Practical 3

Name: Aaron Emmanuel Rocque
Batch: A2 Roll No: 31
Subject: DAA Lab

Problem Statement:

A grinch is given the job of partitioning 2n players into two teams of n players each. Each player has a numerical rating that measures how good he/she is at the game. He seeks to divide the players as unfairly as possible, so as to create the biggest possible talent imbalance between team A and team B. Show how the Grinch can do the job in O(n log n) time. As the application is memory intensive, hence use a cache-friendly algorithm.

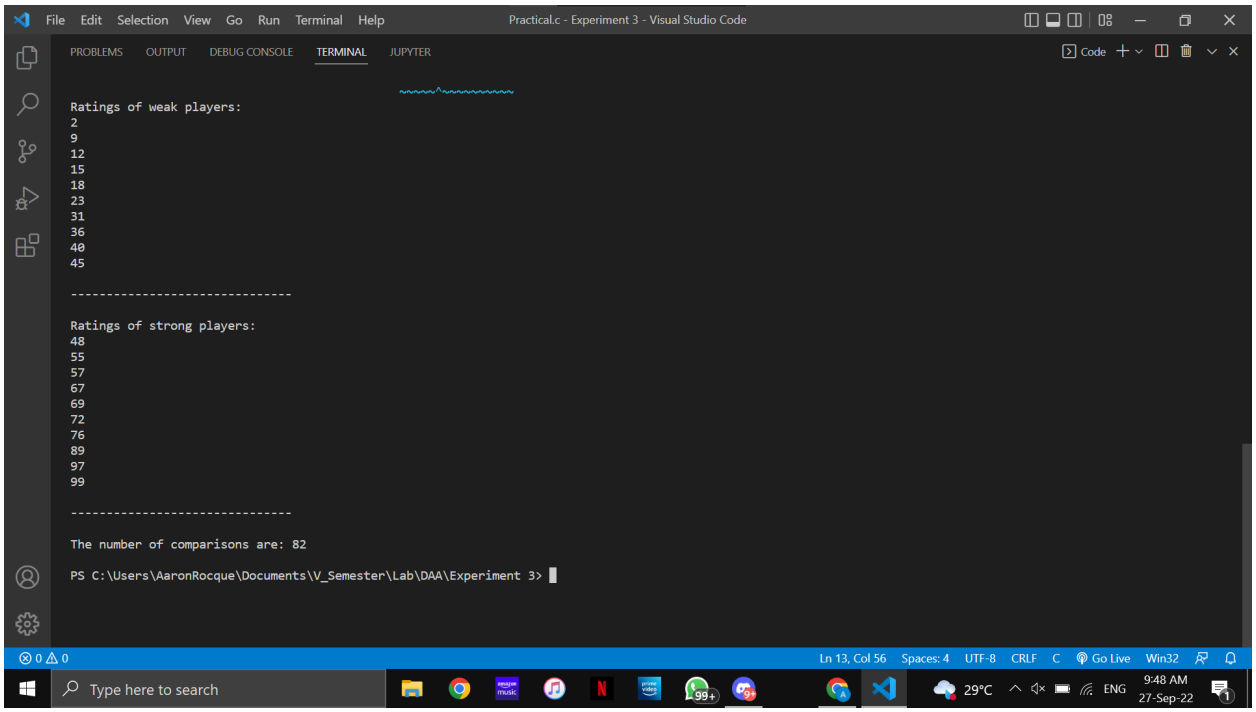
Code:

```c
#include <stdio.h>

int partition(int A[], int start, int end);
void quick_sort(int A[], int start, int end);

int a=0;

int main()
{

    //Rating of players out of 100
    //int A[20] = {9, 12, 72, 15, 45, 23, 67, 40, 18, 55, 89, 2, 99, 31, 57, 69,
76, 97, 36, 48};

    //int A[20] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};

    int A[20] = {20,19,18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1};



    quick_sort(A, 0, 20);
```

