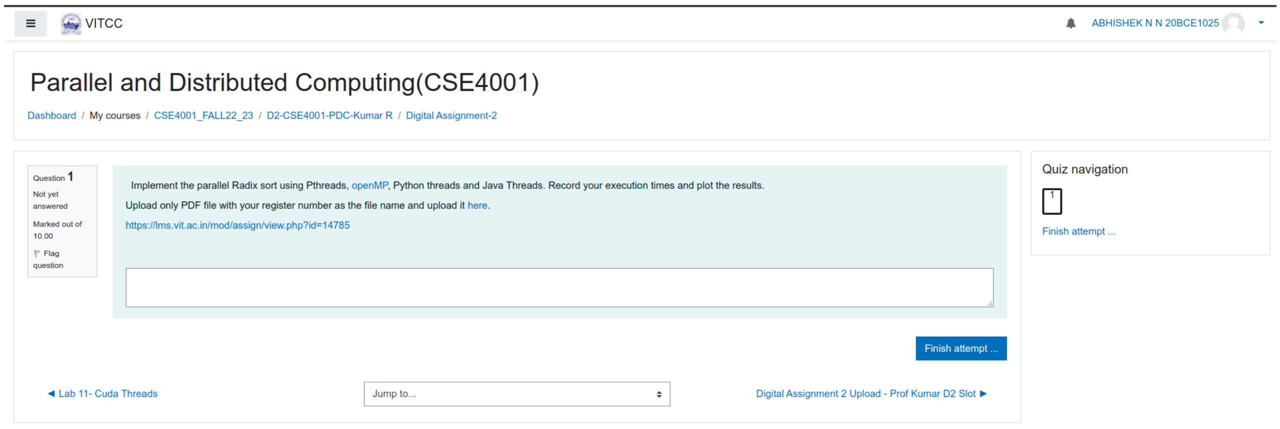
VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Digital Assignment 2

Programme	:	**BTech. CSE Core**	Semester	:	**Fall 22-23**
Course	:	**Parallel and Distributed Computing**	Code	:	**CSE4001**
Faculty	:	**Dr. Kumar R**	Slot	:	**D2**
Name	:	**Abhishek N N**	Register No.	:	**20BCE1025**



Parallel and Distributed Computing(CSE4001)

Dashboard / My courses / CSE4001_FALL22_23 / D2-CSE4001-PDC-Kumar R / Digital Assignment-2

Question 1
Not yet answered
Marked out of 10.00

Implement the parallel Radix sort using Pthreads, openMP, Python threads and Java Threads. Record your execution times and plot the results.

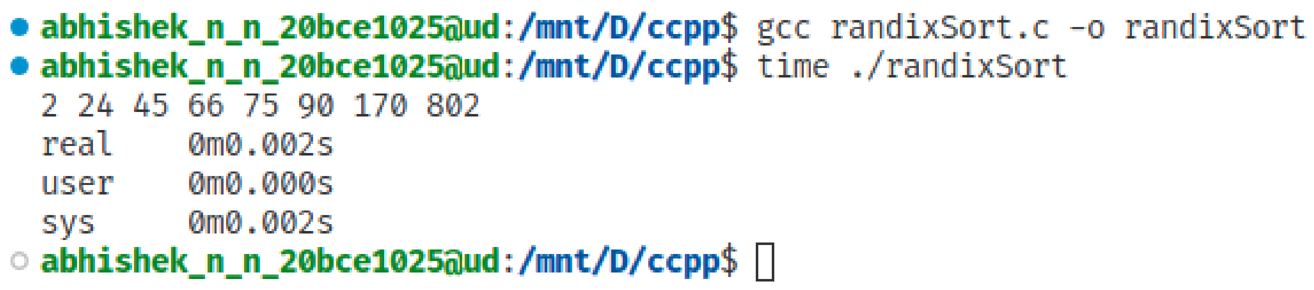
Upload only PDF file with your register number as the file name and upload it here.

