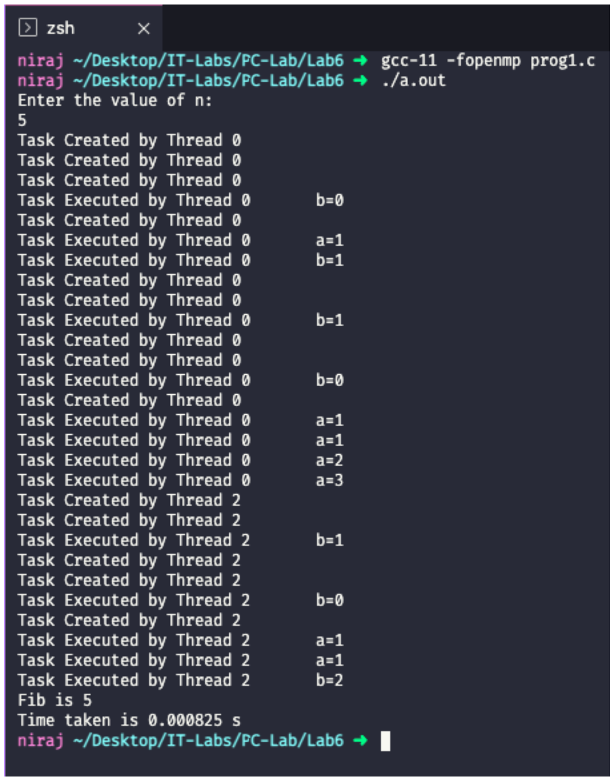
IT301 – Parallel Computing

Assignment – 6

Name: Niraj Nandish

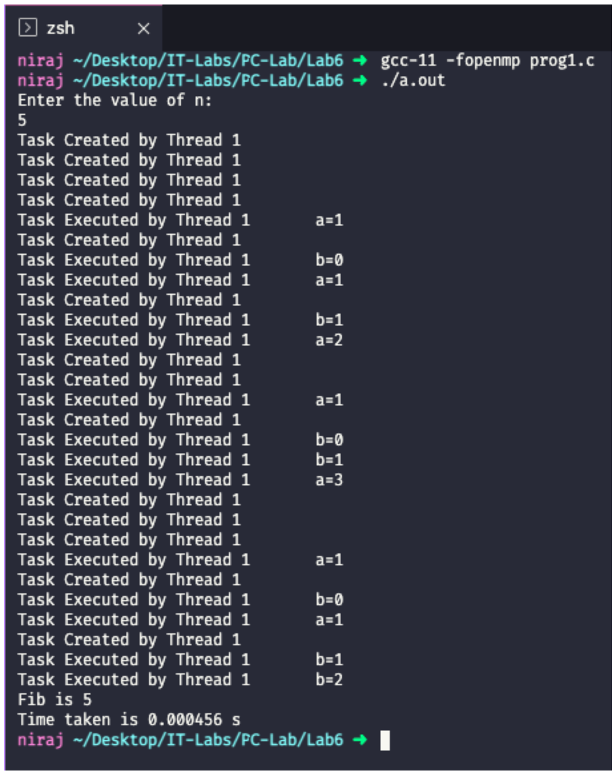
Roll No: 191IT234

1. Program 1
   a. Observation – When the if() clause is present with the condition of "n<5" then when we enter 5 for the value of "n", the tasks become undeferred and a single thread executes the whole program. But when the if() clause is removed, then the tasks gets executed parallelly.



```
niraj ~/Desktop/IT-Labs/PC-Lab/Lab6 ➜ gcc-11 -fopenmp prog1.c
niraj ~/Desktop/IT-Labs/PC-Lab/Lab6 ➜ ./a.out
Enter the value of n:
5
Task Created by Thread 0
Task Created by Thread 0
Task Created by Thread 0
Task Executed by Thread 0       b=0
Task Created by Thread 0
Task Executed by Thread 0       a=1
Task Executed by Thread 0       b=1
Task Created by Thread 0
Task Created by Thread 0
Task Executed by Thread 0       b=1
Task Created by Thread 0
Task Created by Thread 0
Task Executed by Thread 0       b=0
Task Created by Thread 0
Task Executed by Thread 0       a=1
Task Executed by Thread 0       a=1
Task Executed by Thread 0       a=2
Task Executed by Thread 0       a=3
Task Created by Thread 2
Task Created by Thread 2
Task Executed by Thread 2       b=1
Task Created by Thread 2
Task Created by Thread 2
Task Executed by Thread 2       b=0
Task Created by Thread 2
Task Executed by Thread 2       a=1
Task Executed by Thread 2       a=1
Task Executed by Thread 2       b=2
Fib is 5
Time taken is 0.000825 s
niraj ~/Desktop/IT-Labs/PC-Lab/Lab6 ➜
```

