CS 4720/5720 Design and Analysis of Algorithms
Homework #3a

Daniel Frey

Answers to homework problems:

1. Solve recurrence relations with Master Theorem. If it can't be solved with Master Theorem, then explain why.

   (a) $T(n) = 5T(\frac{n}{3}) + n$
   $a = 5, b = 3, d = 1, b^d = 3^1 = 3$
   $a = 5 > 3 = b^d \Rightarrow$ Case 3
   $T(n) \in \Theta(n^{\log_3 5})$

   (b) $T(n) = 2.7T(\frac{n}{5}) + n^2$
   $a = 2.7, b = 5, d = 2, b^d = 5^2 = 25$
   $a = 2.7 < 25 = b^d \Rightarrow$ Case 1
   $T(n) \in \Theta(n^2)$

   (c) $T(n) = 2T(n-1) + n$
   Master Theorem doesn't apply. Problem is shrinking by constant amount.

   (d) $T(n) = 1.1T(0.2n) + 1$ *(Note:* $0.2 = \frac{1}{5}$*)*
   $a = 1.1, b = 5, d = 0, b^d = 5^0 = 1$
   $a = 1.1 > 1 = b^d \Rightarrow$ Case 3
   $T(n) \in \Theta(n^{\log_5 1.1})$

   (e) $T(n) = 2T(\frac{n}{2}) + n\log_2 n$
   $a = 2, b = 2, d = 1, b^d = 2^1 = 2$
   $a = 2 = 2 = b^d \Rightarrow$ Case 2
   $T(n) \in \Theta(n{\log_2}^2 n)$

   (f) $T(n) = 2T(\frac{n}{2}) + \sqrt{n}$
   $a = 2, b = 2, d = \frac{1}{2}, b^d = 2^{\frac{1}{2}} = \sqrt{2}$
   $a = 2 > \sqrt{2} = b^d \Rightarrow$ Case 3
   $T(n) \in \Theta(n^{\log_2 2}) = \Theta(n)$

   (g) $T(n) = 4T(\frac{n}{2}) + \sqrt{n^4 - n + 10}$
   Informal: $\sqrt{n^4 - n + 10} \approx \sqrt{n^4} = n^2 \in \Theta(n^2)$
   Formal: $\lim_{n\to\infty} \frac{\sqrt{n^4-n+10}}{n^2} = \lim_{n\to\infty} \sqrt{\frac{n^4-n+10}{n^4}} = \lim_{n\to\infty} \sqrt{1 - \frac{1}{n^3} - \frac{10}{n^4}} = \sqrt{1} = 1 \in \Theta(n^2)$
   $a = 4, b = 2, d = 2, b^d = 2^2 = 4$
   $a = 4 = 4 = b^d \Rightarrow$ Case 2
   $T(n) \in \Theta(n^2 \log_2 n)$

   (h) $T(n) = 7T(\frac{n}{3}) + \sum_{i=1}^{n} i$
   $\sum_{i=1}^{n} i = \frac{1}{2}n(n+1) \in \Theta(n^2)$
   $a = 7, b = 3, d = 2, b^d = 3^2 = 9$
   $a = 7 < 9 = b^d \Rightarrow$ Case 1
   $T(n) \in \Theta(n^2)$

(i) $T(n) = 4T(\frac{n}{2}) + n^n$
Master Theorem doesn't apply. $f(n) = n^n$ is not polynomial.

(j) $T(n) = 8T(\frac{n}{3}) + n^3$
$a = 8, b = 3, d = 3, b^d = 3^3 = 27$
$a = 8 < 27 = b^d \Rightarrow$ Case 1
$T(n) \in \Theta(n^3)$

2. Apply the Master Theorem to the divide-and-conquer algorithms to find worst-case order of growth.

(a) Divide into 3 chunks, solve 1 chunk, split is constant division.
$a = 1, b = 3, d = 0, b^d = 3^0 = 1$
$a = 1 = 1 = b^d \Rightarrow$ Case 2
$T(n) \in \Theta(n^0 \log_3 n) = \Theta(\log_3 n)$

(b) Divide into 2 chunks, solve 2 chunks and add, split is constant division, addition is constant.
$a = 2, b = 2, d = 0, b^d = 2^0 = 1$
$a = 2 > 1 = b^d \Rightarrow$ Case 3
$T(n) \in \Theta(n^{\log_2 2}) = \Theta(n)$

3. Re-write merge sort algorithm broken into k pieces.

```
MergeSort(A[0,n-1])
  if n > 1 :
    return Merge(MergeSort of $k$ "chunks")
  ret A
Merge (A[ ] "chunks")
  while (items to be sorted)
    compare elements
      insert lower into output array
    increment output array counter
```

(a) Worst-case efficiency using Master Theorem.
$T_{Merge}(n) = \sum_{i=0}^{n-1} 1 = n \in \Theta(n)$
$T_{MergeSort}(n) = kT(\frac{n}{k}) + \Theta(n)$
$a = k, b = k, d = 1, b^d = k^1 = k$
$a = k = k = b^k \Rightarrow$ Case 2
$T(n) \in \Theta(n \log_k n)$

(b) Better or worse than original merge sort (splits into 2)?
Merge sort into k pieces is in the same class as the ordinary merge sort, however, note that the log bases are different. Therefore, for $k > 2$ merge sort into k pieces will be slightly worse.

(c) Why choose ordinary $k = 2$ merge sort over merge sort that splits into $k > 2$ pieces?
Ordinary merge sort, for $k = 2$, is more often chosen because there is less overhead when combining, since there are less comparisons when compared to a merge sort that's split into $k > 2$ pieces.