

Q1

Code:

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

#define SIZE 100
#define SEGMENTS 10
#define SEGMENT_SIZE (SIZE / SEGMENTS)

typedef struct {
    int *arr;
    int start;
} args;

void* thread_func(void* arg) {
    args* a = (args*)arg;
    int sum = 0;

    if (a->start < SIZE) {
        for (int i = a->start; i < a->start + SEGMENT_SIZE; i++)
            sum += a->arr[i];

        int* result = malloc(sizeof(int));
        *result = sum;
        return result;
    }

    int total = 0;
    for (int i = 0; i < SEGMENTS; i++)
        total += a->arr[i];

    int* result = malloc(sizeof(int));
    *result = total;
    return result;
}

int main() {
    int arr[SIZE];
    pthread_t threads[SEGMENTS + 1];
    args thread_args[SEGMENTS + 1];
    int *partial_sums[SEGMENTS];
    int results[SEGMENTS];

    for (int i = 0; i < SIZE; i++) {
        arr[i] = i + 1;
    }

    for (int i = 0; i < SEGMENTS; i++) {
        thread_args[i].arr = arr;
        thread_args[i].start = i * SEGMENT_SIZE;
        pthread_create(&threads[i], NULL, thread_func, &thread_args[i]);
    }
}
```

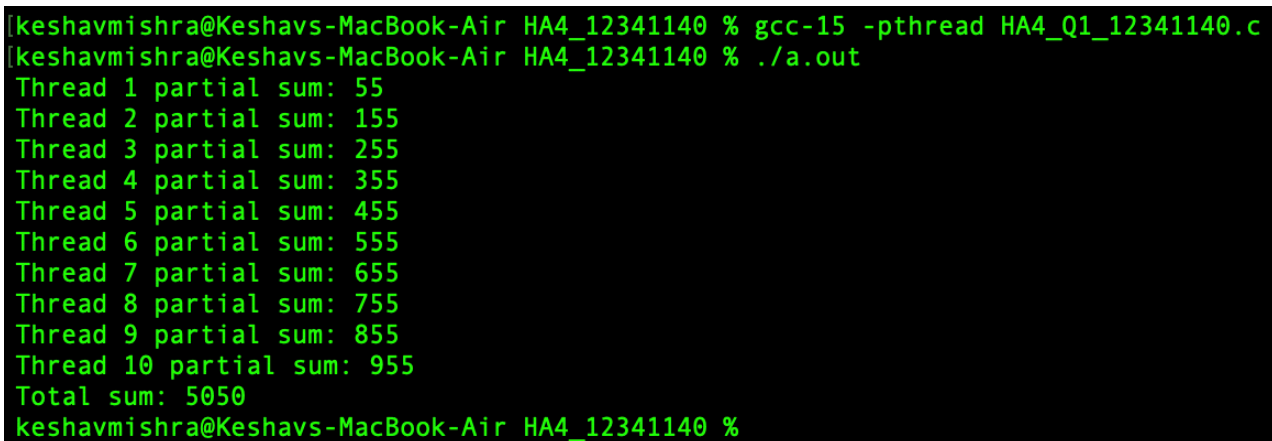
```

}
for (int i = 0; i < SEGMENTS; i++) {
    pthread_join(threads[i], (void**)&partial_sums[i]);
    results[i] = *partial_sums[i];
    printf("Thread %d partial sum: %d\n", i + 1, results[i]);
    free(partial_sums[i]);
}
thread_args[SEGMENTS].arr = results;
thread_args[SEGMENTS].start = SIZE;
int *total_sum;
pthread_create(&threads[SEGMENTS], NULL, thread_func, &thread_args[SEGMENTS]);
pthread_join(threads[SEGMENTS], (void**)&total_sum);

printf("Total sum: %d\n", *total_sum);
free(total_sum);