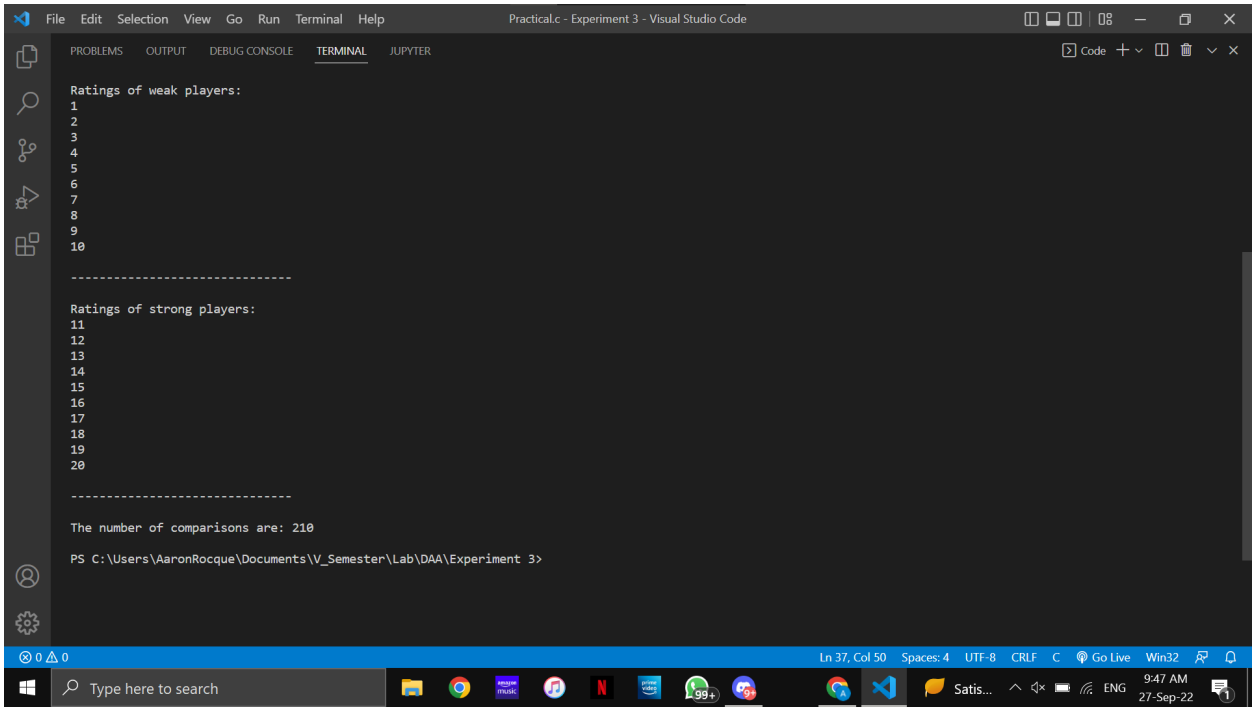
```
    printf("Ratings of weak players:\n");
    for (int i = 0; i < 10; i++)
    {
        printf("%d\n", A[i]);
    }

    printf("\n------------------------------\n\n");

    printf("Ratings of strong players:\n");
    for (int i = 10; i < 20; i++)
    {
        printf("%d\n", A[i]);
    }

    printf("\n------------------------------\n\n");

    printf("The number of comparisons are: %d\n\n", a);
}

int partition(int A[], int start, int end)
{

    int i = start + 1;
    int piv = A[start]; // making the first element of the array as the PIVOT
    int temp;           // used for swapping

    for (int j = start + 1; j <= end; j++)
    {

        /*
        Will loop the array all the way till the end and whenever it finds a
number
        that is less than the PIVOT, it will swap with the PIVOT.
        At the end of the loop, PIVOT is at it's correct position with elements
        less than or equal to, to its left and greater than the PIVOT to it's
left.
        */

        //Swapping
```

```
        if (A[j] < piv)
        {
            temp = A[i];
            A[i] = A[j];
            A[j] = temp;
            i += 1;
        }
        a++;

    }

    // put the pivot element in its proper place.
    temp = A[start];
    A[start] = A[i - 1];
    A[i - 1] = temp;

    return i - 1; // return the position of the pivot
}

void quick_sort(int A[], int start, int end)
{
    if (start < end)
    {
        // stores the position of pivot element
        int piv_pos = partition(A, start, end);
        quick_sort(A, start, piv_pos - 1); // sorts the left side of pivot.
        quick_sort(A, piv_pos + 1, end);   // sorts the right side of pivot.
    }
}
```

Output (Unsorted array - Random):



Output (Completely sorted array):



Output (Completely unsorted array):