CPSC 221: Theory Assignment 2

Posted: March 5, 2017 @ 23:55 **Due**: Thursday, March 16, 2017 at 23:59

History of Non-Trivial Changes to this Document: **Updated on March 15 @ 18:10**

- Mar. 15 @ 18:10: For Q11, we're OK with using induction on the height of the tree instead of on the number of nodes. The number of nodes and the height are closely related.
- Mar. 15 @ 01:10: Due to the timing mismatch of lectures among the 3 sections, we're going to move Q9 (hashing), Q10 (hashing), and Q12 (preorder/inorder) to Theory Assignment #3.
- Mar. 8 @ 23:15: Use "theory2" for the handin folder.
- Mar. 8 @ 13:25: For Q3, use k > 1.

Submission Instructions

Submit your solutions using `handin`. You can write your solutions by hand and scan the pages or take pictures of them with your phone (but organize them according to the next paragraph); or better still: use a word processing package to typeset your solutions and produce a `.pdf` file.

Important: (1) Clearly label the question number of any file you submit. (2) Don't submit sideways or upside-down photos; these are a pain for the markers to work with. Use PDF to create a single document that can be read in the regular vertical way. Pretend that the marker is going to print your document on 8.5" x 11" paper, and will read down the page(s) in the normal way. (3) If using a photo, please don't use a high resolution because it can quickly consume a lot of space and upload/download time. (4) Be sure to include your name, CS userid, partner's name (if applicable), and partner's CS userid in your file. Two marks for this assignment will be awarded for complying with these rules.

To Submit: Copy the files that contain your solutions to the directory `~/cs221/theory2` in your home directory on an undergraduate machine. (You may have to create this directory using `mkdir ~/cs221/theory2`.) Then, run: `handin cs221 theory2` from your home directory.

We encourage you to work in pairs. **Be sure to include the names and ugrad CS login IDs of both partners on your solution, but only one partner should submit the assignment.**

Late Submission Policy: The late penalty is 3% per hour (or portion thereof), with no late assignments being accepted after 9 hours. For example, if you hand in your program at 02:10 AM on the morning after the due date, then this is 3 hours late; so, you would lose 3 * 3% = 9% of the maximum possible mark.

Part 1: Complexity

1. Use contradiction to prove that x^5 is not $O(x^2)$.

2. Prove that $1^3 + 2^3 + 3^3 + ... + n^3$ is $\Theta(n^4)$.

Part 2: Recurrences

3. Consider the following recurrence relation:

   $E_1 = 0$
   $E_k = E_{k-1} + k + 1$ for all integers $k > 1$

   a. Use substitution (at each iteration) to find an explicit formula for the sequence.
   b. Use induction to verify the correctness of the formula.

Part 3: Quicksort

4. Use Jon Bentley's `Quicksort` algorithm (see the lecture notes) to sort the following array:

5	3	2	8	1	0	6	7	4

However, to choose the pivot element, don't simply pick the first element of the partition being sorted. Instead, use the *median-of-3* rule, that is, for your pivot element for the current partition, choose the median of: {first element, middle element, last element} where the middle element of this partition is defined as follows:

If i is the subscript of the first element, and j is the subscript of the last element, then the subscript of the middle element is $k = \text{ceiling}((i + j) / 2)$.

Thus, for the first call, the subscripts are $i = 0$, $j = 8$, and $k = \text{ceiling}((0 + 8) / 2) = 4$; therefore, we take the median of the three values 5, 1, and 4—which in this case is the value 4. This means the last element (4) gets swapped with the first element (5) before starting the Quicksort algorithm. Note that this operation is done in O(1) time and does not affect the complexity of Quicksort.

For your answer to this question, show the results of each call to `qsort`, and what the next call would be. In other words, trace the execution. It's up to you as to whether you show the trace table or not.

Part 4: Heaps

5. a) Convert the following unordered array to a minimum heap using the Heapify (build heap in O(*n*) time) algorithm. Show the steps as you progress.

E	A	D	C	G	B	F	I	H

   b) Perform **Heapsort** on the resulting heap in (a). Show your work, as you progress.

6. Consider a heap *H* with *n* keys in it. Give an efficient algorithm for reporting all the keys in *H* that are less than or equal to a given query key *q* (which is not necessarily in *H*). For example, if the heap contains $n = 7$ keys, say the elements 2, 4, 6, 8, 10, 20, and 30, and if q=9, then we need to report elements 2, 4, 6, and 8. The keys do *not* have to be reported in sorted order. Your algorithm should run in O(*k*) time, where *k* is the number of keys reported.

Part 5: Program Correctness and Loop Invariants

7. The following *while* loop implements a way to multiply two numbers that was developed by the ancient Egyptians:

   Algorithm: (Note that the indentations describe the code blocks that apply to the scope of the *while* and *if* statements.)

   [Pre-condition: *A* and *B* are positive integers, $x = A$, $y = B$, and $product = 0$.]

```
while ($y$ != 0)
        $r = y$ mod 2
        if ($r = 0$)
                $x = 2 * x$
                $y = y / 2$
        if ($r =1$)
                $product = product + x$
                $y = y – 1$
```

   [Post-condition: $product = A * B$]

   Prove the correctness of this loop with respect to its pre- and post-conditions by using the *loop invariant:*

   "$x*y + product = A * B$".

8. Use a *loop invariant* to prove that when the following algorithm terminates, `pow` is equal to a^n :

```
i = 1
pow = 1
while (i ≤ n) {
      pow = pow * a
      i = i + 1
}
```

Part 6: Hash Functions

9. [Moved to Theory Assignment #3] Which of the following 3 choices would make the best hash function for an 11-digit account number (assuming a random digit between 0 and 9 is assigned for each of the 11 positions)? Briefly, justify your answer by analyzing and comparing the 3 hash functions given below: $h_i(k)$. We'll use open addressing with 1000 array cells, and we'll use linear probing to resolve collisions. There are 250 keys to hash.

   a) Hash function $h_1(k)$ = floor(sqrt(k)) % 1000

   b) Take the 4 digits in the positions 2, 4, 6, and 8 (offsets start at 0, like a C++ array); call them *a*, *b*, *c*, and *d*, respectively; concatenate them; and then compute $h_2(k)$ = (*abcd*) % 1000. Note: *abcd* does *not* mean *a***b***c***d*.

   c) $h_3(k)$ = ((the number of zeroes in k) + (the number of ones in k) + (the number of twos in k) + ... + (the number of nines in k)) % 1000.

10. [Moved to Theory Assignment #3] Using a hash table with 11 locations and the hash function $h(k) = k \% 11$, show the hash table that results when the following integers are inserted in this order:

    26, 42, 5, 44, 92, 59, 40, 36, 12, 60, 80

    … if we assume that collisions are resolved using:

    a) Linear probing
    b) Quadratic probing
    c) Double hashing with the following secondary hash function:

    $h_2(x) = (2x) \% 11$, if this expression is non-zero
    $h_2(x) = 1$, if the expression in the previous line is zero

Part 7: Trees and Induction

11. [Update: We'll accept a solution that uses induction on the height of the tree.] A *ternary tree* is either empty or consists of a node called the root and three ternary trees (called the left, middle, and right subtrees). Prove that a ternary tree of height h has at most $(3^{h+1} - 1)/2$ nodes, by using induction on the number of nodes in in the tree. Note that the empty tree has height –1.

12. [Moved to Theory Assignment #3] If the preorder visitation of nodes (containing one-character variables and operators) in a binary tree is the following sequence: $*a*+bc+*def$ then:

    a) Draw its expression tree.

    b) What is its inorder sequence?