return 0;
}

```



```

[keshavmishra@Keshavs-MacBook-Air HA4_12341140 % gcc-15 -pthread HA4_Q1_12341140.c
[keshavmishra@Keshavs-MacBook-Air HA4_12341140 % ./a.out
Thread 1 partial sum: 55
Thread 2 partial sum: 155
Thread 3 partial sum: 255
Thread 4 partial sum: 355
Thread 5 partial sum: 455
Thread 6 partial sum: 555
Thread 7 partial sum: 655
Thread 8 partial sum: 755
Thread 9 partial sum: 855
Thread 10 partial sum: 955
Total sum: 5050
[keshavmishra@Keshavs-MacBook-Air HA4_12341140 %

```

Screenshot of output of HA4_Q1_12341140.c

Explanation:

The program divides an array of 100 integers into 10 segments, where each of the first 10 threads computes the sum of one segment. The 11th thread then sums these 10 partial results to get the total sum of all numbers from 1 to 100. The final output shows each thread's partial sum and the total sum 5050.

Q2

Code:

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

#define SIZE 100
#define SEGMENTS 10
#define SEGMENT_SIZE (SIZE / SEGMENTS)

typedef struct {
    int *arr;
    int start;
} args;

void* thread_func(void* arg) {
    args* a = (args*)arg;
    int sum = 0;

    if (a->start < SIZE) {
        for (int i = a->start; i < a->start + SEGMENT_SIZE; i++)
            sum += a->arr[i];
        int* result = malloc(sizeof(int));
        *result = sum;
        return result;
    }

    int total = 0;
    for (int i = 0; i < SEGMENTS; i++)
        total += a->arr[i];
    int* result = malloc(sizeof(int));
    *result = total;
    return result;
}

int main() {
    int arr[SIZE];
    pthread_t threads[SEGMENTS + 1];
    args thread_args[SEGMENTS + 1];
    int *partial_sums[SEGMENTS];
    int results[SEGMENTS];

    for (int i = 0; i < SIZE; i++) {
        arr[i] = i + 1;
    }
    for (int i = 0; i < SEGMENTS; i++) {
        thread_args[i].arr = arr;
        thread_args[i].start = i * SEGMENT_SIZE;
        pthread_create(&threads[i], NULL, thread_func, &thread_args[i]);
    }
    for (int i = 0; i < SEGMENTS; i++) {
```

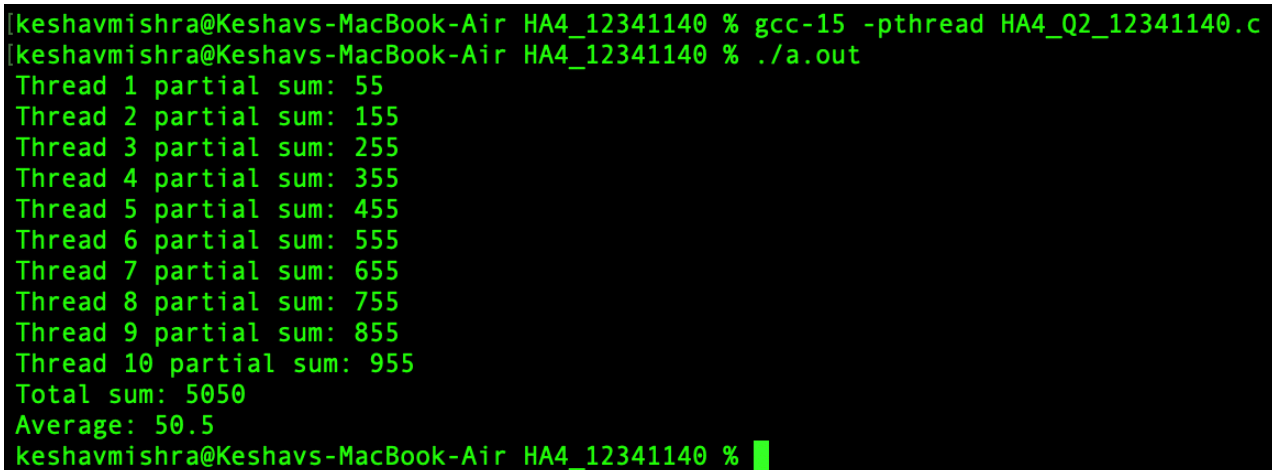
```

pthread_join(threads[i], (void**)&partial_sums[i]);
results[i] = *partial_sums[i];
printf("Thread %d partial sum: %d\n", i + 1, results[i]);
free(partial_sums[i]);
}

thread_args[SEGMENTS].arr = results;
thread_args[SEGMENTS].start = SIZE;
int *total_sum;
pthread_create(&threads[SEGMENTS], NULL, thread_func, &thread_args[SEGMENTS]);
pthread_join(threads[SEGMENTS], (void**)&total_sum);

double average = (double)(*total_sum) / SIZE;
printf("Total sum: %d\n", *total_sum);
printf("Average: %.1f\n", average);
free(total_sum);
return 0;
}

```



```

[keshavmishra@Keshavs-MacBook-Air HA4_12341140 % gcc-15 -pthread HA4_Q2_12341140.c
[keshavmishra@Keshavs-MacBook-Air HA4_12341140 % ./a.out
Thread 1 partial sum: 55
Thread 2 partial sum: 155
Thread 3 partial sum: 255
Thread 4 partial sum: 355
Thread 5 partial sum: 455
Thread 6 partial sum: 555
Thread 7 partial sum: 655
Thread 8 partial sum: 755
Thread 9 partial sum: 855
Thread 10 partial sum: 955
Total sum: 5050
Average: 50.5
[keshavmishra@Keshavs-MacBook-Air HA4_12341140 % █

```

Screenshot of output of HA4_Q2_12341140.c

Explanation:

The program divides the array of 100 integers into 10 parts, where the first 10 threads compute partial sums of each segment. The 11th thread sums all partial sums to get the total, and the main function calculates the average by dividing the total sum by 100. The final output displays all partial sums, total sum 5050, and average 50.5.

