CS 240E: Data Structures and Data Management **Winter 2023**

Midterm Practice Problems

Note: This is a sample of problems designed to help prepare for the midterm exam. These problems do *not* encompass the entire coverage of the exam, and should not be used as a reference for its content.

1 Runtime Analysis

1.1 Expected run-time analysis

Consider the following pseudocode. What is its expected run-time?

```
int randomizedMystery(int n) {
   if (n<=1) return 0
   else {
      i = random(n)
      randomizedMystery(i)
   }
}
```

1.2 Amortized analysis

Consider the following method to simulate a queue with two stacks S_1, S_2.

- initialize: initialize both S_1 and S_2 as empty stacks
- enqueue(x): do $S_1.push(x)$
- dequeue(): If S_2 is empty, do $S_2.push(S_1.pop())$ until S_1 is empty. Then return $S_2.pop()$.

Using a suitable potential function, argue that it is a potential function and show that all operations take constant amortized time.

2 Sorting algorithms

2.1 Sorting an array

Given an array $A[0...n-1]$ of numbers, show that if $A[i] \geq A[j]$ for all j with $j \leq i - \log n$, the array can be sorted in $o(n \log n)$ time.

2.2 Multi-Way Merge

Given a set of k sorted arrays, where the combination of the k arrays has n elements in total, design a worst case $O(n \log k)$ time algorithm that produces a single sorted array containing all n elements.

3 Skip Lists

Suppose you have a skip list with only three levels. The top level contains only the sentinels. The lowest level has $n + 2$ keys: $-\infty, a_0, \cdots, a_{n-1}, \infty$, while the middle level contains $k + 2$ keys including the sentinels. Assume k divides n. Suppose that the k entries are evenly spread out and the first entry corresponds to a_0.

1. What is the worst case time for a query? Give a tight bound involving k and n.
2. Given n, how should you choose k to minimise the worst case, and what does the worst case become in that case?

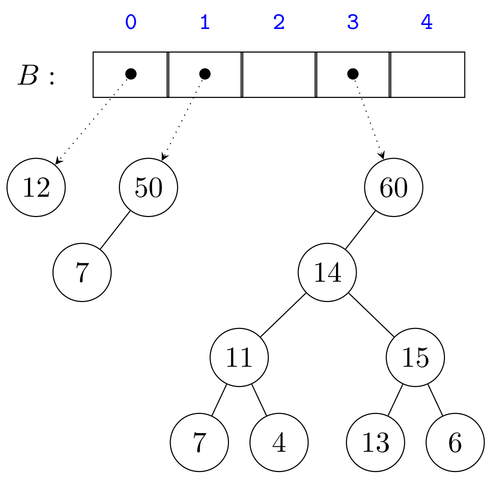
4 Lower Bound

Suppose you own n electrical devices. Each of them comes with a charger cable, which you tossed into a box when you got it. But now it is time to recharge the devices, and so you must find for each one the correct charger cable. For each device, exactly one charging cable is correct. The charging cables look similar enough that you cannot compare them amongst themselves. The only thing that you can do is plug a cable into a device, which will tell you whether the plug fits, or is too big, or is too small.
Argue that any algorithm to find cables for all devices must use $\Omega(n \log n)$ such operations in the worst case.

5 Misc. Computation

5.1 Binomial heap.

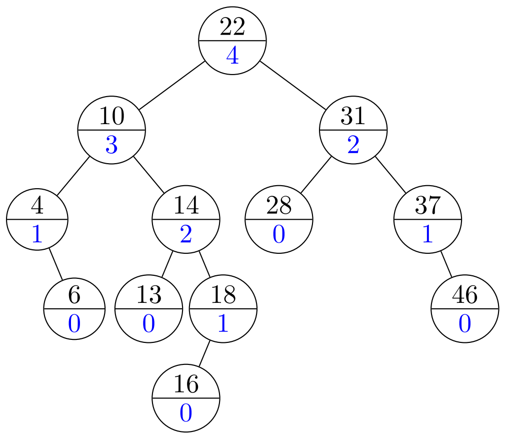
Consider the following binomial heap.



