
    #pragma omp parallel
    {
        int x = 0;
        #pragma omp for ordered
        for(int i=0;i<10;i++)
        {
            x = x+i;
            #pragma omp ordered
            printf("in i=%d x=%d thread=%d\n",i,x,omp_get_thread_num());
        }
```

```
    }
    end = clock();
    double diff = (end-start)*1.0/CLOCKS_PER_SEC;
   printf("The execution time is %lf seconds\n",diff);
}
```

Output:

```
in i=0 x=0 thread=0
in i=1 x=1 thread=1
in i=2 x=2 thread=2
in i=3 x=3 thread=3
in i=4 x=4 thread=4
in i=5 x=5 thread=5
in i=6 x=6 thread=6
in i=7 x=7 thread=7
in i=8 x=8 thread=8
in i=9 x=9 thread=9
The execution time is 0.000721 seconds
```

Graph: