Algorithms and Data Structures

Arnav Singh

March 2, 2020

4 Sheet 4

4.1 Merge Sort

4.1.1 Implemented Merge sort in zip file check merge.cpp file

4.1.2 Graph of the computation time of the algorithm as the value of k varies

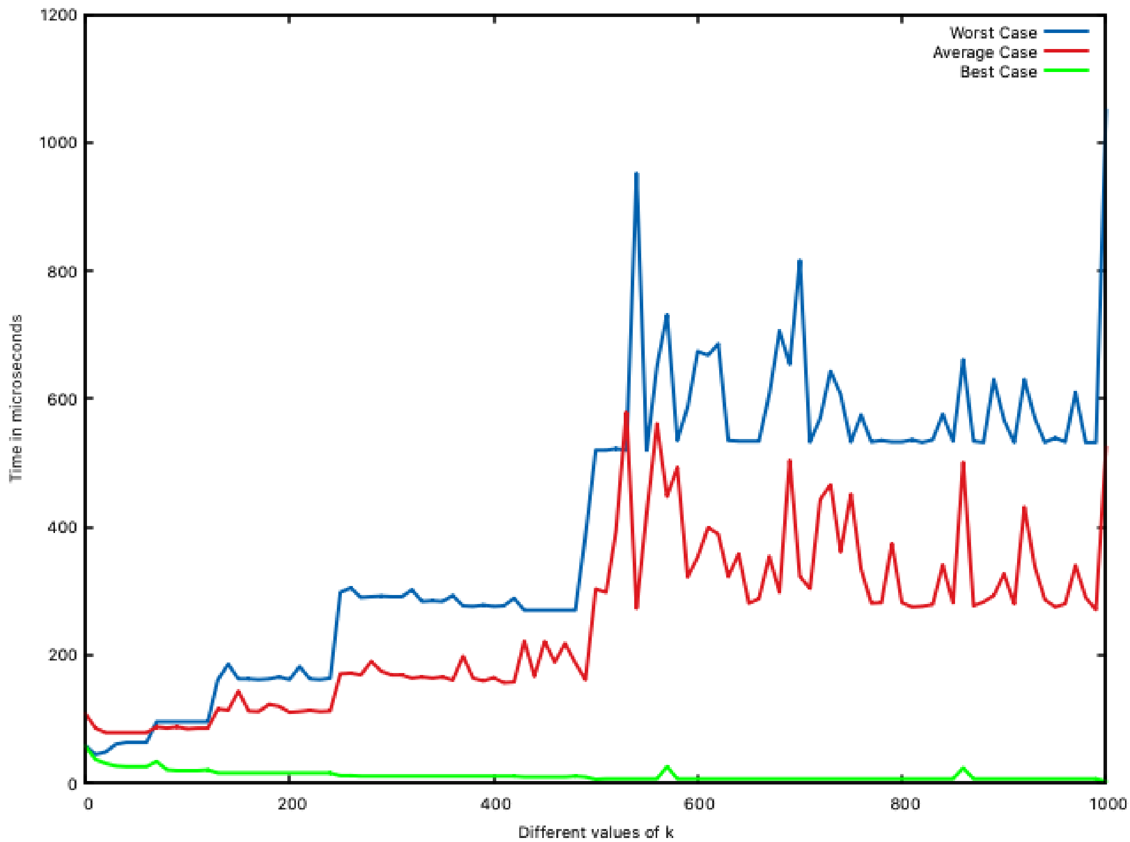

Figure 1: Plot of the computation time(microseconds) of the three cases with respect to increasing value of k

4.1.3 Interpretation of Graph

	Best	Average	Worst
Merge sort	O(nlogn)	O(nlogn)	O(nlogn)
Insertion sort	O(n)	O(n^2)	O(n^2)

For the best case, from the graph we can see that as the value of k changes the best case requires the least amount of time. This is mainly due to the fact that the best case time complexity of insertion sort is O(n), and hence when the value of k increases only insertion sort is being applied and hence we expect a much linear curve.
For the average case, the average case of merge sort is $\theta(nlogn)$ and insertion sort is $O(n^2)$, the total average complexity becomes $\Theta(\frac{n}{k}log\frac{n}{k} + k^2)$, which satisfies for all values of k.
For the worst case, as k becomes larger and larger the k^2 term from $\Theta(\frac{n}{k}log\frac{n}{k} + k^2)$ becomes much more significant and hence insertion sort is applied hence the time complexity is nothing but $O(n^2)$.