1. Show the resulting binomial heap when performing *insert*(40).
2. Show the resulting binomial heap when performing *deleteMax*.

5.2 AVL-trees

Consider the following AVL-tree.



1. Could there be an AVL-tree that (like this one) has height 4, but fewer nodes? Why not?
2. Show the result after performing *insert*(17).
3. Show the result after performing *delete*(4).
4. Pick a node z and a child y of z such that performing a single rotation at z to bring y up yields again an AVL-tree.
5. Pick a node z and a child y and grandchild x of z such that performing a double rotation at z to bring x up yields again an AVL-tree.

5.3 Priority queues

Our goal is to determine the output of a device that records a number x, and then prints x every $t(x)$ seconds.

More specifically, given an array t of n positive integers, the device will *print* the i-th item every $t[i]$ seconds. It is guaranteed that t is such that no two prints ever occur at the same time.

Give an algorithm to determine the first k prints in $O(kn\log(kn))$ time.

5.4 Asymptotics with rounding

Prove or disprove: there exists a polynomial $p(n)$ such that

$$\lceil \log\log n \rceil ! \in O\left(p\left(n\right)\right).$$

5.5 Average case analysis

Consider a deterministic linear search algorithm *deterministic-search*(A, x). The algorithm searches the input array A for the element x in order, considering $A[0]$, $A[1], \ldots, A[n]$ until it either finds $A[i] = x$, or it reaches the end of the array.

1. Suppose there is exactly one index i such that $A[i] = x$. What is the average-case runtime of *deterministic-search*?
2. Suppose there are $k \geq 1$ indices i such that $A[i] = x$. What is the average-case runtime of *deterministic-search* now?

5.6 Uniform random permutations

Consider the following procedure for generating a permutation of the array A:

```
permute_by_cyclic(A[0..n-1]):
    B[0..n-1] = new array
    offset = random(n)
    for(i = 0..n-1):
        dest = (i + offset) % n
        B[dest] = A[i]
    return B
```

1. Show that each element $A[i]$ has a $1/n$ probability of going to any particular position in B.
2. Show that the resulting permutation is **not** uniformly random.

5.7 Root-leaf path

We are given a complete binary tree of height $h > 0$ with nodes numbered $0, \ldots, n-1$ in level order:

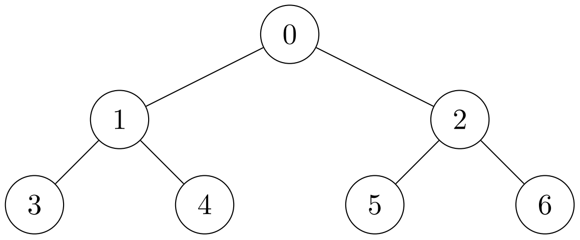


Figure 1: Example tree.

Every node has a flag which is either *true* or *false*. Initially, all flags are *false*.

We will be *dropping balls* one by one from the root of the tree. Each time a ball is dropped it first visits an internal node, an either follows to the left subtree, or to the right subtree. If the node's flag's current value is *false*, then ball goes to the left; and if the value is *true*, the ball goes to the right. In both cases, the ball switches this flag's value. The ball stops when it reaches a leaf node.

Give an algorithm to determine the number of the leaf node at which the i-th ball stops in $O(h)$ time. Note that the runtime of the algorithm is independent of i.

5.8 Batch processing

This problem is harder than what we should expect on the midterm. Nonetheless, it is very good review of the techniques we saw in class, lazy deletion in particular.

We are given a two-dimensional grid that contains n cells. Initially, all cells except one are white. We perform n operations, each of which first calculates the minimum distance from a given white cell to a black cell, and then paints the white cell black.

Assume that we have an $O(n)$ algorithm *closest*(w) which returns the closest black cell to the white cell w. Give an $O(n\sqrt{n})$ algorithm to process all n operations.