CH08-320201 Huzeifa Hashim

Algorithms & Data Structures.

Homework # 2

Problem 2.2 Recurrences.

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

$\therefore$ Master method since of the form $T(n) = aT(n/b) + f(n)$.

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

$f(n) = 2n.$

$\therefore 2n < n^2$

since the function $f(n)$ is polynomially smaller than $n^{\log_b a}$

$T(n) = \Theta(n^2)$

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

Master method.

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

$n^{\log_b a} = n^{1.465}$

$\therefore f(n) < n^{1.465}$

$\therefore T(n) = \Theta(n^{1.465})$

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

$n^{\log_b a} = n^{3.585}$

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

since. $n^{\log_b a} > fn$

$T(n) = \Theta(n^{3.585})$

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

using the master method.

$3T(n/5) + T(n/2) \leq 11T(n/10)$

$\therefore 11T(n/10) + 2^n$

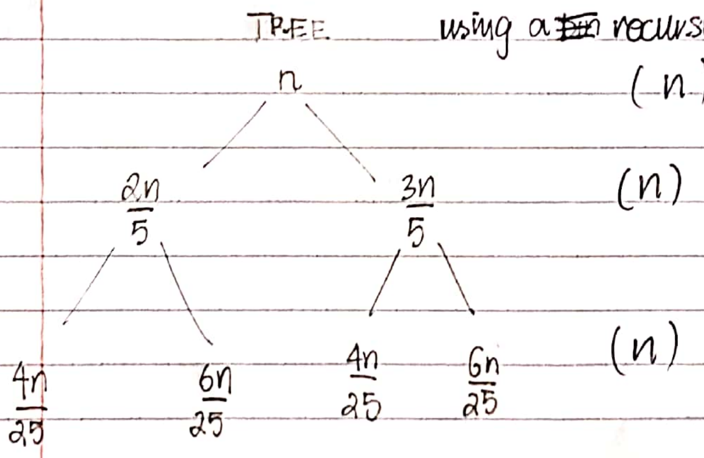
$n^\wedge \log_{10}^{11} = n^{1.04}$

since $f(n)$ is polynomially larger then $n^\wedge \log_b a$

$T(n) = \Theta(2^n)$

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

using a recursion tree.



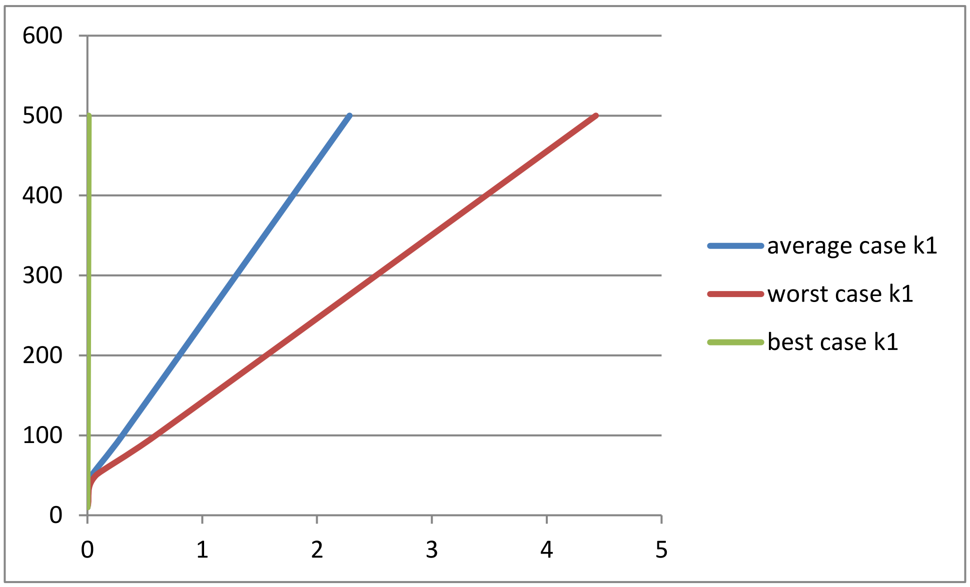
$\therefore T(n) = \Theta(n \log n)$.

