Homework 4: Algorithm and Data Structure

Blen Daniel Assefa

February 29

Problem 4.1 *Merge Sort*

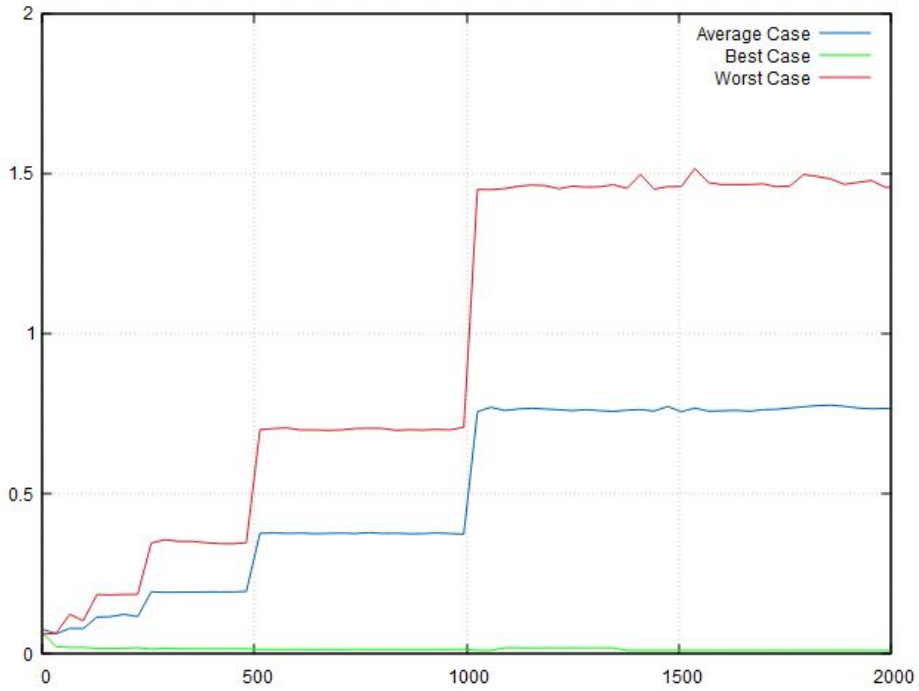
(a) Implement a variant of Merge Sort that does not divide the problem all the way down to subproblems of size 1. Instead, when reaching subsequences of length k it applies Insertion Sort on these n/k subsequences

Please refer to the “Mergesort.c” file.

(b) Apply it to the different sequences which satisfy best case, worst case and average case for different values of k. Plot the execution times for different values of k.

Bestcase: When the list is already sorted
Worstcase: When the list is reversed
Averagecase: When the list elements are randomly generated

Graph looks Like:



(c) How do the different values of k change the best-, average-, and worst-case asymptotic time complexities for this variant? Explain/prove your answer.

Bestcase: For the best case, no matter the value of K, the time complexity is not affected. This is because both mergesort and insertion sort's bestcase time complexity is O(n).

Averagecase: For the average case the mergesort is $\Theta(nlogn)$ and insertion sort is $\Theta(n^2)$. Hence, the time complexity is $\Theta(nlogn + n^2)$ and this is what is displayed for the various values of K.

Worstcase: For the Worst case, insertion sort is $\Theta(n^2)$ but merge sort is just $\Theta(nlogn)$. At first, they will have a time complexity of $\Theta(nlogn + n^2)$, however, as K gets bigger and bigger, just the insertions sort is applied so the complexity becomes $\Theta(n^2)$.

(d) Based on the results from (b) and (c), how would you choose k in practice? Briefly explain.

It depends on the type of array. If the array is already sorted, then the large value of K will mean, I will only apply insertion sort. In fact, if the array has less than 30 elements, then insertion sort is better than stated in the slides.
If the array is large and not sorted, however, I would choose the value of K to be 1, as this way, it applies just the merge sort.

Problem 4.2 Recurrences

Use the substitution method, the recursion tree, or the master theorem method to derive upper and lower bounds for T(n) in each of the following recurrences. Make the bounds as tight as possible. Assume that T(n) is constant for n ≤ 2.

(a) $T(n) = 36T(n/6) + 2n$

Using master method:
$T(n) = aT(n/b) + f(n)$
$a = 36,\ b = 6$
$f(n) = 2n$

$n^{log_b^a} = n^{log_6^{36}} = n^2$
$f(n) = \Theta(n^2)$

(b) $T(n) = 5T(n/3) + 17n^{1.2}$

Using master method:
$T(n) = aT(n/b) + f(n)$

$a = 5,\ b = 3$

$f(n) = 17n^{1.2}$

$n^{log_b^a} = n^{1.46}$

$f(n) = \Theta(n^{1.46-\varepsilon})\ for\ \varepsilon = 0.26$

$T(n) = \Theta(n^{1.46})$

(c) $T(n) = 12T(n/2) + n^2 lg\ n,$

Using master method:

$T(n) = aT(n/b) + f(n)$

$a = 12,\ b = 2$

$f(n) = n^2 lg\ n$

$n^{log_b^a} = n^{3.59}$

$n^{3.59} > n^2 lgn,$ *this will be polynomially greater for some value and we can find some constant* ε

$T(n) = \Theta(n^{3.59})$

(d) $T(n) = 3T(n/5) + T(n/2) + 2^n,$

(e) $T(n) = T(2n/5) + T(3n/5) + \Theta(n).$