Fundamental Algorithms - Spring 2018
Homework 5

Daniel Rivera Ruiz
Department of Computer Science
New York University
`drr342@nyu.edu`

1. (a) For $k = n$, `MERGE-SORT` takes $T = \Theta(k \log_2(k))$. So with $k = n^2$ we have:
$$T = \Theta(n^2 \log_2(n^2))$$
$$T = \Theta(2n^2 \log_2(n))$$
$$T = \Theta(n^2 \log_2(n))$$
   (b) From $n = 2^m$ it follows that $m = \log_2(n)$, and therefore the time for `ANNA` is given by
$$T = \Theta(m^2 2^m)$$
$$T = \Theta(n \log_2^2(n))$$
   (c) From $n = 2^m$ it follows that $m = \log_2(n)$ and $5^m = 5^{\log_2(n)}$. Therefore, the time for `BOB` is given by
$$T = \Theta(5^m)$$
$$T = \Theta(5^{\log_2(n)})$$
   (d) `COUNTING-SORT` takes $\Theta(\max(N, K))$ for N elements in the range $[0, K]$. In this case we have (for all $n > 1$):
$$T = \Theta(\max(n^2, n^3 - 1))$$
$$T = \Theta(n^3 - 1)$$
$$T = \Theta(n^3)$$
   (e) `RADIX-SORT` takes $\Theta(D \max(N, K))$ for N elements in the range $[0, K^D]$. In this case we have $N = n^2$, $K = n$ and $K^D = n^3 - 1$. In the asymptotic analysis, $n^3 - 1 \approx n^3$ and therefore $K^D \approx n^3$. Finally it follows that $D = 3$ and we have:
$$T = \Theta(D \max(N, K))$$
$$T = \Theta(3 \max(n^2, n))$$
$$T = \Theta(3n^2)$$

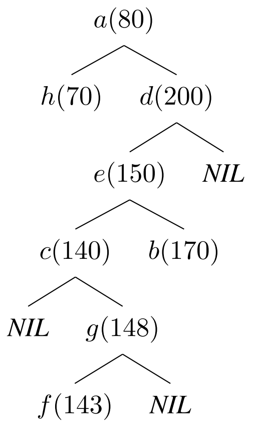
2. (a) `insert`(COBB)
$h(\text{COBB}) = (3 + 15 + 2 + 2) \mod 7 = 22 \mod 7 = 1$
   (b) `insert`(RUTH)
$h(\text{RUTH}) = (18 + 21 + 20 + 8) \mod 7 = 67 \mod 7 = 4$
   (c) `insert`(ROSE)
$h(\text{ROSE}) = (18 + 15 + 19 + 5) \mod 7 = 57 \mod 7 = 1$
   (d) `search`(BUZ)
$h(\text{BUZ}) = (2 + 21 + 26) \mod 7 = 49 \mod 7 = 0$

(e) `insert`(DOC)
$h(\text{DOC}) = (4 + 15 + 3) \mod 7 = 22 \mod 7 = 1$

(f) `delete`(COBB)
$h(\text{COBB}) = (3 + 15 + 2 + 2) \mod 7 = 22 \mod 7 = 1$

h	T_a	T_b	T_c	T_d	T_e	T_f
0	∅	∅	∅	∅ (searched)	∅	∅
1	COBB	COBB	COBB_ROSE	COBB_ROSE	COBB_ROSE_DOC	ROSE_DOC
2	∅	∅	∅	∅	∅	∅
3	∅	∅	∅	∅	∅	∅
4	∅	RUTH	RUTH	RUTH	RUTH	RUTH
5	∅	∅	∅	∅	∅	∅
6	∅	∅	∅	∅	∅	∅

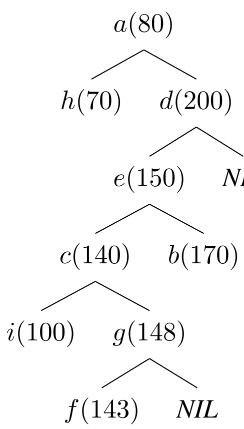
3. Picture of the tree:



The operation `INSERT[i]` where `key[i]=100` will traverse the tree starting at the root, comparing the key of the value to insert with the key of the node until it finds a free spot (*NIL*) in the tree. (If the key to insert is less than the key of the node, the algorithm continues to the left child and it continues to the right child otherwise):

Step	x	$k(x)$	p	$k(i) < k(x)$
0	a	80	*NIL*	false
1	d	200	a	true
2	e	150	d	true
3	c	140	e	true
4	*NIL*	*NIL*	c	finish

As we can see from the table, i should be inserted as the left child of c, resulting in the following updated tree:



4. (a) The output of `FANG[A]` will be $x + 1$. What the function does is add one to the LSB of A (`A[0]++`) and then deal with the possibility of a carry bit in the `while` loop: if there is carry (`A[i] = 2`) the bit is set to zero and the next element of the array is incremented by one. This process continues until there is nothing to carry or until the last element of the array is reached.

   (b) The worst-case time for `FANG` is when the `while` loop is executed the maximum number of times i.e., when the whole array needs to be traversed. For this to happen, all the elements of A must be 1 so there will always be a carry bit. Under these circumstances, the `while` instruction will be reached K times, and since $N = 2^K$, the worst-case time complexity for `FANG` will be $\Theta(\log_2(N))$.
   The best-case time for `FANG` occurs when `A[0] = 0` (regardless of the other values in A) because no carry will be generated and the code inside the `while` loop will never execute. Under these circumstances, the time complexity is clearly $\Theta(1)$.

   (c) What `VIKAS` does is execute `FANG` $N - 1$ times starting with $A = [0, \dots, 0](x = 0)$. This means that after the `for` loop executes once, we will have $A = [1, 0, \dots, 0](x = 1)$, then $A = [0, 1, 0, \dots, 0](x = 2)$ for the second execution and so on until we finally get $A = [1, \dots, 1](x = N - 1)$. The final value of A will have, clearly, K 1's.

   (d) To calculate the time complexity of `VIKAS`, we will simply think of A as the binary representation of x and notice the following:
   - For all A such that $\text{LSB}(A) = 1$ the `while` loop in `FANG` will execute once. This will be true for half of the values A takes while executing `VIKAS` i.e., $\frac{N-1}{2}$.
   - If the following bit in A is also 1, there will be an additional execution of the `while` loop. This will happen for half of the numbers in the previous count, or what it's the same, one fourth of all the possible values of A, which is $\frac{N-1}{4}$.
   - Following this train of thought, we will be executing additional `while` loops in an ever decreasing amount of numbers as we enforce the stronger condition that more bits equal 1.

   With this observations, it is clear that the time complexity of `VIKAS` can be expressed as follows:
   $$T = \frac{N-1}{2} + \frac{N-1}{4} + \ldots + \frac{N-1}{2^K} = \sum_{i=1}^{K} \left(\frac{N-1}{2^i} \right)$$
   To simplify the calculation we will consider the infinite summation. Since we are doing asymptotic analysis, adding these smaller terms will not affect the result:
   $$T = \Theta\left(\sum_{i=1}^{\infty} \frac{N-1}{2^i} \right)$$
   $$T = \Theta\left((N-1) \sum_{i=1}^{\infty} \frac{1}{2^i} \right)$$
   $$T = \Theta(N-1)$$
   $$T = \Theta(N)$$