https://lms.vit.ac.in/mod/assign/view.php?id=14785

C Serial

```c
// C implementation of Radix Sort

#include <stdio.h>

// A utility function to get maximum value in arr[]
int getMax(int arr[], int n) {
     int mx = arr[0];
     for (int i = 1; i < n; i++)
          if (arr[i] > mx)
               mx = arr[i];
     return mx;
}
// A utility function to print an array
void print(int arr[], int n){
     for (int i = 0; i < n; i++)
          printf("%d ",arr[i]);
}

// A function to do counting sort of arr[] according to
// the digit represented by exp.
void countSort(int arr[], int n, int exp){
     int output[n]; // output array
     int i, count[10] = { 0 };

     // Store count of occurrences in count[]
     for (i = 0; i < n; i++)
          count[(arr[i] / exp) % 10]++;

     // Change count[i] so that count[i] now contains actual
     // position of this digit in output[]
     for (i = 1; i < 10; i++)
          count[i] += count[i - 1];

     // Build the output array
     for (i = n - 1; i >= 0; i--) {
          output[count[(arr[i] / exp) % 10] - 1] = arr[i];
          count[(arr[i] / exp) % 10]--;
     }

     // Copy the output array to arr[], so that arr[] now
     // contains sorted numbers according to current digit
     for (i = 0; i < n; i++)
          arr[i] = output[i];
}
```

```c
// The main function to that sorts arr[] of size n using
// Radix Sort
void radixsort(int arr[], int n){
     // Find the maximum number to know number of digits
     int m = getMax(arr, n);

     // Do counting sort for every digit. Note that instead
     // of passing digit number, exp is passed. exp is 10^i
     // where i is current digit number
     for (int exp = 1; m / exp > 0; exp *= 10)
          countSort(arr, n, exp);
}
// Driver Code
int main(){
     int arr[] = { 170, 45, 75, 90, 802, 24, 2, 66 };
     int n = sizeof(arr) / sizeof(arr[0]);
     // Function Call
     radixsort(arr, n);
     print(arr, n);
     return 0;
}
```

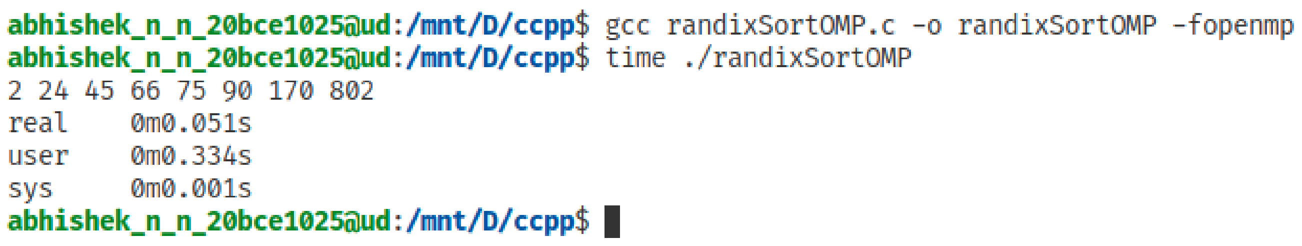


```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ gcc randixSort.c -o randixSort
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ time ./randixSort
2 24 45 66 75 90 170 802
real    0m0.002s
user    0m0.000s
sys     0m0.002s
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ 
```