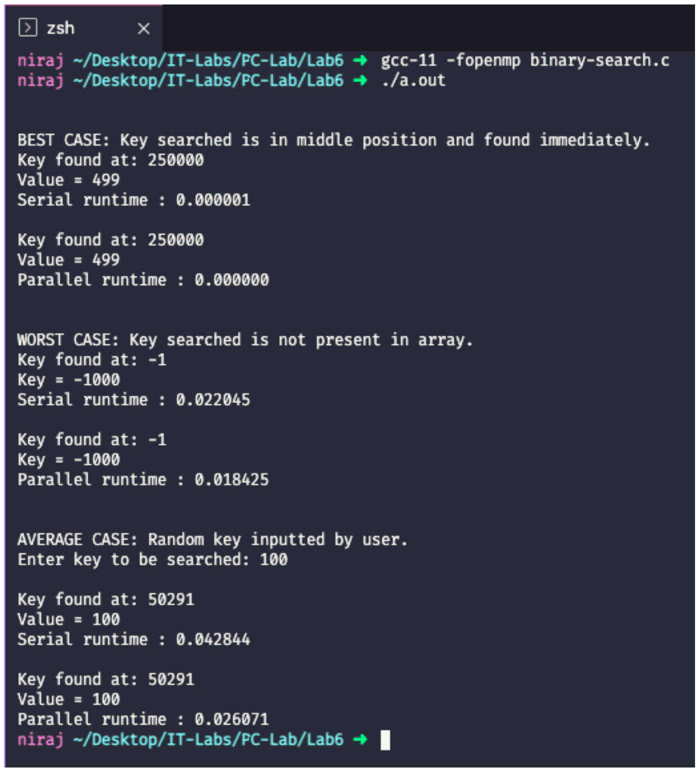
if() clause absent



```
niraj ~/Desktop/IT-Labs/PC-Lab/Lab6 ➜ gcc-11 -fopenmp prog1.c
niraj ~/Desktop/IT-Labs/PC-Lab/Lab6 ➜ ./a.out
Enter the value of n:
5
Task Created by Thread 1
Task Created by Thread 1
Task Created by Thread 1
Task Created by Thread 1
Task Executed by Thread 1       a=1
Task Created by Thread 1
Task Executed by Thread 1       b=0
Task Executed by Thread 1       a=1
Task Created by Thread 1
Task Executed by Thread 1       b=1
Task Executed by Thread 1       a=2
Task Created by Thread 1
Task Created by Thread 1
Task Executed by Thread 1       a=1
Task Created by Thread 1
Task Executed by Thread 1       b=0
Task Executed by Thread 1       b=1
Task Executed by Thread 1       a=3
Task Created by Thread 1
Task Created by Thread 1
Task Created by Thread 1
Task Executed by Thread 1       a=1
Task Created by Thread 1
Task Executed by Thread 1       b=0
Task Executed by Thread 1       a=1
Task Created by Thread 1
Task Executed by Thread 1       b=1
Task Executed by Thread 1       b=2
Fib is 5
Time taken is 0.000456 s
niraj ~/Desktop/IT-Labs/PC-Lab/Lab6 ➜
```

