CS 600 Advanced Algorithms

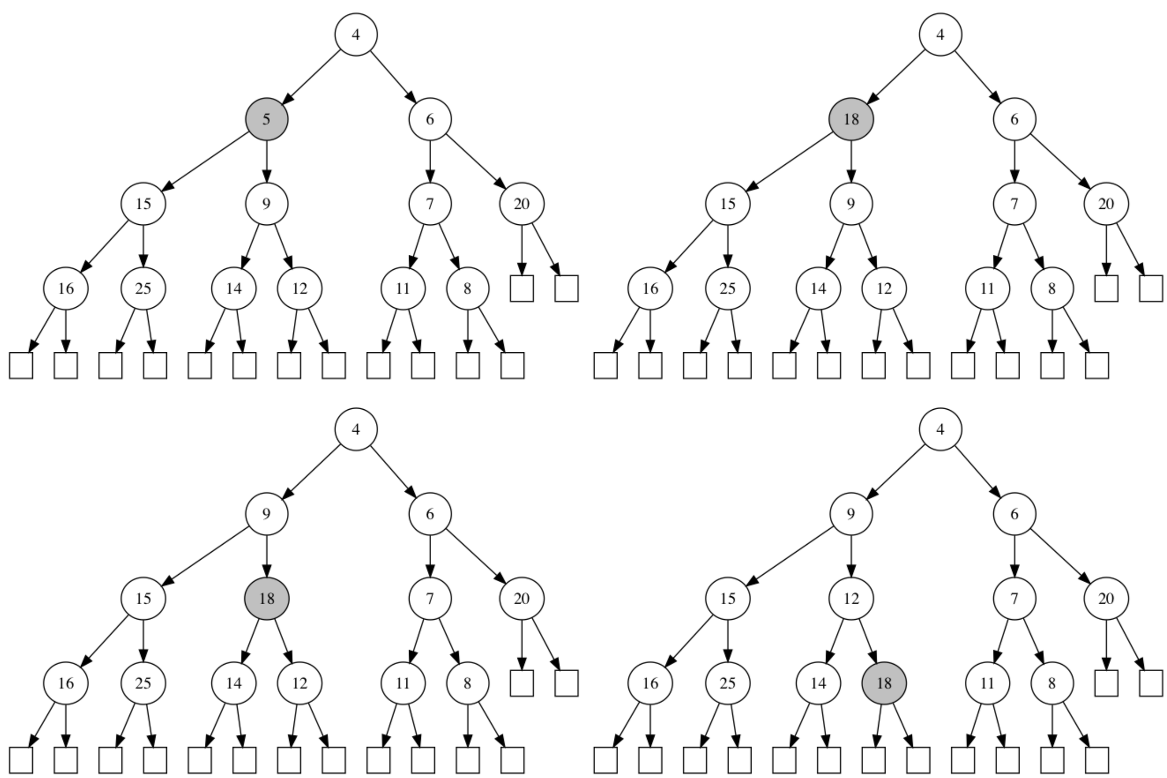
Homework 3 Solutions

1 R-5.6 Worst-case Insertion-sort

The worst-case for insertion sort is a list which is in reverse order. For example, $42, 36, 29, 25, 17, 3, 1$. With such a list, each element will first be moved to the front and then moved back to its original location incrementally. $n - 1$ comparisons are required in order to place the n^{th} element. Summing this up, we get a total of $n(n - 1)/2$ comparisons for a list of length n. This implies $\Omega(n^2)$ time overall.

2 R-5.14 Replacing nodes in a heap

We update the node with key 5 to 18 and continue to swap it with a child node that has the smallest key until we can no longer swap. The steps are shown in the images below.



3 C-5.9 Keys smaller than or equal to a given query in a heap

We assume a tree interface to the heap. We start at the root of T and recursively search the left and right subtrees, if the root of these subtrees has a key smaller than the query, x. We add each such key we encounter to a list and return it once we can no longer recurse into a subtree. This algorithm takes $O(k)$ time where k is the number of elements returned, because there are no nodes in T that have a key larger than x and a descendant with a key less than x.

Algorithm FINDALLLESSTHANQUERY(T, x, v)

Input: A heap T and an query key x
Output: A list of k nodes with keys less than or equal to x

```
if T.isExternal(v) or key(v) > x then
    return
Add v to output list
FINDALLLESSTHANQUERY(T, x, T.leftChild(v))
FINDALLLESSTHANQUERY(T, x, T.rightChild(v))
```

---

4 A-5.3 Frequent Flyer priority

We store upgrade requests in a priority queue, Q, that is implemented with *locator* objects (recall that a locator is a mechanism for maintaining an association between an element and its current position is a container). This ensures that access runs in $O(1)$ time. When a flyer requests an upgrade, we add their request to Q and store the locator for this request in an array D, giving the flyer an index i in D as a confirmation number. If a flyer requests a cancellation, we use their confirmation number, i, and the locator in $D[i]$ to remove the request from Q. To process k upgrades, we perform k removeMin operations on Q. By using a heap to implement the queue, we are assured that all update operations are performed in $O(\log n)$ time. Processing k upgrades takes a total of $O(k \log n)$ time.

5 C-6.6 Multimap data structure

In a *multimap*, as opposed to a map, multiple elements with the same key but different values are allowed. We need to extend the $\mathsf{put}(k, v)$ operation to accomodate multiple values with the same key. We can use a hash function to map the key values into a hash table. Keys in this hash table are associated to linked lists (similar to a regular map with separate chaining). We store all possible multple values for the same key in the linked list associated with that key. The $\mathsf{put}(k, v)$ and $\mathsf{insert}(k, v)$ operations are done at the beginning of the linked list associated with k and run in $O(1)$ expected time. The $\mathsf{findAll}(k)$ operation searches through the linked list at the location where k is hashed to and returns s elements in that list with key k. This runs in $O(1 + s)$ time.

6 A-6.4 Bear-Anteater problem

We start by creating a cuckoo hash table, T, that users (i, j) pairs as keys. The value, c, for a key indicates that the score represented by the pair (i, j) has occured c times in the past during the half-time in games between the Bears and the Anteaters. Initially T is empty. During the processing task, we scan through the list of half-time scores and for each (i, j) pair, we perform a lookup in T. If we find an entry, c, we update it to a new value $c' \leftarrow c + 1$. If we don't find an entry, we insert a new count, $c = 1$. The querying task during half-time is performed by doing a lookup of the current score, (i, j), in T. If we don't find a value, it means that the score hasn't occured in previous games between these two teams. The processing phase takes linear expected time and the query phase takes worst-case constant time.

7 A-6.5 Plagiarism Checker

We use a hash table of size at least $2N$ where N is the total length of the documents in D. Additionally we use a polynomial hash function, h to map a sequence of words to hash values. The important observation is that if you have the hash value, $h(W)$ for a sequence, $W = (w_1, w_2, \ldots, w_m)$ of m words, then you can compute the hash value, $h(W')$, for the next sequence, $W' = (w_2, w_3, \ldots, w_{m+1})$ using the formula

$$h(W') = ah(W) - a^{m-1}w_1 + w_{m+1}$$

where a is the constant used by the polynomial hash function h. This is what allows us to process the document d in $O(n + m)$ expected time instead of $O(nm)$ time, given the assumption that there are not too many collisions (which each take $O(m)$ time to check for plagiarism).