Problem 4-1

Answer:

The algorithm:

1. Sort the $2n$ players according to their ranks, using Heap sort. (Or other $O(n \log n)$ sorting method)
2. Put the first $n$ players in one team, and the last $n$ players in another.

Running time: Sorting will take $O(n \log n)$ time, and (2) will take $O(n)$ time. Totally, It is an $O(n \log n)$ algorithm.

Problem 4-2

Answer:

(a) Find the $\max$ and $\min$ of the set takes $O(n)$ time, and $|\max - \min|$ is the number we want.

(b) return $|S(n) - S(1)|$.

(c) the algorithm:

1. heap sort the $n$ numbers.
2. return the maximum of $|S(i) - S(i + 1)|$ for $i = 1, \ldots, n - 1$.

(d) same as the second step of (c).
Problem 4-5

Answer:

a. Algorithm:

1. Heap sort the array of numbers $A(1, \ldots, n)$.
2. $\text{mode} = 0$
3. $k = 0$
4. FOR $i = 1$ TO $n-1$ DO
5. \hspace{1em} if $A(i) = A(i+1)$ then $k \leftarrow k + 1$
6. \hspace{1em} else if $\text{mode} < k$ then $\text{mode} \leftarrow k$
7. \hspace{1em} $k \leftarrow 0$
8. END
9. return $\text{mode}$

The running time is $O(n \log n)$.

b. (4) The algorithm:

1. If There is only one number, return it as the answer.
2. Put the $n$ numbers into $\lfloor n/2 \rfloor$ pairs. If there is one number alone (lone number), save for future use.
3. Discard all pairs in which the two numbers are not identical.
4. Discard one number from a pair with two identical numbers.
5. If the number of remaining numbers is odd, discard the lone number if it exists.
6. Repeat the procedure for the remaining numbers.

Proof of Correctness: First observe that in step 3 we discard at least as many bad numbers as good ones, therefore, after step 3, there are more good numbers than bad ones remaining.

If there are odd number of identical pairs, the number of pairs containing good number must exceed the number of pairs containing bad numbers. So if we discard the lone number, the remaining set of numbers still hold the assumption that there are more good numbers than bad ones.
On the other hand, if there are even number of identical pairs, and there is a lone number, it might be the case that the bad number pairs equal to the good number pairs. In this case, the lone number must be good, thus keeping the lone number maintains the assumption. The number in the remaining set is obviously reduced to $\leq \lfloor n/2 \rfloor$. Also according to the pigeon hole principle, there is at least one number remained. Thus our algorithm is right. The running time will be $O(n + n/2 + n/4 + \cdots) = O(2n) = O(n)$.

**Problem 4-6**

**Answer:**

The algorithm:

1. Heap sort $S_1$.
2. For each number $a_i$ in $S_2$ Do
3. binary search $x - a_i$ in $S_1$.
4. If found, return true.
5. End
6. return false

The sorting will take $O(n \log(n))$ time. Each binary search will take $O(\log(n))$ time and there are $n$ of them. Therefore the running time is $O(n \log(n))$.

**Problem 4-12**

**Answer:**

The Algorithm:

1. Form a min heap with the array of $n$ unsorted integers. Time take $O(n)$.
2. Repeat the next two steps $k$ times.
3. Delete the next top element in the heap and put it in the output.

4. Move the last element to the top of the heap and heapify.

Problem 4-13

Answer:

(a) Both
(b) Both
(c) Max-heap
(d) Sorted Array

Problem 4-14

Answer:

The idea is to keep pointers to the $k$ sorted lists, and maintain a heap of size $k$. The first $k$ element in the stack comes from the first element of each of the $k$ sorted array. Each time we extract the maximum/minimum element from the heap to the target array. Suppose this element is from array $i$, then we insert the next element in the array $i$ to the heap. Each insertion takes $O(\log k)$ time, after $n$ times all the elements in the arrays are moved to the target array being sorted. The total cost is therefore $O(n \log k)$.

Problem 4-15

Answer:

No idea. (Everyone will get full credits for this problem.)

Problem 4-16

Answer:
(a) As in the exam, compare in pairs.

(b) The same as (a), we compare in triples. For each triple we need at most 3 comparisons to know its order. And we need another 3 comparisons to pick the 3 largest elements among the 6 numbers (the triple we are looking at along with the 3 largest numbers we found so far) we are looking at. Thus the total comparison is at most $3 + (n - 3)/3 \cdot 6 = 2n - 3$. For this algorithm we need to know the largest and the second-largest keys.

**Problem 4-17**

**Answer:**

\[
\text{RANDOMIZED-SELECT}(A, p, r, i) \\
1 \quad \text{if } p = r \\
2 \quad \text{then return } A[p] \\
3 \quad q \leftarrow \text{PARTITION}(A, p, r) \\
4 \quad q \leftarrow q - p + 1 \\
5 \quad \text{if } i = k \\
6 \quad \text{then return } A[q] \\
7 \quad \text{elseif } i < k \\
8 \quad \text{then return RANDOMIZED-SELECT}(A, p, q - 1, i) \\
9 \quad \text{else return RANDOMIZED-SELECT}(A, q + 1, r, i - k)
\]

**Problem 4-18**

**Answer:**

(a) $\theta(n^2)$. Since each time an array with length $i$ will be partitioned two parts with length $i - 1$ and 0.

(b) $\theta(n \log n)$. Since each time the array will be partitioned into the parts with similar length.

(c) $\theta(n^2)$. Since each time an array with length $i$ will be partitioned two parts with length $i - 2$ and 1.
(d) $\theta(n \log n)$. Since each time the array will be partitioned into the parts with similar length.

**Problem 4-19**

**Answer:**

(a) The recursion for the algorithm is $T(n) = 2T(n/2) + n$, which we have $T(n) = n \log n$.

(b) The recursion for this algorithm is $T(n) = T(n/3) + T(2n/3) + n$,

![Figure 1: The estimation of the recursion $T(n) = T(n/3) + T(2n/3) + n$.](image)

As shown in figure 1, the computation of $T(n)$ can be regarded as a unbalanced binary tree, the cost of each tree level sums up to $n$. Thus the upper bound for $T(n)$ is $n \log_{3/2} n$, the lower bound is $n \log_3 n$, which implies $T(n) = \Theta(n \log n)$.

**Problem 4-24**

**Answer:**

Same as the exam question.

**Problem 4-25**

**Answer:**
(a) Same as 4-24, here \( k = O(\log n) \).

(b) The lower bound is derived for any array without knowing more information of the array to be sorted. (The assumption about the array is different).