Q3

Code:

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

#define SIZE 100
#define SEGMENTS 10
#define SEGMENT_SIZE (SIZE / SEGMENTS)

typedef struct {
    int *arr;
    int start;
} args;

void* thread_func(void* arg) {
    args* a = (args*)arg;
    int max = a->arr[a->start];

    if (a->start < SIZE) {
        for (int i = a->start + 1; i < a->start + SEGMENT_SIZE; i++)
            if (a->arr[i] > max)
                max = a->arr[i];
        int* result = malloc(sizeof(int));
        *result = max;
        return result;
    }

    int overall = a->arr[0];
    for (int i = 1; i < SEGMENTS; i++)
        if (a->arr[i] > overall)
            overall = a->arr[i];
    int* result = malloc(sizeof(int));
    *result = overall;
    return result;
}

int main() {
    int arr[SIZE];
    pthread_t threads[SEGMENTS + 1];
    args thread_args[SEGMENTS + 1];
    int *partial_max[SEGMENTS];
    int results[SEGMENTS];

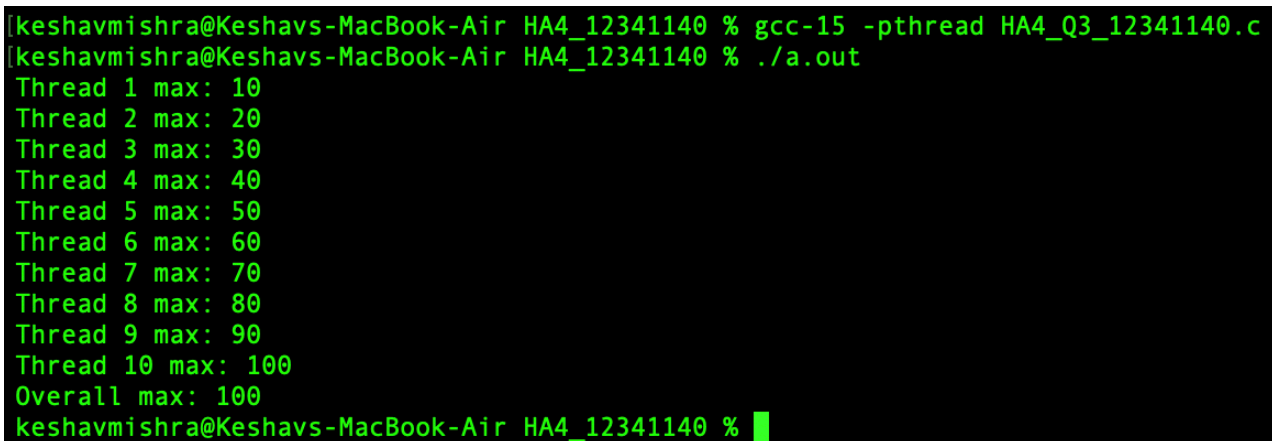
    for (int i = 0; i < SIZE; i++) {
        arr[i] = i + 1;
    }
    for (int i = 0; i < SEGMENTS; i++) {
        thread_args[i].arr = arr;
        thread_args[i].start = i * SEGMENT_SIZE;
        pthread_create(&threads[i], NULL, thread_func, &thread_args[i]);
    }
}
```

```

}
for (int i = 0; i < SEGMENTS; i++) {
    pthread_join(threads[i], (void**)&partial_max[i]);
    results[i] = *partial_max[i];
    printf("Thread %d max: %d\n", i + 1, results[i]);
    free(partial_max[i]);
}
thread_args[SEGMENTS].arr = results;
thread_args[SEGMENTS].start = SIZE;
int *overall_max;
pthread_create(&threads[SEGMENTS], NULL, thread_func, &thread_args[SEGMENTS]);
pthread_join(threads[SEGMENTS], (void**)&overall_max);

printf("Overall max: %d\n", *overall_max);
free(overall_max);
return 0;
}

```



```

[keshavmishra@Keshavs-MacBook-Air HA4_12341140 % gcc-15 -pthread HA4_Q3_12341140.c
[keshavmishra@Keshavs-MacBook-Air HA4_12341140 % ./a.out
Thread 1 max: 10
Thread 2 max: 20
Thread 3 max: 30
Thread 4 max: 40
Thread 5 max: 50
Thread 6 max: 60
Thread 7 max: 70
Thread 8 max: 80
Thread 9 max: 90
Thread 10 max: 100
Overall max: 100
[keshavmishra@Keshavs-MacBook-Air HA4_12341140 % █

```

Screenshot of output of HA4_Q3_12341140.c

Explanation:

The program divides an array of 100 integers into 10 segments, and each of the first 10 threads finds the maximum value in its segment. The 11th thread then compares all segment maxima to determine the overall maximum value in the array. The final output displays the maximum from each thread and the overall maximum, which is 100.