Tennis Tournament Handout

Input is a power of 2:

If the input set S has n elements, where n is a power of 2.

1. If $n = 2$, then $S = \{s_1, s_2\}$ schedule match $s_1 \times s_2$.
2. Partition the n items into two subsets, A and B.
3. Let $A = \{a_0, a_1, \cdots, a_{n/2-1}\}$, and $B = \{b_0, b_1, \cdots, b_{n/2-1}\}$. For the first $n/2$ days $d = 1, 2, \ldots, n/2$, schedule matches $a_i \times b_x$, where $x = (i + d - 1) mod n/2$.
4. Use this algorithm recursively to solve the problem for A and B. Notice that it can be done simultaneously. Since the matches for the two sets can be overlapped, this takes only $n/2 - 1$ days (by induction hypothesis). Thus the total schedule takes $n/2 + n/2 - 1$ days.

We show that the algorithm works by induction on the number of recursive calls.

BASIS: If $n = 2$, we have no recursive calls, and the algorithm schedules a championship for n elements in $n - 1$ days.

STEP: Assume the algorithm works for $n = 2^k$. We will show that it works for $n = 2^{k+1}$. Step 4 of the algorithm schedules matches between all elements of A and all elements of B, over a period of $n/2$ days. Since $n/2 = 2^k$, we have by I.H. that step 5 schedules matches between all pairs of elements in A, and all pairs of lements in B, over a period of $\frac{n}{2} - 1$ days. Since these two schedules can be carried out simultaneously, the total scheduling of set S takes $n - 1$ days. Running time: The algorithm will recurse on 2 sets each of size $n/2$ to output their schedules. The time taken to do the "merge" step is clearly $n^2/4$ steps (to output the entire schedule).

$$T(n) = 2T(n/2) + O(n^2)$$

Hence the running time of the algorithm is $O(n^2)$. This is easy to see from the general recurrence we developed in class. (Pick $a = 2, b = 2, k = 2$ and since $a < b^k$ the running time is $O(n^k)$. This is not surprising, since the size of the output is $O(n^2)$.

Input is not a power of 2:

Now we address the case for general n. First we will consider the case when n is odd. If n is odd, then add a "dummy" player to make the number of players equal to $n+1$ (an even number). Schedule these players in $(n+1)-1 = n$ days (in a minute we will show how to do this). Each day, the player who is supposed to play against the dummy player will not play any match (since the dummy isnt a real player).

Now we will consider the case when n is even. The idea is very similar to what we did when n was a power of 2.

1. If $n = 2$, then $S = \{s_1, s_2\}$ schedule match $s_1 \times s_2$.
2. Partition the n items into two subsets, A and B.
3. Let $A = \{a_0, a_1, \cdots, a_{n/2-1}\}$, and $B = \{b_0, b_1, \cdots, b_{n/2-1}\}$.
4. Use this algorithm recursively to solve the problem for A and B. Notice that it can be done simultaneously. Since the matches for the two sets can be overlapped.
5. Now we need to schedule matches between players from set A and players from set B.

In case each of the sets A and B contain an even number of players, then the number of days used up by them is $n/2-1$ and after that we still have $n/2$ remaining days to schedule the matches between A and B. Thus we can fit all the players within $n-1$ days (as required). The harder case is when each of the sets A and B contain an odd number of players (or $n/2$ is odd). We will still schedule A and B simultaneously in $n/2$ days, but now we have to schedule the remaining matches in $n/2-1$ days.

Notice that while A and B are playing their matches (in the recursive calls), on each day there is one free player in each group. (In fact each player is free on *exactly* one day since he plays $n/2-1$ matches in $n/2$ days. Let us now name the players according to the days they are free: On day $d = 1, ..., n/2$ we will say that a_{d-1} and b_{d-1} are free. Hence on day d, we will make a_{d-1} play b_{d-1}. After $n/2$ days, both the groups have been scheduled recursively and the players from each group have played one opposite group player as well. Each player in A (or B) now only has to play $n/2-1$ remaining players from the opposite group. This can be done in $n/2-1$ days as before.

For the next $n/2-1$ days $d = 1, 2, \ldots, n/2-1$, schedule matches $a_i \times b_x$, where $x = (i+d) \bmod n/2$. (Since a_i has already played b_i.) The resulting algorithm obviously has running time $O(n^2)$ (since it only performs $O(n)$ more work than the previous algorithm in the "merge" step).