CPSC 320 2020S2: Tutorial 2

Ruolin Li

31764160

1 Asymptotic notation with 1 parameter

For the following operations on a Binary Search Tree (BST) that stores n items (keys or key/data pairs), choose the **tightest worst-case running time** for a good implementation of the operation.

Note: In this class, we use "tight big-O" in an informal sense. We consider a bound $O(f(n))$ to be a tight big-O bound if there is no alternative "reasonable" big-O bound $O(g(n))$ where $g(n) = o(f(n))$. A "reasonable" bound is typically a simple combination of functions such as $\log n$, $\sqrt{n}$, n, n^k, 2^n, or $n!$. There should be no lower-order terms, and no unnecessary constant coefficients.

1. Find the smallest key in a (not necessarily balanced) BST.

   ◯ $O(1)$ ◯ $O(\lg n)$ ◯ $O(n)$ ◯ $O(n \lg n)$ ◯ $O(n^2)$

2. Build a BST whose keys are the numbers between 1 and n, by inserting them in that order.

   ◯ $O(1)$ ◯ $O(\lg n)$ ◯ $O(n)$ ◯ $O(n \lg n)$ ◯ $O(n^2)$

3. Given a key k stored in a *balanced* BST, in which each node stores the size of its subtree, count the number of keys x such that $x < k$. The BST is balanced if its depth is $O(\log n)$.

   ◯ $O(1)$ ◯ $O(\lg n)$ ◯ $O(n)$ ◯ $O(n \lg n)$ ◯ $O(n^2)$

2 Asymptotic notation with multiple parameters

Algorithms that find occurrences of a pattern in a text are used in myriad applications, including text processing as well as detecting interesting motifs (patterns) in genomes (texts). Let $P[1..k]$ and $T[1..n]$ be strings, called the pattern and text, respectively. We say that P occurs at offset o in T if $P[1..k] = T[1 + o..k + o]$. The following algorithm finds the offsets of all occurrences of a pattern in a text:

```
FUNCTION FindPattern(P[1...k],T[1...n]):
       // output the list of all offsets of occurrences of pattern P in text T
       offset_list = [ ]
       for o from 0 to n-k:
           num_matches = 0
           for j from 0 to k-1:
               if T[1+o+j] == P[1+j]:
                  num_matches++
           if num_matches == k:
              add o to offset_list
       return offset_list
```

To analyze the running time of this algorithm, we'll extend big-O notation to functions with two parameters. Specifically, let $T(n, k)$ be the worst-case running time of a given algorithm whose input size is described using a pair (n, k) of nonnegative integers. We say that $T(n, k)$ *is big-O of function* $f(n, k)$ and write $T(n, k) = O(f(n, k))$ if there are nonnegative integers c, n_0, and k_0 such that for all inputs of size (n, k) with $n \geq n_0$ and $k \geq k_0$, $T(n, k) \leq cf(n, k)$.

As a function of n and k, what terms below describe the worst-case running time of Algorithm FindPattern? Check all answers that apply. (Make sure to write down a brief justification for your choices, and don't forget to do that also in future problems and in the assignment.)

◯ $\Theta(k^2)$ ◯ $\Theta((n-k+1)^2)$ ◯ $\Theta((n-k+1)k)$ ◯ $\Theta(nk)$

Tutorial 2

Ruolin Li
31764160
2020.07.11.

For $\Theta(k^2)$. we let $k=1$, then $k^2=1$
$(n-k+1)k \notin O(1)$ which is equivalently $\Theta(k^2)$

For $\Theta((n-k+1)^2)$, we assume $k=n$,
$(n-k+1)^2 = (n-n+1)^2 = 1 \Rightarrow \notin \Theta((n-k+1)^2)$
$(n-k+1)k \notin O(1)$

For $\Theta(nk)$, $(n-k+1)k \in O(nk)$. Let $k=n-1$
$(n-n+1+1)(n-1) = 2n-2 < 2n \leq c \cdot nk$
So $(n-k+1)k < c \cdot nk$ which means
it $\notin \Omega$, as a result, it $\notin \Theta$.

The whole algorithm:

offset_list = []	O(1) time
the first for loop:	O(n-k+1) time
num_matches = 0	O(1) time
the second for loop:	O(K) time
the following steps takes O(1) for each	
return	O(1) time

O(1)
O(1)
O(1)

Θ: $\in O$ and $\in \Omega$

Tutorial 2

Ruolin Li
31764160
2020.07.11.

For $\Theta(k^2)$, we let $k=1$, then $k^2=1$
$(n-k+1)k \notin O(1)$ which is equivalently $\Theta(k^2)$

For $\Theta((n-k+1)^2)$, we assume $k=n$,
$(n-k+1)^2 = (n-n+1)^2 = 1 \Rightarrow \notin \Theta((n-k+1)^2)$
$(n-k+1)k \notin O(1)$

For $\Theta(nk)$, $(n-k+1)k \in O(nk)$. Let $k=n-1$
$(n-n+1+1)(n-1) = 2n-2 < 2n \leq c \cdot nk$
So $(n-k+1)k < c \cdot nk$ which means
it $\notin \Omega$, as a result, it $\notin \Theta$.

The whole algorithm:

offset_list = []	O(1) time
the first for loop:	O(n-k+1) time
num_matches = 0	O(1) time
the second for loop:	O(k) time
the following steps takes O(1) for each	
return	O(1) time