

**Part I.** (30 points) Solve each of the following problems. Partial credit may be awarded in certain cases. Show all work, and be sure to thoroughly explain your answer when necessary. On algorithm problems, you can use any language (English, pseudo-code, Python, Java, etc.).

1. Describe a greedy approach to solve the following problem. Also find the running time. Justify that your algorithm works.

You are given a set of  $n$  points  $X = \{x_1, x_2, \dots, x_n\}$ , where each point is a real number on the number line. You want to cover points using intervals of a constant length  $l$  (also given). Your goal is to find a set of  $N$  intervals  $I = \{i_1, i_2, \dots, i_n\}$  that covers all points using the minimum number of intervals ( $N$ ); each element of  $I$  represents the starting point of one interval. An interval  $i$  covers a point  $x$  if  $x \geq i$  and  $x \leq i + l$ .

For example, if you have the points  $X = \{1, 2, 3\}$  and  $l = 1$ , one optimal solution would be  $I = \{1, 2\}$ , while another would be  $\{0, 2\}$ .

9 pts

2. Johnny Appleseed wants to maximize the distance he travels so that he can plant the most apple seeds, and grow lots of trees. You are given a directed graph  $G = (V, E)$ , and a weight function  $w$ . The vertices represent potential fields that Johnny can plant apple trees in, while the edges represent traveling between these fields.  $w$  takes a vertex  $a \in V$  and returns a positive real number representing the number of seeds that field can hold. The directed graph will actually be a tree (no cycles), so the answer will not be  $\infty$  (if he keeps going along the cycle, planting more and more trees in the same fields).  $V$  will also be sorted such that all children of a vertex come after that vertex (and each parent of a vertex comes before the vertex).

Describe an efficient algorithm to compute the total weight of the longest path (i.e., the sum of the weights of the edges comprising the path where this sum is maximized). Then, find the running time of your solution.

12 pts
--------

3. The following problem deals with max-heaps. A max-heap is a complete binary tree satisfying the max-heap property. A complete binary tree is a binary tree where all levels are filled except possibly the last, and in the last nodes are filled from left to right. The max-heap property indicates that the children of a node have a lower value than the node (or, the parent of each node has a higher value than the node).
- (a) Describe an algorithm to optimally maintain a max-heap when the largest (topmost) element is removed.
  - (b) Describe an algorithm to optimally maintain a max-heap when a new element is inserted.
  - (c) Find the running time of each of the above algorithms in terms of  $n$ , the number of elements in the tree. Justify your answer.

9 pts

- 
4. Calculate the running time of the following algorithm. First, write a recurrence relation (express the running time for an input of size  $n$  in terms of an input of some smaller size). Then, solve the recurrence relation (find a non-recursive running time solution).

9 pts
7 pts

---

**Algorithm 1** Mystery

---

**Require:**  $A$  and  $B$ , arrays of length  $n$

```

 $X_1 = A[1 : \frac{n}{2}]$  { $X_1$  contains the first half of  $A$ }
 $X_2 = A[\frac{n}{2} : n]$  { $X_2$  contains the second half of  $A$ }
 $X_3 = B[1 : \frac{n}{2}]$ 
 $X_4 = B[\frac{n}{2} : n]$ 
 $c = \text{Mystery}(X_1, X_3)$ 
 $d = \text{Mystery}(X_1, X_4)$ 
 $e = \text{Mystery}(X_2, X_3)$ 
return  $\text{get}(n, c * d * e)$ 
    
```

Assume that the get function takes  $O(n)$  time, and the multiplication of  $c$ ,  $d$ , and  $e$  takes constant time.

---

**Part II.** (30 points) Solve each of the following problems; your final answer should be either a number or a choice in a multiple choice question. Show all work used to arrive at your solution for partial credit.

1. A linked list would be less effective than an array for a task that heavily uses which of the following operations? Insert element (not necessarily at the end), delete (not necessarily from the end), look-up by index, sort, look-up by value (find index of a value).