C OpenMp

```c
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>
#define BASE_BITS 8
#define BASE (1 << BASE_BITS)
#define MASK (BASE-1)
#define DIGITS(v, shift) (((v) >> shift) & MASK)
// A utility function to print an array
void print(int arr[], int n) {
     for (int i = 0; i < n; i++)
          printf("%d ", arr[i]);
}
void omp_lsd_radix_sort(size_t n, unsigned data[n]) {
     unsigned * buffer = malloc(n*sizeof(unsigned));
     int total_digits = sizeof(unsigned)*8;
     //Each thread use local_bucket to move data
     size_t i;
     for(int shift = 0; shift < total_digits; shift+=BASE_BITS) {
          size_t bucket[BASE] = {0};
          size_t local_bucket[BASE] = {0}; // size needed in each
          bucket/thread
          //1st pass, scan whole and check the count
          #pragma omp parallel firstprivate(local_bucket)
          {
               #pragma omp for schedule(static) nowait
               for(i = 0; i < n; i++){
                    local_bucket[DIGITS(data[i], shift)]++;
               }
               #pragma omp critical
               for(i = 0; i < BASE; i++) {
                    bucket[i] += local_bucket[i];
               }
               #pragma omp barrier
               #pragma omp single
               for (i = 1; i < BASE; i++) {
                    bucket[i] += bucket[i - 1];
               }
               int nthreads = omp_get_num_threads();
               int tid = omp_get_thread_num();
               for(int cur_t = nthreads - 1; cur_t >= 0; cur_t--) {
                    if(cur_t == tid) {
                    for(i = 0; i < BASE; i++) {
                         bucket[i] -= local_bucket[i];
                         local_bucket[i] = bucket[i];
                    }
                    } else { //just do barrier
```

```
                #pragma omp barrier
                }
            }
            #pragma omp for schedule(static)
            for(i = 0; i < n; i++) { //note here the end
            condition
                buffer[local_bucket[DIGITS(data[i], shift)]++] =
                data[i];
            }
        }
        //now move data
        unsigned* tmp = data;
        data = buffer;
        buffer = tmp;
    }
    free(buffer);
}

int main() {
    int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
    int n = sizeof(arr) / sizeof(arr[0]);
    // Function Call
    omp_lsd_radix_sort(n, arr);
    print(arr, n);
    return 0;
}
```



```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ gcc randixSortOMP.c -o randixSortOMP -fopenmp
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ time ./randixSortOMP
2 24 45 66 75 90 170 802
real    0m0.051s
user    0m0.334s
sys     0m0.001s
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$
```