4.1.4 Bonus:

BONUS IS IN THE BONUS SEPARATE FILE.

4.2 Recurrences

4.2.1 $T(n) = 36T(\frac{n}{6}) + 2n$

The given recurrence is in the form of $T(n) = aT(\frac{n}{b}) + f(n)$ and hence we can use the Master method for solving this. Here we have $a = 36$ and $b = 6$

$$n^{\log_b a}{=}n^2 \text{ and } f(n) = 2n$$

This is Case 1 as $f(n)$ is polynomially smaller than n^2.
$f(n) = O(n^{2-e})$ for $e = 1$, Thus by Master Theorem, $T(n) = \Theta(n^2)$.

4.2.2 $T(n) = 5T(\frac{n}{3}) + 17n^{1.2}$

The above recurrence is also in the form of $T(n) = aT(\frac{n}{b}) + f(n)$ and hence we will apply the Master Method for solving this. Here our $a = 5$ and $b = 3$.

If the recurrence is of the form $T(n) = aT(\frac{n}{b}) + \Theta(n^k log^p n)$, where $a \geq 1$, $b > 1$, $k \geq 0$ and p is a real number, then:

1) If $a > b^k$, then $T(n) = \Theta(n^{log_b^a})$
2) If $a = b^k$
   a. If $p > -1$, then $T(n) = \Theta(n^{log_b^a} log^{p+1} n)$
   b. If $p = -1$, then $T(n) = \Theta(n^{log_b^a} loglogn)$
   c. If $p < -1$, then $T(n) = \Theta(n^{log_b^a})$
3) If $a < b^k$
   a. If $p \geq 0$, then $T(n) = \Theta(n^k log^p n)$
   b. If $p < 0$, then $T(n) = O(n^k)$

Figure 2: Extended Master Theorem

This is also Case 1 as $f(n)$ is polynomially smaller than $n^{1.47}$
$f(n) = O(n^{1.47-e)}$for $e = 0.27$
Thus by Master Theorem, $T(n) = \Theta(n^{1.47}) = \Theta(n^{\log_3 5})$

4.2.3 $T(n) = 12T(\frac{n}{2}) + n^2 \log n$

This question falls in the case of the extended master theorem. I have referred to this question from Geeks4Geeks:
`https://www.geeksforgeeks.org/advanced-master-theorem-for-divide-and-conquer-r`
If the recurrence is in the form $T(n) = aT(\frac{n}{b}) + n^k log^p n$. According to figure 2 above,we can determine the time complexity by evaluating the above conditions.
Here our $a = 12, b = 2, k = 2, p = 1$ The first condition is satisfied since $12 > 2^2$ hence according to the reference, our Time Complexity $T(n) = \Theta(n^{\log_2 12)}$

4.2.4 $T(n) = 3T(\frac{n}{5}) + T(\frac{n}{2}) + 2^n$

There are a number of ways to solve this problem. As T(n) is increasing and tends to ∞ , $T(\frac{n}{2})$ grows faster than $T(\frac{n}{5})$, which means that T(n/2) is much more significant than T(n/5) hence we can simplify the expression down to $T(n) = 4T(\frac{n}{2}) + 2^n$ We can now use the master theorem.
$a = 4$ and $b = 2$ and $f(n) = 2^n$

$n^{\log_b a}$=n^2 and $f(n) = 2^n$
This is Case 3 as $f(n)$ is polynomially larger than 2^n
hence $T(n) = \Theta(f(n))$=$\Theta(2^n)$....BONUS IS IN THE BONUS FILE