Huzaifa Hashim

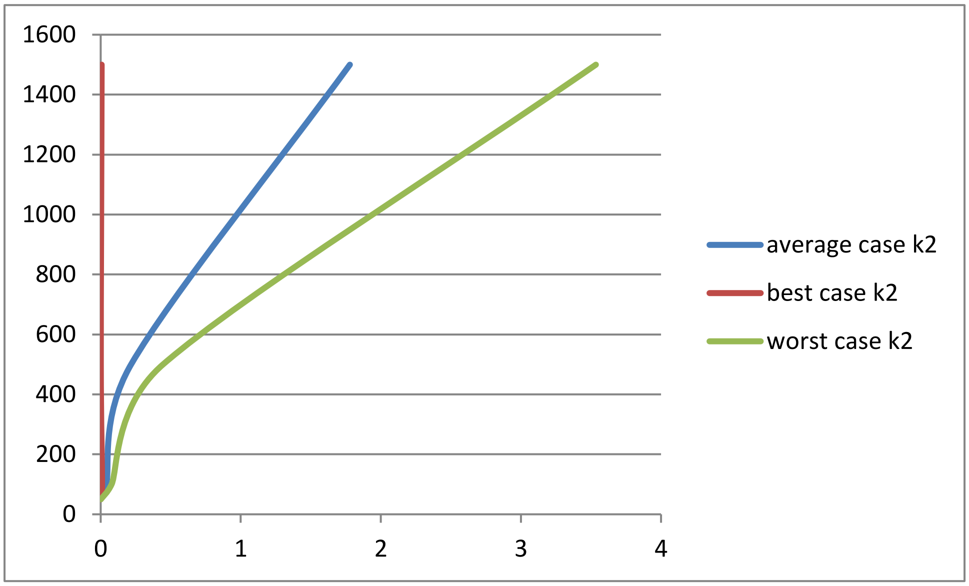
Algorithms and Data Structures
CH08-320201
Homework 2

Problem 2.1
Merge Sort

a. Please refer to program file
b. Please note that these do not include the plots from HW 1 since my algorithm was incorrect for HW1



Input(n)	**Subseries(k)**	**Case**	**Time**
100	10	Average	0.001
		Best	0.001
		Worst	0.001
500	50	Average	0.039
		Best	0.001
		Worst	0.075
1000	100	Average	0.305
		Best	0.003
		Worst	0.597
2000	500	Average	2.283
		Best	0.011
		Worst	4.428



Input(n)	Subseries(k)	Case	Time
100	50	Average	0.001
		Best	0.001
		Worst	0.001
500	100	Average	0.04
		Best	0.01
		Worst	0.078
1000	500	Average	0.217
		Best	0.001
		Worst	0.445
2000	1500	Average	1.779
		Best	0.007
		Worst	3.535

c. Insertion sort takes *Θ*(k^2) time per *k*-element list in the worst case. Therefore, sorting *n/k* lists of k elements each takes *Θ*(k^2 *n/k*)=*Θ*(*nk*) worst-case time. In theory, the merge-insertion sort algorithm improves on the overall time and achieves *Θ(n log(n/k))* by merging lists, and then merging pairwise lists until there is just one list. A pairwise merging would require *Θ(n)* work at each sub-level.

In theory this algorithm works on *n* elements and even if they are partitioned in different subsequences, it finishes with 1 sorted list with *n* elements which amounts to *log(n/k)* levels. Therefore, in theory and possibly not my code, the total time required for sorting becomes *Θ(nlog(n/k)*

d. The value for *k* should be the largest length of integers on which insertion sort is faster than merge sort.