3 pts

2. Consider the following function. What value is returned when `mystery(60)` is called?

```
int mystery(int x) {
    int v = 0;
    for(int d = x/2; d < 2 * x; d++) {
        v += x % d;
    }
    return v;
}
```

7 pts

3. Consider the following function. How many lines are printed when `mystery(111222333)` is called?

```
void mystery(int x) {
    for(int i = 0; i < 3 && x > 0; i++) {
        println(x);
        x = x / 10;
        mystery(x);
    }
}
```

10 pts

4. Evaluate the expression,  $53_{20} - 37_8$ . Your answer must be in base ten.

4 pts

5. Consider the following function. What value is returned when `juiciness(true, 20, 10)` is called?

```
double juiciness(bool check, double x, double y) {
    if(!check) {
        x += y;
        --y;
    } else {
        x--;
        y++;
        x -= y;
    }

    return x / y;
}
```

6 pts

**Solutions to TAMS Tournament 2011**

**Part I.**

1. Each interval should occur at the leftmost (lowest) uncovered point. For example, the first interval should begin at  $x_1$ . The running time of this algorithm is, then,  $n$ .  
 Justification: suppose you are given an optimal solution  $O$ . If the leftmost interval in  $O$  begins at a point right of  $x_1$ , then the solution is infeasible (it doesn't cover all points). Assuming that the interval begins at or left of  $x_1$ , we can shift the interval right until we reach  $x_1$  without uncovering any points. This idea can be applied to the next interval as well; therefore, our solution is correct. □
2. Go through  $V$  in the sorted order. If the current vertex  $x$  has no parents, set its  $f(x)$  to  $w(x)$ . Otherwise, set  $f(x)$  to  $w(x)$  plus the maximum  $f(x)$  of its parents. After looping through all vertices, return the highest  $f(x) \forall x \in V$ . □
3. ( a ) Assume without a loss of generality that the left child is always greater than the right node. Then, replace the top element with its left child. The top element now has three nodes; so, make the center child a child of the leftmost child. Now, the top element has two children, but its leftmost child has three. Repeat the process of replacement until the bottom of the tree is reached. □
3. ( b ) Insert the element in the next position (that is, in the leftmost position of the bottom row). If the inserted element is greater than its parent, replace it with its parent. If its new parent is still greater, replace again. Repeat until it has no parent (it is at the root of the tree) or it is not greater than its parent. □
3. ( c ) The running time of each algorithm is  $O(\log n)$ . This is because both algorithms run in time with respect to the height of the tree, and height of a complete binary tree is  $O(\log n)$ . □
4. The recurrence relation is  $T(n) = 3T(n/2) + n$ . Solving this yields  $O(n^{\log_2 3})$ . □

**Part II.**

1. Look-up by index only. □
2.  $x \% d = x - d$  for  $d$  between  $x/2$  and  $x$ , and  $= x$  for  $d$  above  $x$ . Thus, the answer is  $.5 * (29 * 30) + 60 * 59$ . This is 3975. □
3. If recursion is not considered, then an execution of mystery with a number of three or more digits would print exactly three lines; with two digits, it would print two lines, and with one digit, it would print one line.

Additionally, each execution would call mystery three times, with one less digit, two less digits, and three less digits. In other words, the number of executions of mystery with  $n$  digits would be equal to the number of executions with  $n+1$ ,  $n+2$ , and  $n+3$  digits. Thus, we can create a table:

```

111222333: 3 lines
11122233: 3 lines
2 x 1112223: 2 * 3 lines = 6 lines
4 x 111222: 4 * 3 lines = 12 lines
7 x 11122: 7 * 3 lines = 21 lines
13 x 1112: 13 * 3 lines = 69 lines
24 x 111: 24 * 3 lines = 72 lines
44 x 11: 44 * 2 lines = 88 lines
81 x 1: 81
    
```

Adding these up, we get 325. □

4. 72 □
5. 8/11 □