C Pthreads

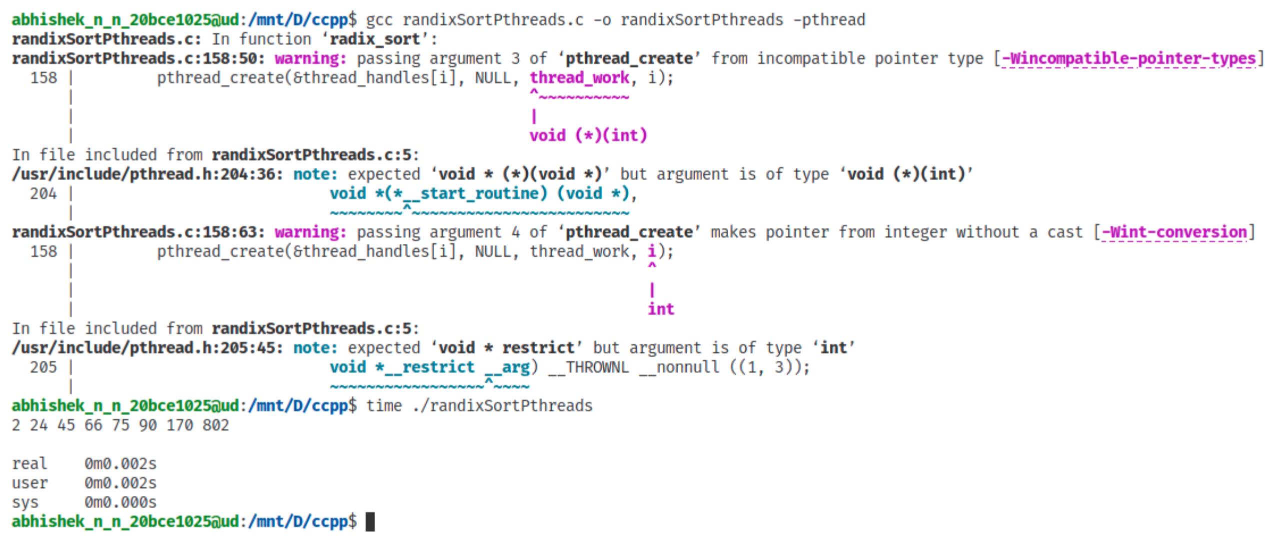
```c
#include <assert.h>
#include <math.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "barrier.h"

#define BITS 29

#define BARRIER_COUNT 1000
pthread_barrier_t barrier;

struct rs_args {
	int id;
	unsigned *val;
	unsigned *tmp;
	int n;
	int *nzeros;
	int *nones;
	int t;
};

struct rs_args *args;
void copy_array(unsigned *dest, unsigned *src, int n) {
	for (; n > 0; n--)
		*dest++ = *src++;
}

void print_array(unsigned *val, int n) {
	int i;
	for (i = 0; i < n; i++)
		printf("%d ", val[i]);
	printf("\n");
}

void initialize_array(unsigned *val, int n) {
	int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
	for (int i = 0; i < n; i++) {
		val[i] = arr[i];
	}
}
```

```
int array_is_sorted(unsigned *val, int n) {
     int i;
     for (i = 1; i < n; i++)
          if (val[i - 1] > val[i])
               return 0;
     return 1;
}

void radix_sort_thread(unsigned *val,unsigned *tmp,int start, int
n,int *nzeros, int *nones,int thread_index,int t) {
     unsigned *src, *dest;
     int bit_pos;
     int index0, index1;
     int i;

     src = val;
     dest = tmp;

     for (bit_pos = 0; bit_pos < BITS; bit_pos++) {
     nzeros[thread_index] = 0;
     for (i = start; i < start + n; i++) {
          if (((src[i] >> bit_pos) & 1) == 0) {
               nzeros[thread_index]++;
          }
     }
     nones[thread_index] = n - nzeros[thread_index];

     pthread_barrier_wait(&barrier);

     index0 = 0;
     index1 = 0;
     for (i = 0; i < thread_index; i++) {
          index0 += nzeros[i];
          index1 += nones[i];
     }
     index1 += index0;
     for (; i < t; i++) {
          index1 += nzeros[i];
     }

     pthread_barrier_wait(&barrier);

     for (i = start; i < start + n; i++) {
          if (((src[i] >> bit_pos) & 1) == 0) {
               dest[index0++] = src[i];
          } else {
               dest[index1++] = src[i];
          }
```

```c
          }

          pthread_barrier_wait(&barrier);

          /* Swap arrays. */
          tmp = src;
          src = dest;
          dest = tmp;
          }
}

void thread_work(int rank) {
          int start, count, n;
          int index = rank;
          n = args[index].n / args[index].t;
          start = args[index].id * n;

          radix_sort_thread(args[index].val, args[index].tmp, start, n,
          args[index].nzeros, args[index].nones, args[index].id,
          args[index].t);
}

void radix_sort(unsigned *val, int n, int t) {
          unsigned *tmp;
          int *nzeros, *nones;
          int r, i;

          long thread;
          pthread_t *thread_handles;

          tmp = (unsigned *)malloc(n * sizeof(unsigned));
          if (!tmp) {
                    fprintf(stderr, "Malloc failed.\n");
                    exit(1);
          }

          nzeros = (int *)malloc(t * sizeof(int));
          if (!nzeros) {
                    fprintf(stderr, "Malloc failed.\n");
                    exit(1);
          }
          nones = (int *)malloc(t * sizeof(int));
          if (!nones) {
                    fprintf(stderr, "Malloc failed.\n");
                    exit(1);
          }

          thread_handles = malloc(t * sizeof(pthread_t));
```

```c
    pthread_barrier_init(&barrier, NULL, t);

    for (i = 0; i < t; i++) {
        args[i].id = i;
        args[i].val = val;
        args[i].tmp = tmp;
        args[i].n = n;
        args[i].nzeros = nzeros;
        args[i].nones = nones;
        args[i].t = t;

        pthread_create(&thread_handles[i], NULL, thread_work, i);
    }
    for (i = 0; i < t; i++) {
        pthread_join(thread_handles[i], NULL);
    }
    pthread_barrier_destroy(&barrier);
    free(thread_handles);
    free(args);
    if (BITS % 2 == 1) {
        copy_array(val, tmp, n);
    }
    free(nzeros);
    free(nones);
    free(tmp);
}
void main(int argc, char *argv[]) {
    int n, t;
    unsigned *val;
    time_t start, end;
    int ok;
    n = 8;
    if (argc > 1)
        n = atoi(argv[1]);
    if (n < 1) {
        fprintf(stderr, "Invalid number of elements.\n");
        exit(1);
    }
    t = 1;
    if (argc > 2)
        t = atoi(argv[2]);
    if (t < 1) {
        fprintf(stderr, "Invalid number of threads.\n");
        exit(1);
    }
    if ((n / t) * t != n) {
        fprintf(stderr, "Number of threads must divide number of
        elements.\n");
```