if() clause present

2. Program 2
    a. Observation – We are initializing the array to a size of 500000 with random integer values. In the best-case scenario, the key to be searched is the mid value of the array. Hence, only one comparison will take place. In the worst-case scenario, the key to be searched will not be present in the array. Hence, the entire array has to be traversed. In the average case scenario, the user inputs a number. In all the cases, we see that parallel runtime is lesser than serial runtime.



```
niraj ~/Desktop/IT-Labs/PC-Lab/Lab6 ➜ gcc-11 -fopenmp binary-search.c
niraj ~/Desktop/IT-Labs/PC-Lab/Lab6 ➜ ./a.out


BEST CASE: Key searched is in middle position and found immediately.
Key found at: 250000
Value = 499
Serial runtime : 0.000001

Key found at: 250000
Value = 499
Parallel runtime : 0.000000


WORST CASE: Key searched is not present in array.
Key found at: -1
Key = -1000
Serial runtime : 0.022045

Key found at: -1
Key = -1000
Parallel runtime : 0.018425


AVERAGE CASE: Random key inputted by user.
Enter key to be searched: 100

Key found at: 50291
Value = 100
Serial runtime : 0.042844

Key found at: 50291
Value = 100
Parallel runtime : 0.026071
niraj ~/Desktop/IT-Labs/PC-Lab/Lab6 ➜
```

b. Code

```c
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define size 500000

int binary_search(int s, int e, int array[], int key, int flag)
{
  if (s > e)
  {
    return -1;
  }

  int a, b;
  int m = (s + e) / 2;

  if (array[m] == key)
  {
    return m;
  }
  else
  {
    #pragma omp task shared(a)
    {
      a = binary_search(s, m - 1, array, key, flag);
    }

    #pragma omp task shared(b)
    {
      b = binary_search(m + 1, e, array, key, flag);
    }

    #pragma omp taskwait
    if (a < b)
    {
      return b;
    }
    else
    {
      return a;
    }
  }

  return 0;
}

int comp(const void *a, const void *b)
{
  return (*(int *)a - *(int *)b);
}

int main()
```

```c
{
  int array[size], x, pos;
  double end, start;

  for (int i = 0; i < size; i++)
  {
    array[i] = rand() % 1000;
  }
  qsort(array, size, sizeof(int), comp);

  printf("\n\nBEST CASE: Key searched is in middle position and found immediately.");
  x = array[size / 2];
  start = omp_get_wtime();
  pos = binary_search(0, size, array, x, 1);
  end = omp_get_wtime();
  printf("\nKey found at: %d", pos);
  printf("\nValue = %d", array[pos]);
  printf("\nSerial runtime : %lf", end - start);
  start = omp_get_wtime();
  pos = binary_search(0, size, array, x, 0);
  end = omp_get_wtime();
  printf("\n\nKey found at: %d", pos);
  printf("\nValue = %d", array[pos]);
  printf("\nParallel runtime : %lf", end - start);

  printf("\n\n\nWORST CASE: Key searched is not present in array.");
  x = -1000;
  start = omp_get_wtime();
  pos = binary_search(0, size, array, x, 1);
  end = omp_get_wtime();
  printf("\nKey found at: %d", pos);
  printf("\nKey = %d", x);
  printf("\nSerial runtime : %lf", end - start);
  start = omp_get_wtime();
  pos = binary_search(0, size, array, x, 0);
  end = omp_get_wtime();
  printf("\n\nKey found at: %d", pos);
  printf("\nKey = %d", x);
  printf("\nParallel runtime : %lf", end - start);

  printf("\n\n\nAVERAGE CASE: Random key inputted by user.");
  printf("\nEnter key to be searched: ");
  scanf("%d", &x);
  start = omp_get_wtime();
  pos = binary_search(0, size, array, x, 1);
  end = omp_get_wtime();
  printf("\nKey found at: %d", pos);
  printf("\nValue = %d", array[pos]);
  printf("\nSerial runtime : %lf", end - start);
  start = omp_get_wtime();
  pos = binary_search(0, size, array, x, 0);
  end = omp_get_wtime();
  printf("\n\nKey found at: %d", pos);
  printf("\nValue = %d", array[pos]);
  printf("\nParallel runtime : %lf\n", end - start);
}
```