Problem Set 1

All parts are due Tuesday, February 28 at 11:59PM.

Name: Faaya Abate Fulas

Collaborators: Marwa Abdulhai

Part A

Problem 1-1.

(a) Group 1:

$f_1(n) = O(n)$
$f_2(n) = O(\log(\log n))$
$f_3(n) = O(n \log))$
$f_4(n) = O(\log n)$
$f_5(n) = n \log \sqrt{n} = 0.5n \log n = O(n \log n)$

Since $f_2(n)$ reduces the size of the problem by its square root, its order of growth is slower than $f_4(n)$ which reduces it by half each time. Therefore, the arrangement of the functions in increasing order of growth is $f_2, f_4, f_1, (f_3 = f_5)$

(b) Group2:

$f_1 = O(n^{6.006} \log n)$
$f_2 = n^2 \log n^{6.006} = 6.006n^2 \log n = O(n \log n) = O(n^2 \log n)$
$f_3 = O(n^3)$
$f_4 = O(n^2 \log n)$
$f_5 = O(n^3 \log n)$

Arrangement: $(f_2 = f_4), f_3, f_5, f_1$

Problem 1-2. I used the Master Theorem method to solve all the recurrences below.

(a) $T(n) = \theta(n)$ since $n^{\log_b a} = \theta(n) > f(n) = \theta(1)$

(b) $T(n) = \theta(n \lg n)$ since $n^{\log_b a} = \theta(n) = f(n) = \theta(n)$ and $k = 0$

(c) $T(n) = \theta(n^{\lg 3})$ since $n^{\log_b a} = \theta(n^{\lg 3}) > f(n) = \theta(n)$

(d) $T(n) = \theta(\log n)$ since $n^{\log_b a} = \theta(1) < f(n) = \theta(\log n)$

(e) $T(n) = \theta(n^2)$ since $n^{\log_b a} = \theta(n) < f(n) = \theta(n^2)$

(f) $T(n) = \theta(n^{\lg 7)})$ since $n^{\log_b a} = \theta(n^{\lg 7}) > f(n) = \theta(n^2)$

Problem 1-3.

1. Start with $n = 0$. Check if $f(0) \geq 0$. If so, done. Else, try $n+ = 1$.
2. Check if $f(1) \geq 0$. If so, done. Else, try $n+ = 1$.
3. Check if $f(2) \geq 0$. If so, done. Else, try $n* = 2$.
4. Do step 3 until an n is found such that $f(n) \geq 0$ holds. Then, set n as an upper bound and $\frac{n}{2}$ as a lower bound and perform binary search in that interval.

The recurrence for this algorithm would be:

$$T(n) = T(2n) + \emptyset(\log n)$$

And, since $n^{\log_{0.5} 1} = \theta(1) < f(n) = \theta(\log n)$, the runtime complexity of the algorithm is $\theta(\log n)$. n can be written as some multiple of k where $k \leq n \leq 2k$ holds, so the runtime complexity can also be expressed as $\theta(\log k)$

Problem 1-4.

Data Structure:

- A global max-heap of all packages with $x.priority$ as key
- A dictionary(hash table) with $x.zip$ as key and a local max-heap as value. $x.priority$ will be the key of the local heaps as well.
- Pointers mapping the location of each package in the global heap to its location in the local heap containing it inside the dictionary.

Let $size(D)$ be the size of the global max-heap and $size(M)$ be the size of some local max-heap in the dictionary. So, $size(D) \geq size(M)$

Operations

INIT(Z):

- Initialize an array for the global max-heap and a dictionary for the local max-heaps

INSERT(D,x):

- Invoke the heap operation $insert(D, x)$ on the global heap. This executes in $O(\log size(D))$ time.
- Find $x.zip$ in the dictionary of local max-heaps.
- If $x.zip$ is in the dictionary, invoke $insert(M, x)$ on the local max heap whose key matches $x.zip$. This executes in $O(\log size(M))$ time. If not, initialize a local max-heap with x as its first element and $x.zip$ as its key in $O(1)$ time.
- Overall this would take $O(\log size(D))$ time since that is what dominates all other running times in INSERT(D,x).

MOST-URGENT(D):

- Since the package is a global max, find $D[0].zip$ in the dictionary and invoke the heap operation $extract - max(M)$on the local max-heap paired to it.This executes in $O(\log size(M))$ time.
- Invoke the heap operation $extract-max(D)$ on the global heap. This executes in $O(\log size(D))$ time.
- Overall this would take $O(\log size(D))$ time since that is the costliest running time in MOST-URGENT(D).

REGIONAL-MOST-URGENT(D,r):

- Find r in the dictionary and locate the maximum priority package M[0] in the global heap using its pointer. This executes in O(1) time because of the use of pointers.
- Invoke the heap operation $increase - key(D, x, D[0] + 1)$ to bubble M[0] up to the top of the heap. This executes in $O(\log size(D))$ time.
- Invoke the heap operation $extract - max(D)$ to remove it from the heap. This also takes $O(\log size(D))$ time
- Invoke the heap operation $extract-max(M)$ on the local heap. This executes in $O(\log size(M))$ time.
- Overall this would take $O(\log size(D))$ time since that is the costliest running time in REGIONAL-MOST-URGENT(D,r).

Part B

Problem 1-5.

(a) *Submit your implementation on alg.csail.mit.edu*