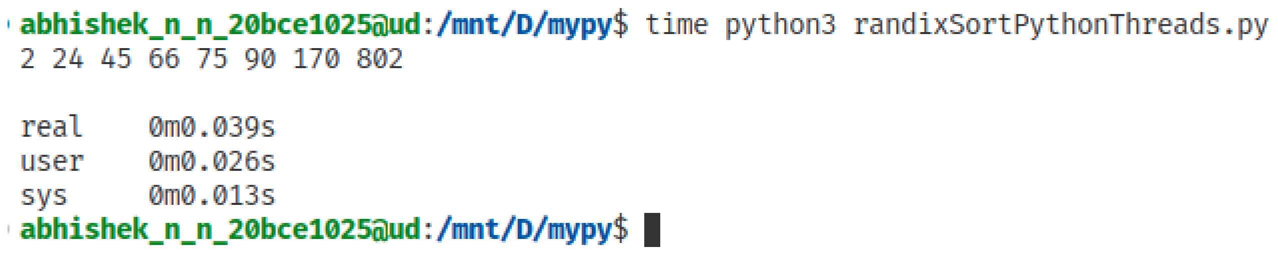
```c
            exit(1);
        }
        val = (unsigned *)malloc(n * sizeof(unsigned));
        if (!val) {
            fprintf(stderr, "Malloc failed.\n");
            exit(1);
        }
        fflush(stdout);
        initialize_array(val, n);
        args = (struct rs_args *)malloc(t * sizeof(struct rs_args));
        if (!args) {
            fprintf(stderr, "Malloc failed.\n");
            exit(1);
        }
        fflush(stdout);
        start = time(0);
        radix_sort(val, n, t);
        end = time(0);
        fflush(stdout);
        ok = array_is_sorted(val, n);

        if (ok && n <= 30) {
            print_array(val, n);
        }

        free(val);
}
```



```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ gcc randixSortPthreads.c -o randixSortPthreads -pthread
randixSortPthreads.c: In function ‘radix_sort’:
randixSortPthreads.c:158:50: warning: passing argument 3 of ‘pthread_create’ from incompatible pointer type [-Wincompatible-pointer-types]
  158 |         pthread_create(&thread_handles[i], NULL, thread_work, i);
      |                                                  ^~~~~~~~~~
      |                                                  |
      |                                                  void (*)(int)
In file included from randixSortPthreads.c:5:
/usr/include/pthread.h:204:36: note: expected ‘void * (*)(void *)’ but argument is of type ‘void (*)(int)’
  204 |                                    void *(*__start_routine) (void *),
      |                                    ~~~~~~~~~^~~~~~~~~~~~~~~~~~~~~~~
randixSortPthreads.c:158:63: warning: passing argument 4 of ‘pthread_create’ makes pointer from integer without a cast [-Wint-conversion]
  158 |         pthread_create(&thread_handles[i], NULL, thread_work, i);
      |                                                               ^
      |                                                               |
      |                                                               int
In file included from randixSortPthreads.c:5:
/usr/include/pthread.h:205:45: note: expected ‘void * restrict’ but argument is of type ‘int’
  205 |                                    void *__restrict __arg) __THROWNL __nonnull ((1, 3));
      |                                    ~~~~~~~~~~~~~~~~~^~~~
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ time ./randixSortPthreads
2 24 45 66 75 90 170 802

real    0m0.002s
user    0m0.002s
sys     0m0.000s
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ █
```

