CSE321 Homework 03

Aikeboer Aizezi
131044086

1. a) Euclid's algrithm $\rightarrow$ $gcd(m,n) = gcd(n, m \bmod n)$.

size of new instance of the problem = $m \bmod n$.

because $0 < m \bmod n < n$, the size n can dcrease by $L \in (0, n)$.

b) formula for decreasing by factor 2:

$$gcd(m,n) = gcd(n,r) = gcd(r, n\%r) \quad , r = m\%r.$$

there two case for r: ① $r \leq n/2$ ② $\frac{n}{2} < r < n$.

① if $r \leq \frac{n}{2}$ then, $n\%r < r \leq \frac{n}{2}$

② if $\frac{n}{2} < r < n$, then $n \cancel{\%} r = n - r < n/2$.

So, $n\%r \leq \frac{n}{2}$.

2. Algorithm:

- Assume all permutations of (1, 2, ... (n-1) are available.
- Start with inserting n into 1, 2, ...,(n-1) by moving right to left.
- then switch direction every time a new permutation of (1,2, ... (n-1) is processed.

~~start~~

~~insert~~

3. 1. if the key is a leaf, make the pointer from it's parent to the key's node null. if it didn't have a parent, make the tree empty.

2. If key's node has one child, make the pointer from its parent to the key's node to point to that single child. if the node is a root, and have one child only, make that child the new root.

3. If the key's node has two children, first the smallest key m in the right subtree. then, exchang m with the immidiat successor (N). finally, delete N in it's new node by using either case 1 or case 2 depending on whether that node is leaf or has ~~one~~ single child.

a) This is not a variable-size-decrease algorithm. Because it dosen't redue the poblem to that of deleting a key from a smaller binary tree.

b) Since finding the smallest key in the right subtree takes following n-2 pointers, the worst case efficiency is $\Theta(n)$. height of a binary tree of n random keys is ~~algor~~ logarithmic, so the average case is $O(\log(n))$.

4. let $M[1,2,\dots n]$ be the input array.
the invarient will be $M[1,2..i]$ containing only -1, $M[i+1,..j]$ containing only 0, $M[l,\dots n]$ containg only 1.

we initialize i and j with 0, l with n+1. we can check if the array is sorted or not by looking at ~~w~~ if l equals j+1.

if $M[j+1] == -1$, in the loop, we swap $M[j+1]$ with $M[i+1]$, and the we do $++i$, $++j$. if $M[j+1]$ equal 0, we just do $++j$. if $M[j+1]$ equals 1, then we swap $M[j+1]$ with $M[l-1]$, then we do $--l$. this algorithm takes $O(n)$ in the worst case.

5. Given a sorted array of distinct integers $A[1,2 \ldots n]$;

algorithm:
- if $n == 1$, check $A[n]$, if $A[n] == n$ return 1, else 0.
- $k = \lceil \frac{n}{2} \rceil$ (upper)
- if $A[k] == k$, return true.

  else if $A[k] > k$, we call this functio inself with input $A[1, 2,,, k-1]$.

  else call this fuction with input$(A[k+1, \ldots n] - k)$.

in this algorithm, we reduce the problem size by $\frac{1}{2}$ in each division, and each function call takes constant time, so

$$T(n) = \underbrace{T(\tfrac{n}{2})}_{\text{reduced by } \frac{1}{2}} + \underbrace{O(1)}_{\text{funcall}}$$

$$= O(\log n).$$