CSC 225: Fall 2017 Assignment #3

Due at beginning of class, Mon. Oct. 30

The boxes for this assignment:

Question	1	2	3	4	5	6	7
Marks							

For any questions about Big Oh, Omega, or Theta in this class, use these definitions:
Assume that T and f are functions mapping the natural numbers $\{0, 1, 2, 3, \cdots\}$ into the reals. Assume k is an integer, $k > 0$.

Definition (Big Oh): A function $T(n) \in O(f(n))$ if there exist constants $n_0 \geq 0$, and $c > 0$, such that for all $n \geq n_0$, $T(n) \leq c * f(n)$.

Definition (Omega): A function T(n) is in $OMEGA(f(n))$ if there exist constants $n_0 \geq 0$, and $c > 0$, such that for all $n \geq n_0$, $T(n) \geq c * f(n)$.

Definition (Theta): The set $\Theta(f(n))$ of functions consists of $\Omega(f(n)) \cap O(f(n))$.

1. Big Oh notation.

   (a) [5] Prove that $f(n) = \sum_{i=1}^{n} i^k \in O(n^{k+1})$.

   (b) [5] Prove that $f(n) = \sum_{i=1}^{n} i^k \in \Omega(n^{k+1})$.

   (c) [5] Prove that $f(n) = 4n^5 - 16n^4 - 34n^3 - 13n^2$ is in $\Theta(n^5)$.

2. The aim of this question is to analyze the time complexity of the following build heap routine:

   heapify(r)
   If r is not null
   1. Heapify the left subtree.
   2. Heapify the right subtree.
   3. Bubble down the key at node r.

   Assume $n = 2^k - 1$ for some integer k. Then, the recurrence for the work is: $T(n) = \log_2(n+1) + 2\,T((n-1)/2)$, $T(1) = 1$. The point of this question is to find a closed formula for the recurrence and to prove that your answer is correct.

(a) [5] Use repeated substitution to convert this recurrence into a sum.

(b) [5] Prove by induction that $S(r) = \sum_{i=1}^{r} i\, 2^i = (r-1)\, 2^{r+1} + 2$.

(c) [5] Use (b) to help find a closed formula for your sum from part (a).

(d) [5] Prove by induction that your formula for $T(n)$ from (c) is correct.

(e) [5] What does this say about the Big Oh time complexity of this heapify routine?

3. To simplify the mathematics for this question, we will assume that $n = 2^k - 1$ for some integer k. The median of a set of n numbers, n odd, is the value that falls in the middle when the values are sorted. Consider the following algorithm, MedianSort, for sorting:

   1. Find the median in $O(n)$ time.
   2. Divide the problem into three subproblems:
      Problem 1: Keys with value less than the median.
      Problem 2: keys with value equal to the median.
      Problem 3: Keys with value greater than the median.
   3. Solve Problems 1 and 3 recursively.
   4. Marry the solutions by concatenating together the answers from problems 1, 2, and 3.

(a) [5] Assume the values to be sorted are distinct. Assume that the data is stored in a linked list as used for assignments #1 and #2. Explain why
$T(n) = n + 2 * T((n-1)/2), T(1) = 1$
is a reasonable choice for a recurrence relation for estimating the running time for problems of size one or more (up to a constant factor). Your explanation should include a discussion of the time complexities for steps 2, 3, and 4 using the linked lists, but just assume without justification that Step 1 takes $O(n)$ time.

(b) [5] Use the method of repeated substitution to solve the recurrence from part (a) where $n = 2^k - 1$ for some integer k. Show all your work including the Step number (0, 1, 2, ...).

(c) [5] Prove by induction that your answer to part (b) is correct. Be careful here: recall that our problem is only defined for $n = 1, 3, 7, 15, \cdots$ so induction that goes from n to $n+1$ is inappropriate.

(d) [5] How long (in the Big Oh sense) does your MedianSort take to sort n data items with only 3 distinct key values? For example, for $n = 9$ problem could be:
1 3 2 2 1 3 2 1 3
Justify your answer (How deep does the recursion go?).

For questions 4-7, justify all your answers.

4. Consider the begin_program method from the next page.

   (a) [4] Set up a recurrence relation for the running time complexity of this algorithm (in terms of Big Oh).

   (b) [3] what is the solution to your recurrence?

   (c) [3] Give a function $f(n)$ that is as simple as possible such that your formula from (b) is in $\Theta(f(n))$.

5. Consider the begin_program method from the next page.

   (a) [4] Set up a recurrence relation for the space complexity of this algorithm (in terms of Big Oh).

   (b) [3] What is the solution to your recurrence?

   (c) [3] Give a function $f(n)$ that is as simple as possible such that your formula from (b) is in $\Theta(f(n))$.

6. Consider the middle_program method from the next page.

   (a) [4] Set up a recurrence relation for the running time complexity of this algorithm (in terms of Big Oh).

   (b) [3] What is the solution to your recurrence?

   (c) [3] Give a function $f(n)$ that is as simple as possible such that your formula from (b) is in $\Theta(f(n))$.

7. Consider the middle_program method from the next page.

   (a) [4] Set up a recurrence relation for the space complexity of this algorithm (in terms of Big Oh).

   (b) [3] What is the solution to your recurrence?

   (c) [3] Give a function $f(n)$ that is as simple as possible such that your formula from (b) is in $\Theta(f(n))$.

The methods for questions 4-7:

```
public class Assign3
{

   public void begin_program(int n)
   {
      int [ ] A;
      int i;

      if (n <= 1) return;

      A= new int[n];
      for (i=0; i < n; i++)
         A[i]= i;

      begin_program(n-1);
   }

   public void middle_program(int n)
   {
      int [ ] A;
      int i;

      if (n <= 1) return;

      A= new int[n];
      for (i=0; i < n; i++)
         A[i]= i;

      middle_program(n/2);

      middle_program(n/2);

   }
}
```