

Practice Midterm exam, Spring 2009, CS16

This practice exam has 6 problems. The real exam will contain three or four problems, which will be rather similar in style (although not identical) to three or four of these problems.

0. You need a priority queue that will hold n items, with n large. Your choices are (1) a heap-based queue, and (2) a list-based queue, in which the items are held in unsorted order. You'll first insert the n items, and then, *after* you're done inserting items in the queue, you'll perform `extractMin` a *very* small number of times ... in fact just *eight* times. (a) Which queue implementation do you choose, and why? (b) Can you think of a superior implementation, and explain its superiority. Hint: it's NOT a sorted list-based queue!

1. The following code transforms an array of real numbers, an array whose size, n , is a power of two; the original array is overwritten during the process, and the transformed data is returned in the original array. (The transformation is superficially similar to one that's sometimes important in computer graphics.)

```
fiddle(A, s, f, n)
// transform a portion of the array A of size n.
// the transformed portion are A[i] where s <= i < f.
// Initially invoked with fiddle(A, 0, len(A), len(A)).

if (n == 1) return A;
mid = s + (1+f-s)/2; // The middle index between s and f

for i = s to mid-1
    A[i] = A[i] + 1;
A = fiddle(A, s, mid-1, n/2);
A = fiddle(A, mid, f, n/2);
return A;
```

If the input array A is $[5, 1, 8, 8, 6, 4, 3, 1]$, then after the first loop, A will be $[6, 2, 9, 9, 6, 4, 3, 1]$. The “fiddle” procedure is then called recursively on the two halves, and so on until each part becomes just one element long, and which point it returns the array. (The result is $[7, 3, 9, 9, 7, 5, 3, 1]$, but that's not important.)

When asked to analyze this algorithm in terms of the size, n , of the input array, Billy says it's $O(n \log n)$, Ellen says it's $O(n^2)$, and Nancy says they're both right, because it *is* $O(n \log n)$, and any procedure whose running time is $O(n \log n)$ also has running time that is $O(n^2)$.

- (i) Pick one of the following
(a) Billy alone is right

- (b) Ellen alone is right
- (c) Nancy, Billy, and Ellen are all right
- (d) None of the above.

(ii) Give a sentence or two of explanation for your answer.

2. I've written a function

```
int f(int n)
```

that takes in *positive* integers and returns positive integers. In fact, it's an *increasing* function; in general, $f(i) > f(i-1)$. But my function is really only defined for integers n in the range $0, 1, \dots, K$, where K is some secret positive integer. I've therefore written the function so that it reads

```
int f(int n) {
  const int K = <hidden>;
  if (n <= 0) throw an exception
  if (n > K) return 0;
  else
    ...
}
```

I want you to find out the largest value taken on by the function f , but I'm not going to tell you the value of K ; you have to find that out by yourself. For example, the function f *could* take on the values

```
f(1) = 6
f(2) = 188
f(3) = 222
f(4) = 112121
f(5) = 0
f(6) = 0
etc.
```

In this case, K is 4 (for the numbers $x = 1, 2, 3, 4$, f is increasing, and for $x > 4$, f is zero), and the maximum value of the function is 112121, i.e., $f(4)$.

Write pseudocode for `FIND_MAX`, which finds the largest value taken on by the function f . Small credit for an $O(K)$ algorithm; more credit for a more efficient algorithm.

3. You're given a `WeakQueue` (a `Queue` that has no "size()" method) full of numbers; you are to compute the *alternating sum* of the numbers, i.e., the first, minus the second, plus the third, and so on. There may be an even or an odd number of numbers in the queue. If the queue contains 3, 4, 7, 9 (with 3 being the head of the queue), then the answer is $3 - 4 + 7 - 9 = -3$. (a) Write a *recursive* procedure for computing this

alternating sum. It may help to write a helper-procedure, but you may be able to write the code without one. (b) What's the big-O running time for your procedure, in terms of the number n of items in the queue initially?

Note: points subtracted for ugly procedures.

4. You're considering using one of two algorithms to solve a problem.

Algorithm A solves a problem of size n by dividing it into three subproblems of size $n/4$, recursively solving each subproblem, and then combining the solutions in time $O(n)$ to produce a final answer.

Algorithm B solves a problem of size n by recursively solving three subproblems of size $n/2$, and then combining the solutions in time $O(1)$ to produce a final answer.

(Both A and B can solve a problem of size 1 in $O(1)$ time.)

You expect to use your algorithm to work on problems for which n is very large. Which algorithm do you choose, and why?

5. The *span* of a node in a binary tree is the absolute value of the difference between its height and its depth. Describe how to modify the three "visit" procedures in an Euler tour to compute the span of every node in a binary tree. Assume that the binary tree is unmarked at the start: it merely consists of nodes and links, and there is no "height" or "depth" information at any node. You may, in the course of your tour, decorate the node with any values you wish; when you're done, every node should be decorated with its span, i.e., the code

```
n.span = <something>
```

should have been executed for every node n of the tree.