

# CS341 Final F2008

1. [Not on final]  
A) Bad.

B)  $T(n) = 2T(n/2) + O(n)$   
mergeKArrays(A)  
    if (A has 1 element)  
        return A  
    lower = lower half of A  
    upper = upper half of A  
    res1 = mergeKArrays(lower)  
    res2 = mergeKArrays(upper)  
    return merge(res1, res2)

2. We first prove that the problem is in NP. Given a truth-value assignment, it takes polynomial-time to check whether the Boolean formula evaluates to true.

We prove that this problem is a reduction of 3-SAT. Given an instance of 3-SAT with  $n$  variables,  $m$  clauses, and Boolean formula  $S$ , we construct an instance of our problem as follows. The Boolean formula  $F$  is the same as  $S$ . We set  $k = n$ .

If there is a solution to this instance of 3-SAT, then there is a satisfying truth-value assignment that satisfies  $F$ . There are at most  $k$  variables set to true, since  $k=n$  and there are at most  $n$  variables. Then we have a solution to our instance of Cheapest 3-SAT. If there is a solution to our instance of Cheapest 3-SAT, we have found a satisfying truth-assignment for the 3-SAT formula, so there is a solution to the instance of 3-SAT.

It takes polynomial-time to construct our Cheapest 3-SAT instance, since  $F = S$  and  $k = n$ . Since 3-SAT is NP-complete, Cheapest 3-SAT must also be NP-complete.

3. A) ??

B) Run the Floyd-Warshall algorithm on  $G$ . The shortest path from a vertex  $v$  to itself is the shortest cycle in the graph. Floyd Warshall takes  $O(n^3)$  time.

4. A)  $2+5 = 7$ ,  $1+3+3 = 7$

B) We first prove that this problem is in NP. Given two sets  $S1$  and  $S2$ , it takes polynomial-time to take the sums of both sets and check whether they are equal. Hence we have a polynomial-time verification algorithm for this problem.

We prove that this problem is a reduction of subset sum. Given an instance of subset sum with positive integers  $a_1 \dots a_n$  and target value  $K$ , we construct an instance of our problem as follows.

Our set of numbers for Balance is  $S = a_1 \dots a_n$ . Let  $S$  be the sum of  $a_1 \dots a_n$ . We add  $S - 2K$  to the set of numbers.

If there is a solution to this instance of subset sum, there is a set of numbers in  $a_1 \dots a_n$  that sum to  $K$ . If we add element  $(S - 2K)$  to this subset, its sum is now  $(S - K)$ . Our remaining numbers in Balance must then sum to  $(S - K)$ . Hence we have found a solution for this instance of our problem.

If there is a solution to this instance of our problem, we have partitioned our set of numbers into two sets of equal sum. The sum of all numbers is  $S + S - 2K = 2S - 2K$ , so each subset must have sum  $S - K$ . One of these subsets must contain the element  $S - 2K$ . Then its remaining elements must sum to  $K$ , and we have found a subset of  $a_1 \dots a_n$  that sums to  $K$ .

It takes linear time to construct the instance of our problem from subset sum. Hence our problem is a polynomial-time reduction of subset sum. Since subset sum is NP-complete, our problem is also NP-complete.

C) Our algorithm works as follows. We first take the sum of all the numbers, let's call it  $S$ . If  $S$  is odd, we terminate. Now we want to find a subset of the numbers that sums to  $S/2$ . Now we can just use the subset sum algorithm to solve the problem. Our recurrence is

$\text{subsum}[i, L] = (\text{subsum}[i+1, L - a_i] \text{ OR } \text{subsum}[i+1, L])$ , where  $\text{subsum}[i, L]$  is whether there is a subset in  $\{1 \dots i\}$  with sum  $L$ .

There are  $S \cdot (S/2)$  subproblems ( $S$  choices for  $i$  and  $S/2$  choices for  $L$ ). Each subproblem can be solved by looking up 2 values. Hence the time complexity is  $O(S^2)$ .

5. [Not covered]

6. A) Only if  $\epsilon$  is sufficiently small.

B) False [Not covered]

C) True

D) Well we haven't found one yet

E) True [Not covered]

7. We construct our subproblem as follows.  $P(j, k) = \text{true}$  if the substring of  $s$  from index  $j$  to  $k$  is a palindrome. Every  $P(j, j) = \text{true}$ . For all  $j < k$ , our recurrence is

$P(j, k) = (s[j] == s[k]) \text{ AND } ((k - j \leq 2) \text{ OR } P(j+1, k-1))$

Then our result is the largest  $k - j$  for all  $P(j, k) = \text{true}$ . In other words,  $P(j, k)$  is true if the characters at  $j$  and  $k$  are the same and the substring between characters  $j$  and  $k$  is a palindrome.

There are  $O(n^2)$  subproblems, since  $j, k$  are both in the range  $[1, n]$ . Each subproblem takes  $O(1)$  time to compute, so we have that the runtime is  $O(n^2)$ .