Python Threads

```python
import threading
def countingSort(arr, exp1):
     n = len(arr)
     output = [0] * (n)
     count = [0] * (10)

     for i in range(0, n):
          index = arr[i] // exp1
          count[index % 10] += 1

     for i in range(1, 10):
          count[i] += count[i - 1]

     i = n - 1
     while i >= 0:
          index = arr[i] // exp1
          output[count[index % 10] - 1] = arr[i]
          count[index % 10] -= 1
          i -= 1

     i = 0
     for i in range(0, len(arr)):
          arr[i] = output[i]

def radixSort(arr):
     max1 = max(arr)
     exp = 1

     while max1 / exp >= 1:
          t=threading.Thread(target=countingSort, args=(arr,exp),
          daemon=True)
          t.start()
          exp *= 10

arr = [170, 45, 75, 90, 802, 24, 2, 66]
radixSort(arr)

for i in range(len(arr)):
     print(arr[i], end=" ")
print("");
```

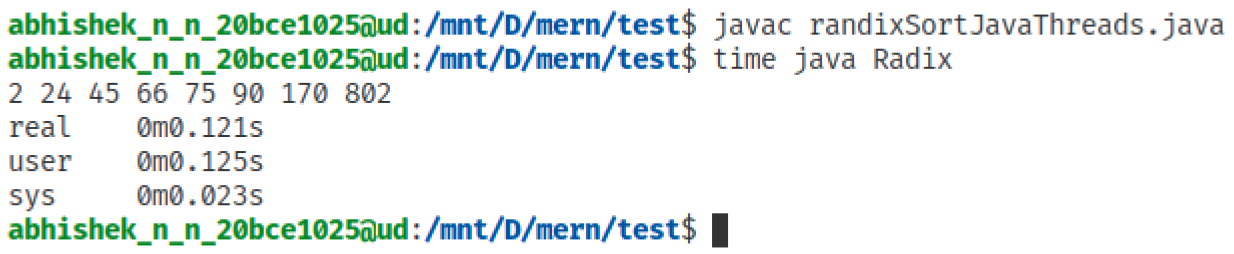


```
abhishek_n_n_20bce1025@ud:/mnt/D/mypy$ time python3 randixSortPythonThreads.py
2 24 45 66 75 90 170 802

real    0m0.039s
user    0m0.026s
sys     0m0.013s
abhishek_n_n_20bce1025@ud:/mnt/D/mypy$
```

Java Threads

```
import java.io.*;
import java.util.*;

class CountSort extends Thread {
     public static int arr[];
     public static int n;
     public static int exp;
     CountSort(int arr[], int n, int exp){
          this.arr=arr;
          this.n=n;
          this.exp=exp;
     }

     public void run() {
          int output[] = new int[n];
          int i;
          int count[] = new int[10];
          Arrays.fill(count, 0);

          for (i = 0; i < n; i++)
               count[(arr[i] / exp) % 10]++;

          for (i = 1; i < 10; i++)
               count[i] += count[i - 1];

          for (i = n - 1; i >= 0; i--) {
               output[count[(arr[i] / exp) % 10] - 1] = arr[i];
               count[(arr[i] / exp) % 10]--;
          }

          for (i = 0; i < n; i++)
               arr[i] = output[i];
     }
}


class Radix {
     static int getMax(int arr[], int n) {
          int mx = arr[0];
          for (int i = 1; i < n; i++)
               if (arr[i] > mx)
                    mx = arr[i];
          return mx;
     }

     static void radixsort(int arr[], int n) {
```

```java
        int m = getMax(arr, n);

        for (int exp = 1; m / exp > 0; exp *= 10){
            CountSort thread = new CountSort(arr, n, exp);
            thread.start();
        }
    }

    static void print(int arr[], int n) {
        for (int i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
    }

    public static void main(String[] args) {
        int arr[] = { 170, 45, 75, 90, 802, 24, 2, 66 };
        int n = arr.length;

        radixsort(arr, n);
        print(arr, n);
        System.out.println("");
    }
}
```



```
abhishek_n_n_20bce1025@ud:/mnt/D/mern/test$ javac randixSortJavaThreads.java
abhishek_n_n_20bce1025@ud:/mnt/D/mern/test$ time java Radix
2 24 45 66 75 90 170 802
real    0m0.121s
user    0m0.125s
sys     0m0.023s
abhishek_n_n_20bce1025@ud:/mnt/D/mern/test$
```